# TOWARDS HARNESSING COMPUTATIONAL WORKFLOW PROVENANCE FOR EXPERIMENT REPORTING

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2016

By
Pinar Alper
School of Computer Science

# Contents

# List of Tables

# List of Figures

# Abstract

We're witnessing the era of *Data-Oriented Science*, where investigations routinely involve computational data analysis. The research lifecycle has now become more elaborate to support the sharing and re-use of scientific data. To establish the veracity of shared data, scientific communities aim for systematising 1) the process of analysing data, and, 2) the reporting of analyses and results. Scientific workflows are a prominent mechanism for systematising analyses by encoding them as automated processes and documenting process executions with *Workflow Provenance*. Meanwhile, systematic reporting calls for discipline-specific *Experimental Metadata* to be provided outlining the context of data analysis such as source/reference datasets and community resources used, analytical methods and their parameter settings. A natural expectation would be that investigations, which adopt a systematic, workflow-based approach to the analysis can be advantageous at the time of reporting. This premise holds weakly. While workflow provenance supports streamlined enactment of analyses, their auditability and verifiability, we conjecture that it has limited contribution to reporting.

This dissertation focuses on eliciting the apparent disconnect of Workflow Provenance and Experimental Metadata as the *provenance gap*. We identify *complexity*, *mixed granularity*, and *genericity* as characteristics of workflow provenance that underlie this gap. In response we develop techniques for provenance *abstraction*, *analysis* and *annotation*. We argue that workflow provenance is accompanied with implicit information, that can be made explicit to inform these techniques. Through empirical evidence we show that workflow steps have common functional characteristics, which we capture in a taxonomy of Workflow Motifs. We show how formally defined Graph Transformations can exploit Motifs to identify causes of complexity in workflows and *abstract* them to structurally simpler forms. We build on insight from prior research to show how execution and provenance collection behaviour of a workflow system can anticipate the granularity characteristics of provenance. We provide declarative anticipatory rules for the *static-analysis* of workflows of the Taverna system. We observe that scientific context is often available in embedded form in data and argue that data can be lifted to become metadata by discipline-specific metadata extractors. We outline a framework, that can be plugged with extractors and provide operators that encapsulate generic procedures to *annotate* workflow provenance. We implement our techniques with technology-independent provenance models and we showcase their benefit using real-world workflows.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

# Acknowledgements

First and foremost I would like to thank my supervisor Prof Carole Goble. By accepting me as a split-site Phd, Carole has turned a distant possibility into a reality and has committed four years of her life to weekly consultations with a chronic worrier. Not only did she help me fight the worries, to the degree possible, she provided excellent supervision with honest and timely feedback on my work.

The second big thanks go to Khalid Belhajjame for his unwavering support throughout these years. Khalid's feedback on my work, his reviews of endless paper and chapter drafts are just a few of his generosities I shall thank.

I would like to thank e-Science lab members at Manchester for their support on all things Taverna-related. Without help from Stian Soiland-Reyes, Alan Williams, Donal Fellows and Finn Bacall it would have taken longer to figure out issues encountered with the Taverna APIs or the myExperiment interfaces. I should also thank Shoaib Sufi for coordinating my communication with the team.

Melanie Price deserves a special thanks for facilitating communication with Carole, for arranging all sorts of logistics and for being a proxy me whenever I could not be in Manchester in person.

I should also thank fellow Phd student Kristian Garza and former student, now graduate Matthew Gamble for sharing their experience throughout the process.

Finally, this would not have been possible without the support of my family. I would like to thank Koray for his constant support and encouragement. My mother Nesrin literally made it possible by overtaking most of my parental duties during the write-up period. And, thanks to my daughter Nil, for her patience and her ever-present childhood energy, which kept me going at difficult times.

# Chapter 1

# Introduction

In this thesis we focus on categories and roles of provenance information within the lifecycle of workflow-based scientific analyses. We take the first step in identifying similarities and differences between provenance as Experimental Metadata and Workflow Provenance collected from trailed workflow executions. We identify *genericity*, *complexity* and *mixed-granularity* as characteristics of Workflow Provenance that block the exploitation this information for the creation of Experimental Metadata. We report four investigations aimed to tackle the identified characteristics. As a foundation, we describe an empirical analysis of workflows that reveals common functional characteristics of computations that workflows embody. We outline graph abstraction techniques that exploit functional characteristics of computations to reduce workflow *complexity*. We outline a rule-based static-analysis technique to check whether workflows would result in *mixed-granularity* provenance upon execution. Finally we describe a process-oriented approach that also exploits functional characteristics of computations to decorate *generic* provenance with domain-specific annotations.

## 1.1 Data-Oriented Science

Computing is transforming the practice of science. The so-called "Fourth Paradigm of scientific research" [HTT09] refers to the current era, where scientists utilise computational tools and technologies to manage, share, federate, analyse, visualise data to underpin scientific findings. Within this dissertation we use the term *Data-Oriented Science* to refer to this scientific practice[1]. The objective in *Data-Oriented Science*

[1]"Data Science" and "Data-Intensive Science" are two other terms commonly used in literature. We refrained from using them as they often bear the following connotations. "Data Science" is commonly

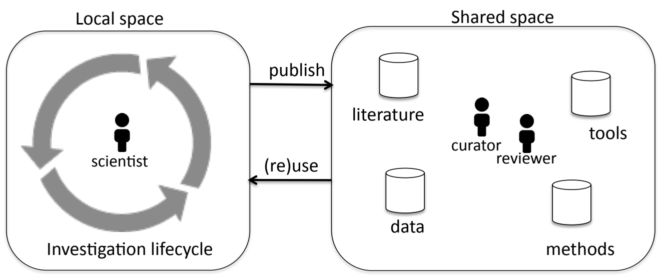


Figure 1.1: Sharing and Re-use of Scientific Work-Products

is to create a richer research ecosystem in which emphasis is given not only to the build-up of scientific knowledge, but also to the build-up and dissemination of other work-products of research such as data, protocols, models and tools. The value proposition of this new approach is that the pace and the quality of science can be improved through re-use of work-products, most notably data, from previous studies. Real-life emergencies such as disease outbreaks [LHL13] or accelerating discoveries in cancer treatment [Gob11] are the most compelling cases for data-oriented science. Beyond these examples, data-oriented and computation assisted exploration have become an inseparable part of science in all fields and at all scales.

Traditionally, research has been seen as a "value-chain of activities" [PMBVdS10], that lead to the generation of knowledge disseminated via the scholarly publication. In the Data-Oriented science view, there are multiple value-chains, not just for scholarly publications, but also for other outputs of research. Value-chains are created through publishing and (re)use as depicted in Figure 1.1. The set of work-products generated in data-oriented investigations can be viewed in two spaces; the local and the shared. All products generated during the course of an investigation make-up the local space, whereas selected products made available as resources in dedicated (often distributed) repositories make up the shared space. Scientists are both the supplier and ultimate consumer of shared resources. An important objective for Data-Oriented Science, then, is the necessity to systematically support value-chains for data, tools and other work-products similar to scholarly publications.

The paradigm of building research on shared products brings challenges in addition to its benefits. One challenge lies in *resource (re)use*. In recent years there has

used to refer to extraction of new information from raw data, typically using data mining techniques. Whereas "Data-Intensive Science" is commonly used to refer to analyses performed over very large (terabyte/petabyte scale) datasets. We opted for "Data-Oriented Science" as we wanted to avoid any implication over the analysis techniques or data size.

been a sharp increase in the number of web-accessible scientific resources. The scientists' challenge is in finding relevant resources, i.e. data and tools, and accessing and integrating them in a way to serve the purpose of their investigation. In this context **Scientific Workflow Systems** have emerged as a solution for systematising resource exploitation. Using workflows scientists weave resources into analysis pipelines, comprised of data analysis activities connected by dataflow dependencies. An example workflow that is from the Biomedical informatics domain and that has been developed using the Taverna system [MSRO$^+$10] is given in Figure 1.2. This pipeline [Min13], is part of a larger study [BZG$^+$15], which combines publicly available genomic and epigenetic datasets to better understand molecular processes involved in Huntington's Disease. The workflow in Figure 1.2 decorates a given list of genes with the biomedical concepts that they're associated with, such as a molecular processes or diseases. More specifically, this is done by invoking web services providing access to a Biomedical literature mining tool called Anni [JSV$^+$08]. Alongside service invocations the workflow involves several other steps needed for data adaptation. These steps transform (extract, re-format) results from one service and feed it to the follow-on service.

Scientific workflows systems have enjoyed notable adoption in disciplines such as Genomics, Astronomy and Environmental sciences. The benefits of workflows are particularly in the following:

- They **ease exploitation of heterogeneous resources** by providing a readily available and extensible resource-access/client infrastructure. E.g. Taverna requires minimal (near zero) effort when introducing external resources with standard access interfaces, such as web services, into pipelines. In addition workflows also ease the exploitation of local resources such as scripts and command line tools. E.g. The adapter steps in Figure 1.2 come from Taverna's local library of script based adapters.

- They **bring rigour to a data oriented investigation** by capturing an explicit and runnable codification of the analysis process. This is of crucial importance when scientists need to establish and explore a parameter space for an analysis, where they (re)run workflows with changing input datasets or parameters and later compare and contrast these results.

- They **provide transparency and efficiency into an analysis by recording provenance from workflow executions**. Provenance in this context corresponds to an execution trail documenting input datasets, the step-wise invocations of analysis

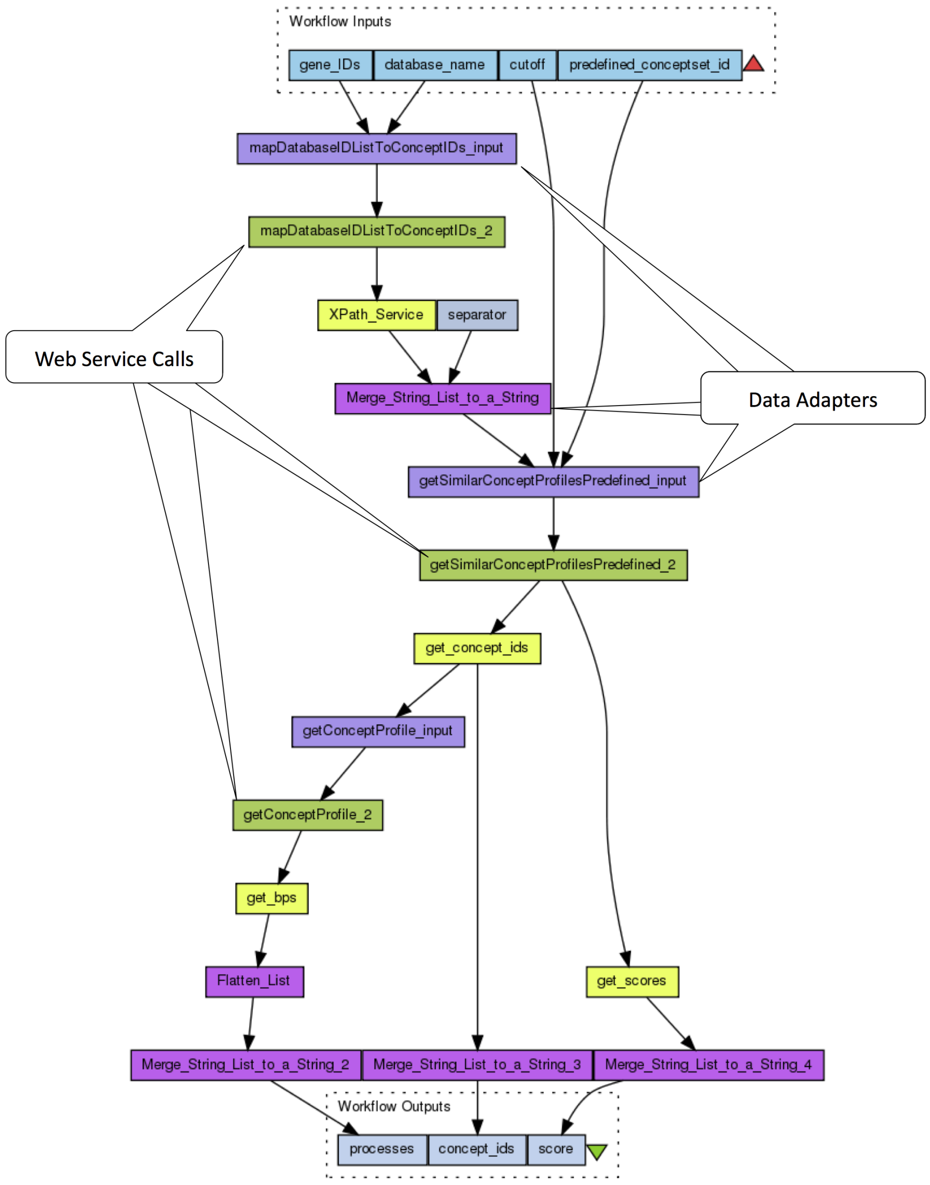

Workflow Inputs
gene_IDs
database_name
cutoff
predefined_conceptset_id
mapDatabaseIDListToConceptIDs_input
mapDatabaseIDListToConceptIDs_2
XPath_Service
separator
Web Service Calls
Merge_String_List_to_a_String
Data Adapters
getSimilarConceptProfilesPredefined_input
getSimilarConceptProfilesPredefined_2
get_concept_ids
getConceptProfile_input
getConceptProfile_2
get_bps
Flatten_List
get_scores
Merge_String_List_to_a_String_2
Merge_String_List_to_a_String_3
Merge_String_List_to_a_String_4
Workflow Outputs
processes
concept_ids
score

Figure 1.2: Workflow for Annotation of Genes with Associated Concepts

activities and the intermediary and final analysis results. Common use for these traces is for debugging workflows or to enable an external reviewer to audit the analysis to confirm that it has occurred as claimed by the scientist. Provenance can also be used to make runs of workflows more efficient by re-using results selected from previous runs.

Another challenge of sustaining value-chains lies in *publishing*. Recent years have witnessed a surge in wider data sharing practices where data coming from a broad range of disciplines is shared through repositories in a manner inclusive of all sizes (big or small) and all publishers (from major databases to single labs) [nat15]. In areas such as Medicine there are established practices for the reporting, the CONSORT initiative's reporting guidelines [SAM10] or the ADaM model for structuring results of clinical trials [Con16] are examples. A similar attempt at establishing practices is now underway in other domains, most notably in life sciences. Wider sharing practices have brought the responsibility of **Experiment Reporting** [TFS+08] to scientists. For reporting, scientists are tasked with

- the selection of data subsets to be published from a pool of results obtained through various parameter explorations, trials and re-runs
- the creation of rich experimental metadata describing the data's provenance and the context in which the data is obtained.

This category of provenance, as experimental metadata, is considered crucial information that helps other scientists to interpret data, understand the background processes, reproduce results and do new research with the data [GST+02]. Experimental metadata is expected to describe, amongst other aspects:

- the sources used, where sources could be both physical artefacts such as tissue samples or digital artefacts such as data files, or external databases.
- the context in which the results were generated, which corresponds to experimental parameters or configurations.
- the methods used, where methods could both be physical such as lab protocols, or computational such as workflows.

Furthermore, such descriptions are expected to be in a structured, machine processable form to allow precise accumulation, cross-linking and search in data repositories. In current practice, experimental metadata is created by a manual curation

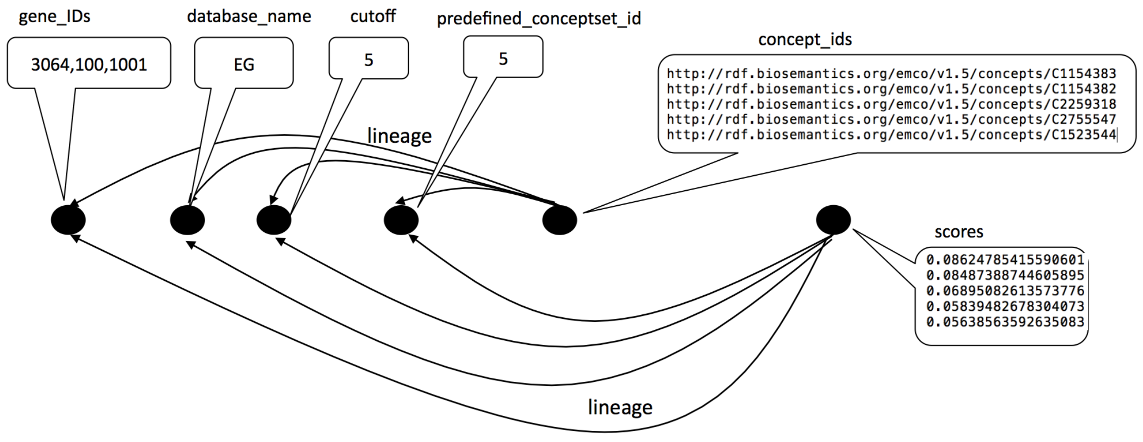

gene_IDs
3064,100,1001
database_name
EG
cutoff
5
predefined_conceptset_id
5
concept_ids
http://rdf.biosemantics.org/emco/v1.5/concepts/C1154383
http://rdf.biosemantics.org/emco/v1.5/concepts/C1154382
http://rdf.biosemantics.org/emco/v1.5/concepts/C2259318
http://rdf.biosemantics.org/emco/v1.5/concepts/C2755547
http://rdf.biosemantics.org/emco/v1.5/concepts/C1523544
lineage
scores
0.08624785415590601
0.08487388744605895
0.06895082613573776
0.05839482678304073
0.05638563592635083
lineage

Figure 1.3: Input and output data values for a particular execution of Huntington's annotation pipeline.

process by scientists. To assist scientists in curation, scientific communities have developed Reporting Guidelines [TFS+08][SRSF+12], Domain-Specific Vocabularies [FAJS05][WDG+15], and Annotation Tools [WOH+12] [SKAC14]. Repositories such as GBIF in environmental sciences [gbi15], GigaDB and Metabolights in biological sciences [SZE+14][HSC+13] are accumulation points of scientific data and along with metadata created with aforementioned models, tools and vocabularies. As of today the majority of experimental metadata in repositories describe experiments involving physical processes (based on lab-work or instrument use). However, we are observing emerging examples of reports from experiments comprised entirely of computational processes [BOHS07].

## 1.2 A Glimpse of the Provenance Gap

Against this setting a natural expectation would be that **studies, which adopt an upfront systematic approach to the computational analysis, such as those using workflows, could save-up on experiment reporting effort later at the time of publishing data**. This premise holds, albeit weakly. Let's illustrate by reviewing information available at the workflow-end and then projecting requirements from the reporting-end on to this information using the Huntington's pipeline as an example.

The first piece of information is the Workflow Descriptions as the one given in Figure 1.2. Workflow languages provide constructs for the conceptual definition of a workflow (often) as a Directed Acyclic Graph of activities, their data communication ports, and dataflow links among ports. In addition to being a conceptual description a workflow is an executable (meta)programme. Workflow systems embody execution engines that coordinate the enactment of activities and the communication of data among activities. Workflow systems are also coupled with execution trailing capabilities that can observe and record metadata about the execution. This historical record of the process, called provenance, is typically represented with standard models, such as the Open Provenance Model [MCF$^+$11] or the W3C PROV Data Model[BDG$^+$12]. Conceptually provenance information is a Directed Acyclic Graph (DAG) comprised of activities and data as nodes, and a number of well-defined relations among these nodes, such as usage and generation of data by activities, and the lineage (or descendancy) relations among data. In addition to recording provenance, workflow systems also record the values of data. A subset of provenance information obtained from one particular execution of Huntington's pipeline is given in Figure 1.3. For brevity only the data nodes and their relations are depicted. Provenance documents input and output data, which are denoted with opaque nodes in the provenance graph. In our example inputs are a number of experimental parameters: *gene_IDs*, about which the related concepts are to be obtained, the $cutoff$ threshold determining the number of concepts to be returned, the $predefined\_concept\_set\_id$ determining that what kind of concepts should be obtained and the $database\_name$ designating the kind of gene identifiers used. The outputs are results as computed by the Anni literature mining service, the $concept\_ids$ retrieved and the *scores* that quantify how strongly the retrieved concepts are related with input genes. The data values (depicted in call-outs in Figure 1.3) typically reside in a separate storage layer.

In short on the workflow-end we have **workflow descriptions, execution provenance and data values**. Projecting requirements from the reporting-end reveals the following gaps:

**Reporting origins.** For investigations based on the exploitation of external resources (databases, web services), it is critical for the credibility of results that any external resources used are cited [Dat11] as origin or contributor. For our example these would be the endpoint and version of the Anni web service and the literature database backing that service. Workflow systems record lineage among data encountered by the workflow engine, henceforth they document origin from a localised viewpoint. Workflow provenance traces contain no explicit information on the ultimate (often external) origin of the results.

**Reporting experimental context.** Contextual information must be supplied when reporting to help data's interpretation and re-use by others [RBR$^+$05] [dOSM15]. For our example contextual information would be the subject of the analysis such as the genes for which the literature has been mined, or attributes of results such as the category of concepts retrieved. As workflow provenance models data and activities in an observed process as opaque nodes, it provides no explicit information on what the data is about, or what kind of data processing occurs in an activity or whether a data node corresponds to a parameter or some result.

**Reporting data selectively.** During reporting not all enactments of an experimental processes and not all results may get reported. Scientists typically perform an analysis several times to test their experimental set-up, to calibrate their configuration parameters, or to understand the effect of parameter change. Workflow provenance comes across as crucial information here as it provides traceability among input parameters and results through lineage, this way provenance can potentially allow scientists to select results based on parameters or other results they are descend from. For instance, the scientist may want to report concept associations obtained from the Huntington's pipeline, and she may be interested in concepts that are associated with certain genes and also those that have an association score greater than a certain value. Analysing the provenance record in Figure 1.3 reveals that not all traceability requirements can be met. As the analyses are undertaken by external resources, in this case using the Anni web service, there are (lineage) relations beyond those tracked by the workflow engine. In Figure 1.3 the outputs of the workflow are five concepts that are associated with input gene ids, and the scores of association for each concept. Also note that these outputs are modelled in a coarse-grained manner as single data artefacts, and there are no explicit and fine-grained relations that link each concept to its score. The fine grained association is documented outside of workflow provenance, within results returned from a single invocation of the Anni service. To make this relation more apparent in an adhoc manner the scientist has extracted concepts and corresponding scores into same length lists, which are in turn converted into a single string representation. So the order of scores and concepts in respective strings provides an implicit traceability among the two results.

**Reporting methods.** Workflows provide the biggest benefit in reporting the method as they capture the entire computational process followed for an analysis. Meanwhile

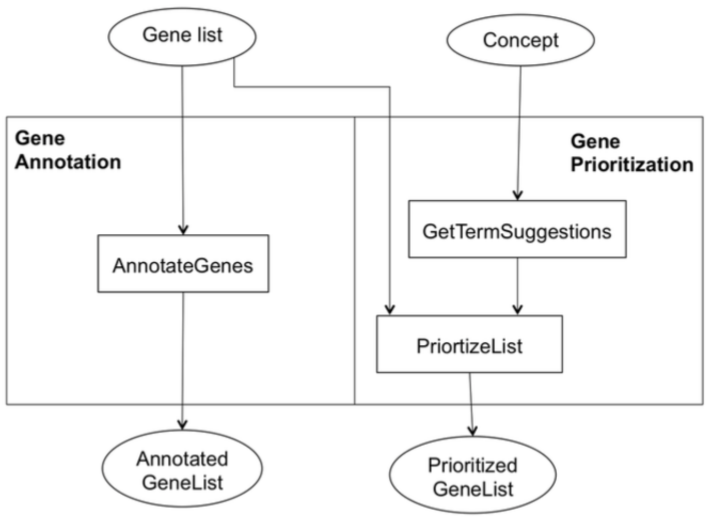


Figure 1.4: Experiment Sketch Depicting Methods used in Huntington's Disease Study

the use of workflows in reporting is not immediate. Scientists report methods by applying abstraction over workflow descriptions. Figure 1.4 depicts the "experiment sketch" [BZG+15], which is as abstraction mechanism used by scientists when reporting workflows [HWB+12]. Here the entire gene annotation pipeline in Figure 1.2 is designated as a single step, one of three major steps involved in the study, which is implemented with a comprehensive workflow involving three sub-workflows.

The above identified gaps can be attributed to known characteristics of workflow provenance. These characteristics and the existing coping mechanisms can be summarised as follows:

*Complexity of Workflows*. The primary function of workflows as integrators of (heterogeneous) resources can have a disruptive effect on their role as documenters of the scientific method followed for the analysis [GAB+12]. Workflows systematise resource integration by making explicit the effort needed to deal with resource heterogeneity, such as with adapter steps (recall from Figure 1.2). This however results in complexity, which is a previously observed characteristics of scientific workflows in literature [BCBDH08] [ABL10]. Reporting calls for eliciting the report-worthy activities of the analysis from the unimportant ones, such as adapters. A common solution adopted by scientists for elicitation is to design workflows in layers using sub-workflows, where lower layers contain the detail of resource integration, and the upper layers abstract away from it. Layered design is a best practice in scientific workflow development [HWB+12] and also instrumental in reporting the method. Meanwhile it remains a largely manual, scientist-driven activity.

*Genericity of Provenance*. In order to cater for heterogeneity in external resources and to allow for diverse scientific analyses workflow systems make minimal assumptions on the structure and type of data and the inner workings of analytical activities. This is commonly known as the black-box assumption in provenance collection [CJ10]. Due to this assumption, workflow provenance can only provide generic information, in the form of activity occurrences and the data consumed/produced. Scientists cope with genericity by relying on descriptive information available in adhoc means; such as the names of workflow elements (activities and ports), or the data values. The common mechanism for obtaining domain-specific information is by manual annotation. All workflow systems allow workflow elements to be annotated in various forms (e.g. key value pairs or semantic annotations). While annotating workflow descriptions is a manageably sized task for scientists, annotating data artefacts created from runs is a prohibitively big task due to iterative analyses for parameter explorations.

*Granularity Discrepancies in Provenance.* In order for scientists to exploit provenance for data selection it is crucial that elements that make up an experiment's design, i.e. the parameters and the activities configured with those parameters, get modelled in provenance with suitable granularities. As reviewed in Chapter 2, most workflow languages provide constructs to enable such modelling within a workflow description. In the context of this dissertation we refer this capability as a workflow system's **support for factorial design** (Outlined in Section 2.5.3 ). With such features analyses can be repetitively executed with differing parameters and over differing datasets. A crucial requirement for having traceability is for factorial design to be reflected in workflow execution provenance. This is challenged by the provenance architectures of workflow systems and the realities of workflow development in a heterogeneous resource environment:

- While factorial design may be encoded in a workflow description, it may not get reflected to execution provenance fully. Provenance collection in workflow systems is performed by add-on components that record processes through external observation. Due to this decoupled design, the granularity of observation (of data or activities) is a factor affecting the faithful reflection of factorial design in provenance. In provenance research there have been case/system-specific definitions of this as fine-grained [MPB10] or collection-oriented provenance [BMWL07]. Meanwhile there is no systematic understanding on what it means for workflow provenance to represent factorial design.

- Considering that a workflow system supports constructs for factorial design, which in turn get faithfully reflected to provenance, this is not a guarantee of traceability among parameters and results. When a study makes use of external resources, concerns such as reducing resource access cost, or the nature of resource interfaces may lead to coarse-grained provenance (as illustrated with the gene annotation workflow's outputs). As a result fine-grained parameter-to-result or result-to-result traceability may not be recorded explicitly. Also, when encoding data adaptation among analytical steps scientists may inadvertently create workflow designs that would generate coarse grained provenance.

The main idea pursued in this thesis is that workflows and provenance collected from workflow executions can be treated to address the above limitations surfacing in the context of experiment reporting. We investigate and develop techniques towards tackling genericity, complexity and granularity-discrepancy aspects. Our approach is:

- inspired by the current practice of scientists in using annotation and abstraction as solutions towards the respective shortcomings.

- is built on additional information that comes from 1) assumptions that scientific workflows in their current empirically observable state allow us to make 2) the exploitation of the well-defined execution behaviour, and predictable provenance collection behaviour of Taverna workflow system.

We briefly discuss these assumptions in the following section.

## 1.3 A Grey-Box Provenance Approach

A prerequisite to the development of computational solutions to address the short comings of provenance is to have extra information on characteristics of provenance elements (activities and data) and their granularities. As discussed earlier within standard workflow provenance there is no indication on the scientific characteristics and/or the significance of computations and the data, henceforth in the current workflow tooling this information needs to be manually added (with annotation) or manually elicited (by abstraction).

Our dissertation research can be characterised as a *grey-box* approach as it brings the following additional information onto provenance:

- Based on extensive empirical analysis, we deduce that computations within scientific workflow activities are not entirely arbitrary and a high-level classification of activity functionality is possible [GAB$^+$14]. We put this extra information to use in three ways.
  - we use activity function as an indicator of (in)significance of an activity from a scientific perspective. We exploit this information for workflow abstraction.
  - we use activity function as an indicator of scientifically significant data processing, which corresponds to a context binding parameters with result data. We exploit this information for provenance annotation. The input and output data values of such activities can be used to build domain specific provenance annotations to describe the context explicitly.
  - we use activity function for certain categories of adapter activities as an indicator of a certain kind of lineage designating data derivation by value-copying. We exploit this information to expand the reach of domain specific annotations to close derivatives of data (created by value-copying) in a workflow provenance trace.
- Based on a systematic review of existing workflow systems in terms of their support for factorial design (presented in Chapter 2) we identify that Taverna provides strong support for factorial design and reflects it faithfully to execution provenance. We identify the well-defined behaviour of Taverna in executing workflows as additional information that can be used to anticipate the granularity characteristics of provenance traces that a workflow's execution will produce.

## 1.4 Research Objectives

The overall aim of this dissertation is to identify the shortcomings of workflow provenance in the context of experiment reporting and to devise techniques to tackle these shortcomings. More specifically our objectives are:

O1 To understand the requirements that the emerging practice of experiment reporting brings in terms of the provenance information as experimental metadata accompanying shared datasets.

O2 To revisit the characteristics of workflow provenance in light of the requirements identified in **O1** and to identify points of overlap as opportunities and the points of mismatch as challenges for research within the threads of workflow provenance Annotation, Analysis and Abstraction (AAA).

O3 To review the existing practice of scientists in workflow development to understand whether there exist patterns that can be generalised to break the black-box assumption over workflow activities.

O4 To devise techniques in AAA that exploit the information revealed in **O3**.

O5 To implement solution techniques using technology-independent workflow and provenance representation models.

O6 To assess the effectiveness of techniques with real-world workflows.

O7 To understand how our solutions compare to existing and related work.

We revisit these objectives in the final chapter of the dissertation (Chapter 8) to asses how we have met them.

## 1.5 Research Hypotheses and Methods

We will now outline our research hypotheses and our methods. Our first hypothesis is concerned with determining the points of overlap and the points of difference between the two categories of provenance: 1) experimental metadata required for reporting and 2) provenance collected from workflow-based analyses.

H1 **Workflow provenance is a potential information source that can support information requirements of experiment reporting.**

Based on findings of research in response to this hypothesis we have observed overlaps that can warrant further research focusing on differences/gaps. The follow-on hypothesis is geared to test whether workflows in their current state allow us to go beyond the black-box view of workflow activities.

H2 **Computations within scientific workflows are not entirely arbitrary; existing practices in workflow development exhibit common patterns that would allows us to obtain a categorisation of functions undertaken by activities (most specifically data adapters).**

Our third hypothesis is geared to test whether the information available in the form of activity categorisations can be used to tackle the complexity of workflow provenance.

H3 **We can abstract workflow descriptions, by using activity functional categorisations as an indicator of activity significance.**

With our fourth hypothesis we investigate how information on the execution behaviour of a workflow system can be used to tackle granularity discrepancies of workflow provenance.

H4 **We can analyse Taverna workflows to anticipate the structure of the provenance traces that their execution will produce. Specifically the analysis can reveal whether or not the granularity characteristics of provenance allows for traceability between designated input parameters and results.**

Our final hypothesis has driven our research on tackling genericity of workflow provenance. We investigate whether context within a workflow-based analysis can be made explicit with annotations and whether activity characteristics can inform the annotation process.

H5 **The experimental context found in an implicit form in workflow descriptions, execution provenance and data values can be turned into explicit form as annotations over data. This task of annotation can be informed by activity characteristics and can be separated into domain specific and generic layers, where the generic layer can be provided as a framework, which when plugged with the domain-specific layer can create annotations on experimental context.**

We use the following methods in research, the mapping of methods to corresponding hypotheses is given in Table 1.1:

- **Comparative Literature Surveys**: This dissertation provides systematic and comparative surveys of literature for two areas; first for provenance support in scientific data processing systems and secondly on provenance abstraction. These two areas contain multiple closely related efforts allowing comparison. On the other hand our surveys in the area of provenance annotation, workflow analysis are system-by system descriptions of related work, as work in these areas are highly heterogeneous and distinct.

- **Theoretical modelling**: We exploit established theoretical approaches to ground our work. Specifically; we model our solutions in Algebraic Graph Transformations for workflow abstraction, we use a Functional notation to represent Taverna's operation, and we use Declarative Logic Programming to denote workflow analysis as a set of rules. These are supplemented with basic set-theoretic descriptions as necessary.

- **Experimental Evaluation**: We utilise experimental evaluation with real-world workflows to understand the effect of abstraction primitives and to compare abstractions generated by our framework to abstractions created by users.

- **Empirical Survey**: We empirically analyse a cohort of real-world workflows as part of our effort to characterise workflow activities. Quantitative measurements are made on observations to understand the extent and frequency of activity characterisations.

- **Case-based Assessment**: We utilise a case involving the querying of workflow provenance to select data subsets of interest. We use the case to showcase the benefits of provenance annotation and workflow analysis. We also use the case to understand the vulnerability points of our provenance annotation approach.

Table 1.1: Hypotheses and the methods used in research.

| **Method** | **H1** (base) | **H2** (analysis) | **H3** (abstraction) | **H4** (analysis) | **H5** (annotation) |
|---|---|---|---|---|---|
| Survey | ✓ | | ✓ | | |
| Theoretical Modelling | | | ✓ | ✓ | |
| Experimental Evaluation | | | ✓ | | |
| Empirical Survey | | ✓ | | | |
| Case-based Assessment | | | | ✓ | ✓ |

## 1.6 Research Contributions

We will now describe specific research activities performed, the contributions resulting from each activity. We view our research to fall under three main themes: Provenance Analysis, Abstraction and Annotation. Concepts used in discussion of research activities and their relations are depicted in Figure 1.5.

### 1.6.1 Understanding Provenance in Data-Oriented Science

In response to hypothesis H1, we have performed a review of selected experiment reports, which resulted in a set of observations on their information requirements. We have also reviewed provenance support in workflow systems to reveal their common characteristics, assumptions and provenance collection and management architectures. We have then projected requirements from experiment reports onto workflow provenance, which revealed overlaps (in support of H1), and also gaps. This projection led to the following contribution:

C1.1 **A formulation of the gap between provenance as experimental metadata and provenance from workflows.**

This formulation is in the form of a set of requirements from experiment reports that highlight a set of characteristics of workflow provenance. We have then used this formulation to assess the level of support that state of the art workflow systems (those with provenance tracking features) provide for experiment reporting. This has resulted in the following contribution:

C1.2 **A comparative survey of workflow systems assessing fitness of their provenance for use in reporting.**

### 1.6.2 Challenging the Black-Box Assumption with Analysis

We have approached hypothesis H2 through a manual analysis of workflow descriptions to check whether they involve recurring and recognisable (or non-arbitrary) data processing. We performed an empirical survey that encompassed 260 workflows from 4 systems and 10 domains. The survey showed that activities within workflows can be characterised at a high-level in terms of their functionality. The Survey also resulted in a categorisation of non-functional, implementation-related patterns of workflow development. We called this characterisation *Workflow Motifs* and represented it as a taxonomy of concepts in an OWL 2 [HKP$^+$09] ontology, which makes up our second contribution:

C2 **Workflow Motif taxonomy.**

Our survey has also produced the quantification of motif occurrences in workflows, which makes up our third contribution:

C3 **Empirical quantification of Motif occurrences in scientific workflows.**

These results revealed that majority of activities in workflows perform data adaptation, identifying them as the prime contributor to workflow complexity. Our survey has also unearthed the current workflow design practices that scientists use to manage complexity.

### 1.6.3 Tackling Workflow Complexity with Abstraction

The extent of adapter activities revealed in our empirical survey allowed us to formulate workflow abstraction (hypothesis H3) as the elimination of adapters from workflow descriptions. As provenance abstraction is a recent and active area of research we have performed a comparative survey of existing approaches. We provide a blueprint that organises the common elements in computationally-assisted provenance abstraction approaches. We have used this blueprint to comparatively analyse state of the art systems, which has resulted in our fourth contribution:

C4 **Abstraction Systems Survey.**

The blueprint identifies as common elements the graph-based modelling of provenance, and the use of policies 1) to drive abstraction and 2) to understand and control the impact of abstraction on the integrity of resulting provenance graphs. In our approach we exploit Graph Transformation as a methodology and provide three declaratively-specified transformations, we call *primitives* for abstracting workflows. Primitives are provided as part of a framework, which comprises our fifth contribution:

C5 **Workflow Summaries Framework.**

This is a prototypical framework which is configured by *abstraction policies* that identify activities with motifs and prescribe how they should be abstracted with primitives. Outputs of abstraction are called *workflow summaries*. Our framework is independent from any particular workflow technology as it adopts the abstract Wfdesc model [BZG+15] for representing workflows and workflow summaries. We assess our approach using real-world Taverna workflows in a two part evaluation process. We first compare the abstractive effect of primitives by comparing the reduction they create in workflows' structural elements. We also use scientists's manual reporting practices as groundtruth and assess the abstraction accuracy of a set of default abstraction policies that encode common sense reporting heuristics.

### 1.6.4 Understanding the Requirements in Data Reporting

In order to reveal specific challenges in the use of provenance for reporting data we performed a case study that involved provenance queries from literature and provenance from Taverna system. This study highlighted the characteristics of workflow provenance in general and Taverna provenance in particular that hamper its use for data selection. More specifically we observed that genericity of provenance causes provenance queries to be implemented partially or in an adhoc manner. On the other hand we observed that mixed granularity traces can have a negative impact on provenance queries and rendering provenance minimally useful for data selection.

### 1.6.5 Tackling Granularity Discrepancies with Analysis

In response to hypothesis H4 we exploit Taverna's well-defined execution behaviour and faithful representation of factorial design in provenance to detect granularity discrepancies before they occur. We build on a prior insight into Taverna provenance [MPB10] to define a set of *static analysis rules* to detect broken factorial design in Taverna workflows, which makes up our sixth contribution:

C6 **Taverna Workflow Analysis Rules.**

We use Datalog to implement rules, where a Taverna workflow is represented as an extensional database of facts; and where rules infer intentional facts on the granularities of data and activities anticipated for the execution of that workflow as.

### 1.6.6 Tackling Provenance Genericity with Annotation

Our observations on Motifs has prompted us to define annotation as a process that mirrors the process of the scientific workflow, where we seek to provision annotations from traces of scientifically significant computations and we propagate annotations whenever data adaptation occurs. In response to hypothesis H5 we have developed a framework, named *Label*Flow, which embodies a set of labelling operators that encapsulate this behaviour, which comprises our final contribution:

C7 **Provenance Labelling Operators.**

Operators embody generic procedures that perform the middle man duty of accessing PROV compliant traces of scientific workflows, provision metadata from data

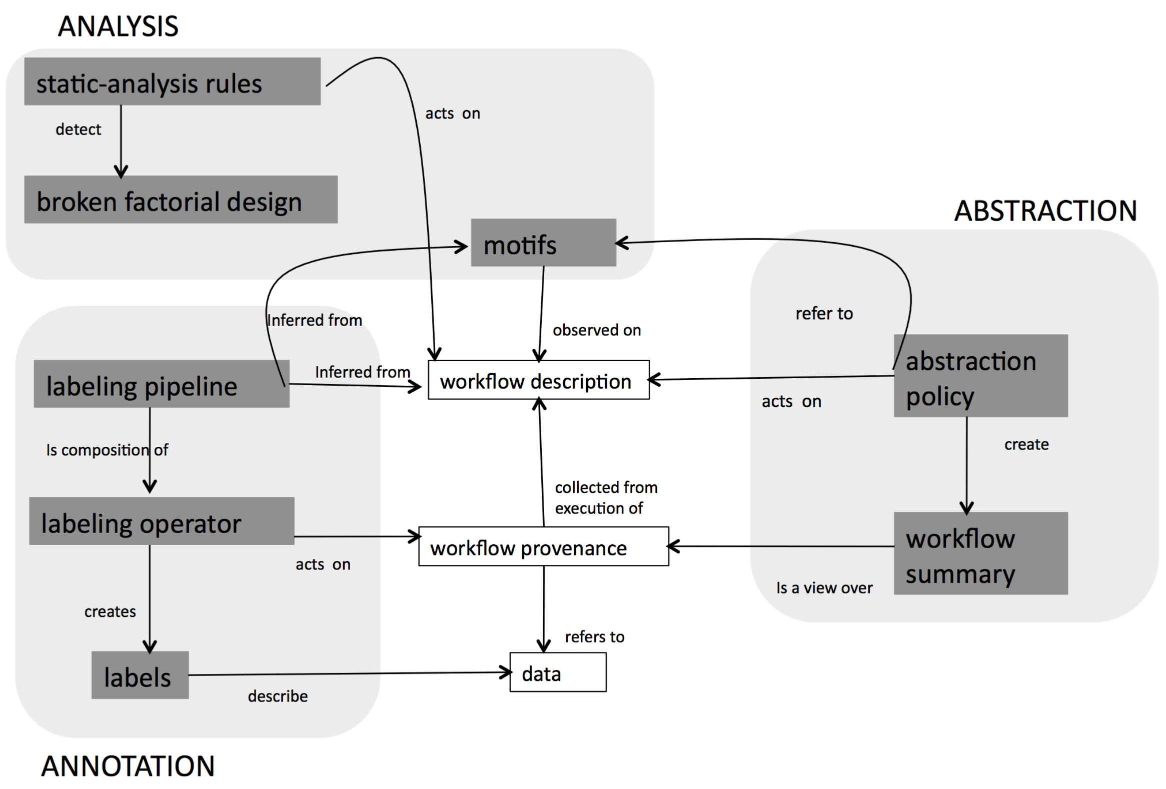


Figure 1.5: Research Threads and Proposed Concepts in each Thread.

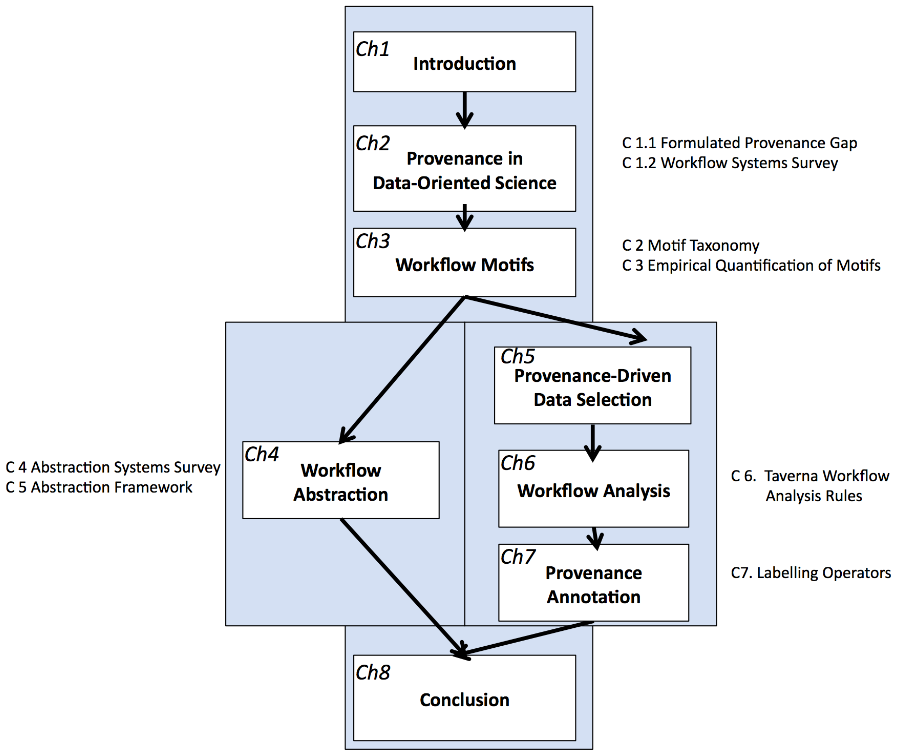

Ch1
Introduction
Ch2
Provenance in Data-Oriented Science
C 1.1 Formulated Provenance Gap
C 1.2 Workflow Systems Survey
Ch3
Workflow Motifs
C 2 Motif Taxonomy
C 3 Empirical Quantification of Motifs
Ch5
Provenance-Driven Data Selection
C 4 Abstraction Systems Survey
C 5 Abstraction Framework
Ch4
Workflow Abstraction
Ch6
Workflow Analysis
C 6. Taverna Workflow Analysis Rules
Ch7
Provenance Annotation
C7. Labelling Operators
Ch8
Conclusion

Figure 1.6: Structure of thesis.

values though the help of domain-specific functions and then decorate the PROV trace with that metadata. We represent metadata in the form of simple attribute-value pairs, we call Labels. Operators are agnostic to the content of labels, therefore they can be used to annotate traces of workflows from different domains. We exploit the structure of the scientific workflow to induce simple processes, called Labelling Pipelines that coordinate the execution of labelling operators.

## 1.7 Publications

The research described in this dissertation has led to the following peer-reviewed publications.

- D. Garijo, P. Alper, K. Belhajjame, et al. Common motifs in scientific workflows: An empirical analysis. In Proceedings of the 8th eScience Conference, pages 1-8, Chicago Illinois USA, October 2012, IEEE.
- D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble. Common motifs in scientific workflows: An empirical analysis. Future Generation

Computer Systems, 36:338351, 2014, Elsevier.

Above two papers describe our empirical analysis over workflows, which resulted in the Motif categorisation. The initial analysis, described in the first paper, has been performed over a corpus limited to the Taverna [MSRO+10] and Wings [GRK+11] workflow systems. We have later expanded the analysis to include Vistrails [CFS+06] and Galaxy [GRH+05] systems and have demonstrated how workflows can be annotated with their Motifs. Text from the second paper describing the extended analysis has been used in Chapter 3 of this dissertation.

- P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz. Enhancing and abstracting scientific workflow provenance for data publishing. In Proceedings of BigPROV, International Workshop on Managing and Querying Provenance Data at Scale, Joint with EDBT/ICDT, pages 313-318, Genoa Italy, March 2013, ACM.

This is a position paper that described our early observations on the provenance gap and outlined a preliminary research agenda. Observations on workflow provenance from this paper have been recited and expanded in Chapter 2 of this thesis. The initial research agenda proposed annotation of provenance with a restricted but set of transparent data processing activities and restrictive assumptions on data structures. In our research we have adopted for a grey-box approach in which we use workflow Motifs and assume partial transparency over activity functions.

- P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz. Small is beautiful: Summarizing scientific workflows using semantic annotations. In Proceedings of the 2nd International Congress on Big Data (BigData 2013), pages 318-425, Santa Clara, CA, USA, June 2013, IEEE.

In this paper we describe our workflow abstraction primitives informally and provide an initial evaluation with a limited workflow cohort. Text from this paper have been used in Chapter 4 of this dissertation, where we formally specify primitives and provide an extended experimental evaluation. In this extended evaluation a larger workflow cohort has been used moreover abstractions created by users in workflow designs has been used as groundtruth in assessing abstractions.

- P. Alper, C. A. Goble, and K. Belhajjame. On assisting scientific data curation in collection-based dataflows using labels. In Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science (WORKS 13), pages 716, Denver, Colorado, November 2013, ACM.

- P. Alper, K. Belhajjame C. A. Goble, and P. Karagoz. *Label*Flow: Exploiting Workflow Provenance to Surface Scientific Data Provenance. In Proceedings of the 14th International Annotation and Provenance Workshop (IPAW), pages 84-96, DLR Cologne Germany, June 2014, Springer.

Above two paper describe our provenance annotation techniques. First one introduces *Label*Flow architecture and the labelling operators. In the second paper we describe our case study on querying of provenance in the presence and absence of labels and provide a comparative assessment. Text from this latter paper has been used in Chapters 5 and 7 of this dissertation.

In addition to publications the thesis author has presented her dissertation research as an invited speaker at the "Scientific Workflows and Provenance" working group meeting of DataONE [dat15] project 2013 plenary meeting.

## 1.8 Thesis Organisation

The thesis outline is presented in Figure 1.6. Chapter 2 provides background and literature review on provenance. This chapter also presents our formulation of the provenance gap and the survey of workflow systems' in terms of their provenance support against the requirements of reporting.

Chapter 3 presents our empirical survey of scientific workflows, we present the motif taxonomy, motif occurrence statistics and discuss the ways motifs inform our provenance treatment techniques.

Chapter 4 presents our research on workflow abstraction. Here we present the comparative survey of provenance abstraction systems. We describe our algebraic graph transformation primitives and workflow abstraction framework and its evaluation.

Chapter 5 presents our case-study, introduces the queries we use to understand how provenance can be used to report data. We discuss the negative impact that genericity and mixed-granularity characteristics has to provenance query accuracy. Chapter 5 also presents the architecture overview of our solutions tackling mixed-granularity and genericity with Taverna Workflow Analysis and *Label*Flow respectively.

Chapter 6 presents our research on workflow analysis to reveal provenance granularity characteristics. Here we present Taverna's constructs to support factorial design, its execution behaviour and the implications of this behaviour on provenance structure. We present our analysis rules and illustrate their deductions with an example.

Chapter 7 presents our research on provenance annotation. We present Labelling Operators and how Labelling pipelines are generated for scientific workflows. We revisit the case study to showcase the benefits of labels and to highlight the vulnerabilities of our annotation approach.

Chapter 8 revisits the contributions presented and discusses future work.

# Chapter 2

# Provenance in Data Oriented Science

## 2.1 Chapter Introduction

In this chapter we provide the background and the primary motivation for our dissertation research. We begin in Section 2.2 by providing the basic definition for provenance and discussing its general application areas including Data-Oriented Science. In Section 2.3 we zoom into the Data-Oriented Investigation Lifecycle identified earlier in Chapter 1 to understand the role and forms of provenance in it. For the purposes of this dissertation we focus on two types of provenance from this lifecycle. One is **experiment reports**, which are distilled forms of provenance that are curated by scientists and that get published alongside the investigation's results (data). The other is **processing trails**, which gets automatically collected from computational data processing systems, most specifically scientific workflow systems. **Our dissertation research takes motivation from two observations we make on these categories of provenance:**

- There is an apparent gap between the two types of provenance. While trail type provenance (from workflows) is used to support a variety of activities during the collection and processing of data, it has limited use in publishing and the creation of experiment reports.
- Trail-type provenance from workflows has potential to be exploited for reporting.

Chapter 2 is dedicated to substantiating these observations and incrementally refining them using the data-oriented investigation lifecycle as a roadmap. From Sections 2.3.1 through to 2.3.3 we make a high-level introduction to trail-type provenance by first identifying the role of workflows in the investigation lifecycle, followed by a discussion on the fundamental motivations for trail-type provenance in workflows as well as other computational instruments. Finally we observe the provenance gap in a fine grained manner by assessing the current uses of workflow provenance in the lifecycle.

**Our dissertation research introduces techniques (described in follow-on chapters) to overcome shortcomings of workflow provenance that arise in reporting scenarios.** In order to understand these shortcomings underlying the provenance gap, in this chapter we illustrate closely the two categories of provenance in disconnect. As a prerequisite we provide the basic intuition for provenance in Section 2.4 followed by detailed analyses of Experiment Reports and Workflow Provenance in Sections 2.5 and 2.6 respectively. In Section 2.7 we formulate the gap between two categories of provenance by mapping observations on experiment reports as a set of challenges over workflow provenance.

The chapter is finalised in Section 2.8.2 with a comparative survey of the state of the art in workflow provenance, where we assess to what degree provenance from these systems meet reporting requirements. Note that this final part presents a generalised survey aiming to motivate our research. Specialised surveys that compare our research to related work are given inline in Chapters 3, 4, 6 and 7.

## 2.2 Provenance Definition

The Merriam-Webster dictionary defines provenance as "the origin or source of something; the history of ownership of a valued object, or work of art or literature" [Mer15]. Pedigree or Lineage are other common terms used synonymously with provenance. In the field of art, where the dictionary definition is drawn from, knowing the history of an object is crucial in establishing its authenticity. Provenance as such gives the interested parties an assurance [Gro05] of the object's value, and may further be an indicator of the object's quality.

In recent years research in computing has paid significant attention to the topic of provenance in the interest of tracking the origins of digital data artefacts. The World Wide Web Consortium (W3C) defines provenance as [Ge13]:

> "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness".

Another definition that is widely cited in literature on computational provenance is given by Simmhan [SPG05] identifying it as:

> "a kind of metadata that pertains to the **derivation history** of a data product starting from its **original sources**".

In the areas of Web Information Sharing, Data Archival and most notably Data-Oriented Science there are emerging requirements that call for transparency into the derivation processes of data artefacts [GGCM12]. Following from Table 2.1 we discuss a few illustrative examples. Note that provenance is a massively overloaded term [Gob02]. The overloading stems from the conflated use of the term for both provenance information itself and the diverse end-uses enabled by provenance.

Table 2.1: Provenance End-Uses

| **Area** | **Used For** |
|---|---|
| Web Information Sharing | Data Trust and Quality scores.<br>Source attribution and credit |
| Data Archival | Version tracking.<br>Custodianship and copyright tracking. |
| Data-Oriented Science | Interpretation and re-use of data.<br>Reproducibility of investigation.<br>Justification, audit of the analytical method. |

The World Wide Web has become the prime medium for sharing information. In order to make the web's deluge of information more accessible to users, a recent trend is to use aggregator services that compile information from multiple sources (e.g. blogs, tweets, news syndicates). The provenance requirement in such value-added services is to keep track of which provider supplies which part of aggregated content. This way aggregators can attribute back to sources and providers can get credit. A further application of provenance in web scenarios is to use the quality, trustworthiness and reliability scores associated with individual sources to compute likewise characteristics for aggregated content [Gol06] [GG14].

Another example area is the practice of data preservation, which has developed as a response to the fast pace of technological development in data storage formats, mediums and data access software. In preservation a data artefact goes through various revisions as a result of consequent archival actions applied to it, such as migrations, edits or digital restoration. Here capturing provenance metadata is a prescribed best-practice, and it is expected to document the migration path of data through versions, the chain of custody and associated copyright restrictions [fSDSC02].

Data-oriented science, which comprises the broad context for this dissertation, builds on cycles of sharing and (re)using datasets [GDRB13]. In order for data consumers to interpret data it is required that provenance is supplied in the form of discipline-specific know-how describing the context in which data was collected and analysed [Gob02]. Examples of contextual know-how are source characteristics (e.g. a source gene sequence obtained from mice), description of analytical protocols (e.g. use of genetic sequence alignment) and protocol configurations (e.g. version of sequence alignment tool used). Another role for provenance is for justification of the working methods of scientists. When scientists provide sufficiently detailed context one may hope that results can be independently reproduced, hence verified by interested parties [GST+02][Dav11]. However contextual metadata alone does not prove that the investigation occurred as the scientists claim it did. A common approach to verify claims is to provide a historical trail documenting the study, providing snapshots of intermediary states, values of configurations and of traceability of stepwise analytical processing [BF05][SPG05]. Such trails could allow an independent reviewer to audit the investigation.

We will now introduce a lifecycle model for data-oriented investigations and identify the categories of provenance in this lifecycle.

## 2.3 Data-Oriented Investigation Lifecycle

The prime commodity in Data-Oriented science is data. The process of obtaining scientific data and its exploitation towards scientific findings is captured in lifecycle models. Pepe and Borgman have identified a model for environmental science [PMBVdS10], Green and Guttman for Social Sciences [GG07], and Matthews large scale facility science [MBJC14]. In Figure 2.1 we provide a simplified version of Pepe and Borgman's model, as it is inclusive of other models. The lifecycle is comprised of Data Collection, Data Analysis and Publishing activities. Note that in reality activities rarely occur in the order implied in Figure 2.1. Investigation are often undertaken in two modes, the *exploratory* (or *tinkering*) mode [MRHBS06] [GGW+09], followed by a *routine* or *production* mode. All three activities in the lifecycle can be performed in these two modes, therefore modes are not depicted separately in Figure 2.1. During exploration scientists are in the process of shaping their study, they might iterate through cycles of data collection and analysis in tinkering mode. When the study design is finalised, it is then enacted in production mode to generate (publishable) results upon

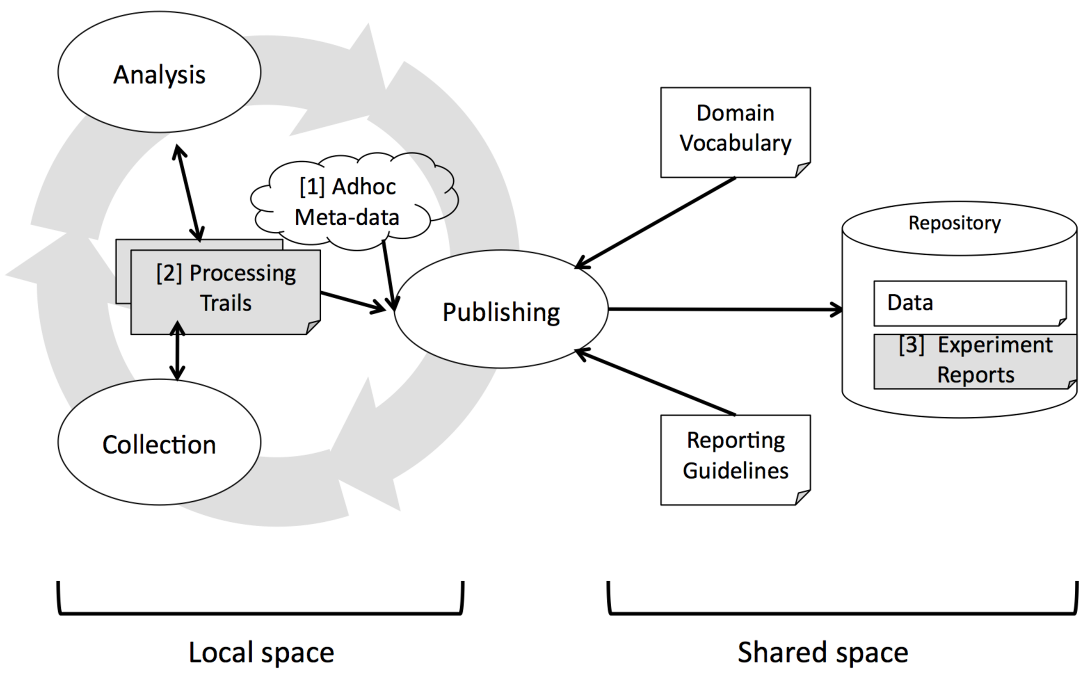


Figure 2.1: Simplified Lifecycle for Data-Oriented Research.

which scientific findings are predicated.

**Collection** refers to the gathering or capture of source datasets. First-hand data collection happens through wet-lab experiments, observations or field studies. It is also possible that scientists re-use data collected by others, in such cases data collection is second-hand, by accessing shared data from repositories [CDS$^+$07], [BWMP07], [GWCS09]. **Analysis** corresponds to a broad range of activities including data grooming such as cleaning, filtering, integrating and format normalisation [GAB$^+$14], or analytical processing, which may range from running simulations, mining data for patterns, or computing and visualising data statistics.

An innate practice in scientific investigations is *recording* metadata about the process as it happens. This commonly takes two forms:

- Records manually created by the scientists in electronic/paper lab notebooks [PG07]. These are loosely structured documentation (depicted as Category (1) in Figure 2.1), which describe research hypotheses, design of the study, findings, and conclusions. Such records are intended for the scientist's own use when refining study designs through iterations, or to communicate their findings with other resident scientists. Such records are also an important input to the publishing process.

- Records automatically created by trailing computational instruments scientists use during data collection and analysis (depicted as Category (2) in Figure 2.1) [SPG05]. Workflows alongside tools, scripts and databases are the most prominent types of such instruments. Trail-type provenance records typically include logs of computational processing including timestamps, software versions, data content hashes and environmental configurations.

The lifecycle comes to fruition with the **publishing** phase. Traditionally, publishing amounts to narrating the study's scientific contributions in the scholarly article, and explaining how it was undertaken in the article's *materials and methods* section. Recall from Chapter 1 that in data-oriented science the output of investigations include not only the scientific findings but also the datasets supporting those findings. As a consequence publishing has become a more elaborate process involving, in addition to narrative descriptions, the creation of metadata providing experimental context to shared datasets [SLN$^{+}$09][fSDSC02]. In this thesis we refer to these (semi)structured metadata artefacts as "Experiment Reports" (depicted as Category (3) metadata in Figure 2.1). In the early days of Data-Oriented science, curators of large-scale institutional repositories had sole responsibility for evaluating submissions and creating experimental metadata. Nowadays data submitting scientists are also expected to perform curatorial duties and reporting is to happen (locally) as part of the investigation lifecycle. In order to assist scientists in describing their experiments, community-initiatives in several domains provide tools [WOH$^{+}$12] [SKAC14], models [SRSF$^{+}$12] [FAJS05] and reporting guidelines [BBB$^{+}$15] [TFS$^{+}$08]. Currently this happens through a manual, publishing-time curation process, where earlier mentioned categories of metadata are used as inputs. Scientists sift through ad hoc metadata (lab notes, file names, file contents), or inspect experiment execution trails to select relevant data subsets and use curation vocabularies and tools to create experimental metadata.

### 2.3.1 Role of Scientific Workflows

Within the lifecycle scientists are faced with complexities in two dimensions. The first is the diversity of scientific resources available for use. Community databases or analytical tools exposed via web services, compute clusters or grids are examples of resources. Use of locally developed scripts, analysis programs, simulations further adds to resource diversity. During the exploratory mode of an investigation scientists find resources [BTN$^{+}$10] [gbi15], explore their interfaces to determine how they should be accessed. Once individual resource selection and configuration is determined, the next challenge lies in bringing resources together into an overall study design that can be used in production mode. In production mode the systematic compositions of resources is a necessity for various reasons. Scientific Methodologies [Mon06] identify replication, comparison and factorial design as desired characteristics for an experiment. An analysis is rarely enacted once, it needs to be repeated to check whether replication is achieved, it needs to be parameterised and re-enacted so that the impact of changing multiple factors can be observed and compared with each other.

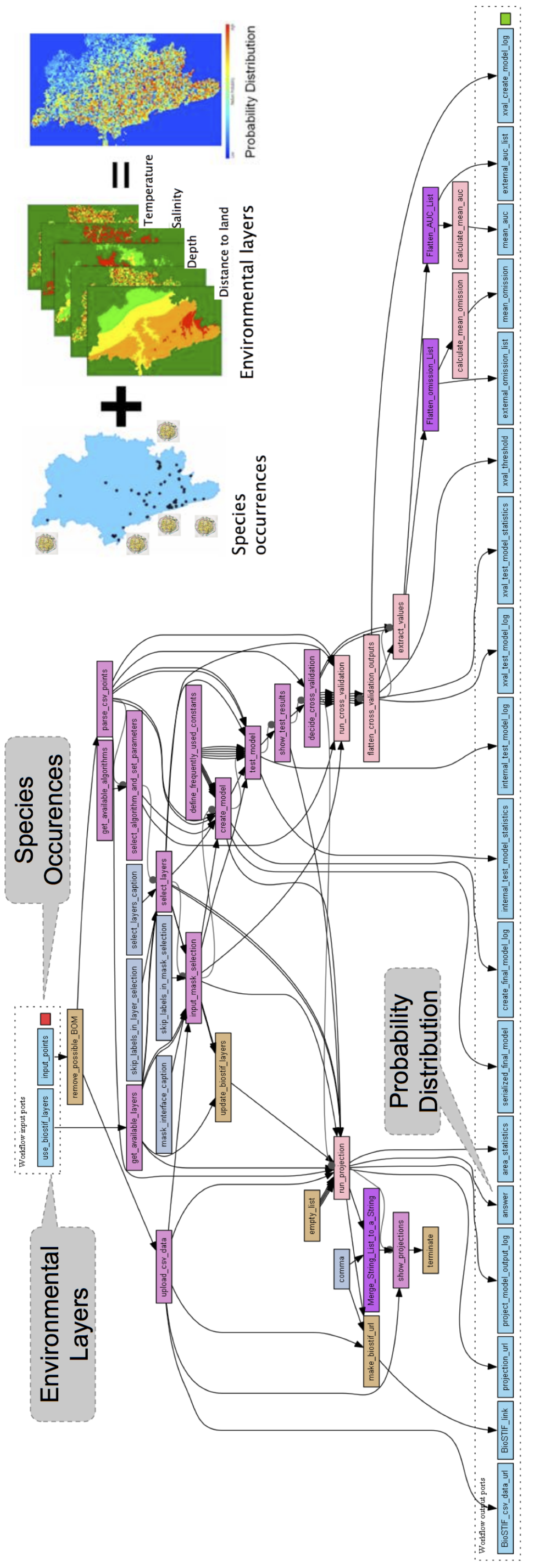


Figure 2.2: Ecological Niche Modeling Workflow.

In recent years Scientific Workflow Systems [DF08] have been adopted in many disciplines to weave data access and analysis functions into workflows. DiscoveryNET [CGG+02], and Triana [Tay06a] are earlier, Wings [GRK+11], Pegasus [DShS+05], Kepler [LAB+06], Taverna [MSRO+10], Vistrails [CFS+06], Galaxy [GRH+05], Loni Pipeline [DLP+10], Nipype [GBM+11], KNIME [BCD+07], and Pipeline Pilot [pip15] are current examples of workflow systems. An example workflow from environmental sciences implemented with the Taverna system is given in Figure 2.2. The workflow [Gio13] is part of a study on the computational modelling and projection of the habitation of invasive species in the Baltic Sea [BOHS07]. The study involves a data analysis protocol named Ecological Niche Modelling of species, hereon referred to as ENM. The main principle in ENM is to feed geo-referenced species occurrence and environment data into a prediction model to obtain a probable species distribution. The ENM workflow addresses the above outlined challenges as follows:

- Workflows bring rigour to a data oriented investigation by capturing an explicit and runnable codification of the analysis process as a set of analytical steps connected by dataflow links. Workflow designs not only capture analysis activities but also capture the experimental parameters (or factors) that can be set up per execution. The ENM workflow is run several times for different input species and layers to compare and discuss the effectiveness of the prediction model. Note that workflows often represent the computationally coordinated parts of the analytical process, which might also involve off-line processes (e.g. field/lab-work ). The benefit of rigour and the ability to re-run analyses with differing parameters are factors that motivate scientists to adopt workflows as instruments of data analysis.

- Access mechanisms to resources are highly heterogeneous. The ENM example combines calls to the OpenModeller projection service [dSMGdS+11], with runs of local scripts and the invocations of an interactive data visualisation and cleaning application called BioSTIF [Kul15]. Workflow systems act as a super-client by providing a ready-to use resource access layer for standard interfaces like command-line invocations, script execution or web service calls, furthermore, they allow extending this layer by plugging in custom interfaces to reach, for instance, applications, such as BioSTIF, or scalable computing platforms such as compute clusters or grids [TMN+10] [DShS+05].

- Workflow systems provide the basic trailing of workflow executions. Recall from earlier we mentioned the importance of such trails for the audit and verification of analyses. Trailing can be done on the resource side or on the client (workflow) side. Resources in scientific domains either support no trailing of their execution or they provide heterogeneous custom trails, which in itself is a challenge to bring together. In the ENM workflow one of the resources is the Open Modeller web service and projection modelling is performed through a series of calls to this service. Meanwhile, this service does not document or correlate the multiple requests made by a client per modelling effort. Another resource is the BioSTIF application. The spreadsheet libraries underpinning BioSTIF allows the tracing/recording or user actions per data cleaning session [OFC$^{+}$15] yet these records comprise an isolated, and tool-specific representation of a part of the analytical process. The trailing capabilities in workflows allow for basic documentation of the analytical process holistically using uniform representations and models.

### 2.3.2 Promise of Trail-Type Provenance

Workflows are one of many computational instruments that scientists use. Bose et al [BF05] have identified these instruments as: Workflows, (Web) Services, Database Queries, Scripts/Programmes, and Tools. For each instrument the approach to obtain provenance differs so does the potential use of this information.

**Workflows** have found acceptance as a methodology to design scientific analyses top-down with provenance in mind. The basic promise of workflow provenance is to have transparency into the experiment in a holistic way identifying it as a process comprised of sub-steps, where each step is documented with basic provenance represented in uniform models and representations (As outlined in Section 2.3.1). This way an experiment acquires transparency needed to support scientific audit and peer review. Workflows require data processing capabilities to be organised and made accessible as resources exploitable by workflow tooling. Workflow designs act as plans that specify how individual resources should be composed and how the overall composition can be configured or parameterised. When compared to provenance from other instruments, workflow provenance has the highest maturity as techniques here have made their way as features into scientist's tooling (reviewed in Section 2.3.3). On the other hand workflows incur a significant cost of adoption for scientists [Pla11]. The ENM workflow has been developed over a 3 year period with a 3 person team comprised of domain scientists and workflow system experts.

**Scripting platforms** such as R [R C13], MATLAB [MAT10] and Python [Ros95] are commonly used by scientists to access statistics, numeric analysis or visualisation libraries. Provenance support for scripts has been a recent area of research, where the intent is to collect information about scientific data processing in a way that is not disruptive to existing working practices of scientists, i.e. without the scientist needing to refactor their existing scripts into more structured forms like scientific workflows. Despite their popularity and widespread use, scripts are considered to have weaker abstraction power, or a lower-level mechanism to document an experimental process [DBK+15].

**Command-Line Tools** are another widely used category of instrument for scientific data processing. The EMBOSS [RLB00] in Genomics or STILTS [Tay06b] in Astronomy are two prominent conmand-line tool sets. Similar to scripts the intent in collecting provenance from tool executions is to be non-intrusive to working-practices while collecting provenance of analytical processing. During the exploratory stages of investigations scientists interact with tools to figure out suitable analytical steps or correct configurations. A distinct motivation here is to record those exploratory activities so as to be able to convert past actions into data analysis recipes (such as workflows) [BKC+01]. Tool provenance is gathered by system-level provenance collection techniques. Hi-Fi [PMMB12], SPADE [GT12], PASS [MRHBS06] and PLUS [CAB+10] are frameworks that embody such techniques.

**Databases** While prior identified categories are used to realise experimental processes, databases are used for a different purpose. In fields such as Biodiversity [gbi15], Astronomy [WOE+99] and Genomics [MGBRS+15] community data repositories are backed by relational databases. Using these, scientists manage and search over metadata regarding scientific resources. Unlike the other categories provenance in databases is focused on data and seeks to rigorously define the relationship between data in a database and the results of queries over that database. This well-defined relation is used to show how the existence and characteristics of source data justifies the existence and characteristics of query answers.

***Characterising our research:*** In a way that parallels the above identified instrument-oriented view of provenance, the research on provenance is typically categorised in three schools [CAB$^+$14], namely, Database, Systems, and Workflow schools [CCT09] [MRHBS06] [DF08]. The Database school originates from investigations on the materialised view maintenance problem [CCT09] where the objective is to bring precision and selectivity to view updates. The Systems school focuses on techniques for the capture of provenance from technological substrates that host computational processes. The Operating System shell [TF08], File systems [MRHBS06] or Program Instrumentation frameworks [SGB14] are examples of such substrates. Outputs of research from the systems school has been applied for provenance collection from Scripts and Command-Line Tools. The Workflow school corresponds to a large and recent area of investigation focusing on the documentation of any process that involves the generation of data. Standard, interoperable models of provenance [MCF$^+$11] [BDG$^+$12], provenance access mechanisms [BCBDH08] and user interfaces [ABL10] have emerged from the workflow school of research. **Against this landscape our dissertation research can be characterised as an approach in the workflow school, that is inspired from solutions the Database and Systems schools.**

### 2.3.3 Uses of Workflow Provenance

In this section we outline the current uses of workflow provenance within data-oriented investigations and observe the gap in reporting. During data collection and analysis scientists interact intensively with data processing instruments (workflows, tool and scripts) to shape the analytical process and obtain results from its execution. This interaction is defined to have a lifecycle of its own. Goble et al [GWG$^+$07] and Mattoso et al [MWT$^+$10] have broken it down into stages of Design, Execution and Result Analysis. We illustrate each stage with sample questions that can be answered with provenance. Throughout the examples we use the term workflow, which may equally be replaced with script or tool. The narrative is summarised in Table 2.2. (We use circles to denote whether workflow provenance is exploited in any stage of lifecycle. Fully-filled circle denotes exploitation, half-filled circle denotes partial exploitation and empty circle denotes no exploitation.)

**Design:** One key requirement for scientists is to track the *evolution* of experimental designs; capturing of workflow versions and revisions are examples of evolution provenance. The requirements for it originate from domains, where workflow design is a particularly complex, lengthy, and exploratory process [CFS$^+$06]. During the exploratory stages of an investigation workflow designs may evolve rapidly due to refinement of analyses or due to changes in external dependencies or computation

| **Investigation Lifecycle** | **Sub-Stages** | **End Use** | **Degree Of Use** |
|---|---|---|---|
| COLLECTION & ANALYSIS | WORKFLOW DESIGN | Evolution | ● |
| | WORKFLOW EXECUTION | Monitoring<br>Validation-Debugging<br>Smart Re-runs<br>Verification | ● |
| | RESULT ANALYSIS | Visualization<br>Comparison | ◓ |
| PUBLISHING | REPORTING DATA | Interpretation & (Re)use | ○ |
| | REPORTING METHOD | Interpretation & (Re)use | ◓ |
| | BUNDLING EXPERIMENT | Audit, Peer Review, Preservation | ● |

Table 2.2: End-Use Analysis for Trail-Type Provenance

infrastructures. In Figure 2.3 an example is given from the Vistrails system. On the left a version tree for a medical image processing pipeline is given. On the right the version named "aliases" selected from the tree is displayed. Examples inquiries over evolution provenance are as follows:

1. Which are the workflows that have been derived from this workflow?

2. At which version of the workflow has this step been added?

3. Which other workflows include calls to this (extinct) service endpoint?

**Execution:** Achieving seamless execution of computational analyses is by far the most prominent motivation for having trail-type provenance. Documenting the execution of an analytical step and the data consumed and produced by it is useful for: *monitoring* executions, *verification* of executions against designs [MDB$^+$13], step-wise *validation* and *debugging* [DShS$^+$05] [dOCVSaM14] of analyses, making runs more efficient by re-using prior results [ABJF06], or remembering optimal configurations of steps [HGB$^+$13]. To illustrate in Figure 2.4 displays Taverna systems "workflow execution panel", where statistics on the health of a particular workflow's run is displayed. Sample inquiries at execution stage can be:

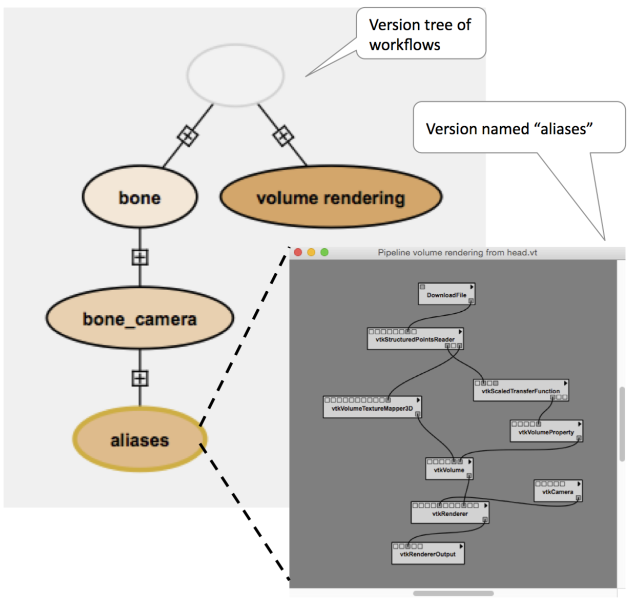
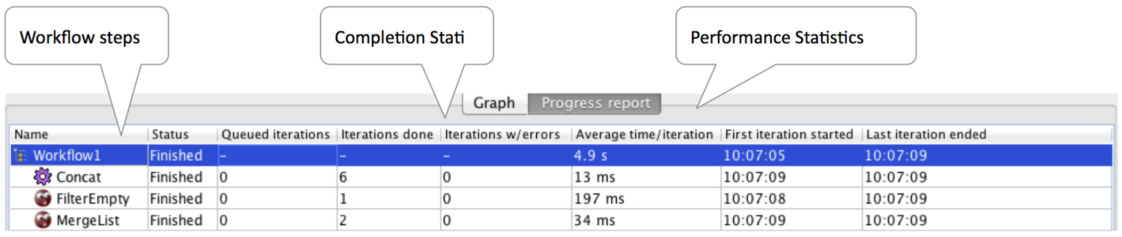


Figure 2.3: Workflow Evolution Tracking in Vistrails

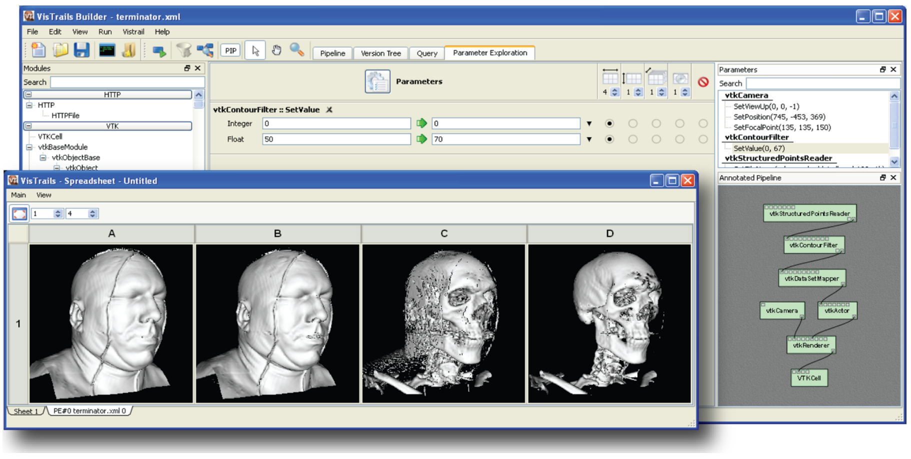


| Name | Status | Queued iterations | Iterations done | Iterations w/errors | Average time/iteration | First iteration started | Last iteration ended |
|---|---|---|---|---|---|---|---|
| Workflow1 | Finished | - | - | - | 4.9 s | 10:07:05 | 10:07:09 |
| Concat | Finished | 0 | 6 | 0 | 13 ms | 10:07:09 | 10:07:09 |
| FilterEmpty | Finished | 0 | 1 | 0 | 197 ms | 10:07:08 | 10:07:09 |
| MergeList | Finished | 0 | 2 | 0 | 34 ms | 10:07:09 | 10:07:09 |

Figure 2.4: Workflow Execution Statistics Collected by Taverna

1. Has this workflow generated the expected number of output files in the last run ?
2. What are the workflow runs with failed invocations of this service ?
3. What are the average execution times for all runs of this workflow ?
4. Re-run this branch of the workflow by re-using upstream results from this morning's run.
5. Obtain all downstream data artefacts in my workspace generated using this erroneous input data.

Both Design and Execution stage inquiries have significant level of support in existing workflow tooling, hence depicted with full-circles in Table2.2.



vtkContourFilter :: SetValue
Integer 0 → 0
Float 50 → 70

vtkCamera
SetViewUp(0, 0, -1)
SetPosition(745, -453, 369)
SetFocalPoint(135, 135, 150)
vtkContourFilter
SetValue(0, 67)
vtkStructuredPointsReader

Figure 2.5: Comparing Results of Different Runs in Vistrails

**Analysis:** Science is exploratory, and so are scientific data analyses. Scientists will run analyses several times with differing parameters. At the result analysis stage scientists may use provenance to *select* data subsets of interest among the larger pool of results [dPHG$^+$13]. Selected results are then inspected, compared or analysed to gain scientific insights, validate hypotheses or to (re)define study scopes [GWG$^+$07]. Scientists may ask for results that fall into a particular context, a particular time frame, or those obtained with a particular parameter setting. Figure 2.5 displays the "parameter sweep" interface of the Vistrails system [CFS$^+$06], here the workflow is configured to run repeatedly using a collection of parameters and results per parameter are visualised side-by-side for comparison. Further illustrative inquiries that support this stage can be:

1. List all projection results that are obtained from the ENM workflow, where input environmental layers contains *water salinity*.

2. Has the ENM projection result obtained for species *Alexandrium minutum* stayed the same for the last three runs.

3. Launch the output of ENM workflow per *input species* using BioSTIF visualiser.

These inquiries follow an access pattern over provenance, which we identify as *Provenance Driven Data Selection* (described in detail in Section 2.4.6). Briefly, these inquiries use provenance as a scaffold to reach data, moreover they often transcend provenance information by placing selective criteria not only on provenance but also on data values. Due to a number of factors in collecting and representing trail-type provenance, these inquiries are not fully supported by scientific workflow tooling. We discuss factors in Section 2.7.

**Publishing**: As introduced in Chapter 1 data-oriented science has put emphasis on publishing as it now involves the creation of experimental metadata. Trail-type provenance is an information source that scientists can use to create experimental metadata. We analyse the publishing stage in terms of three categories of experimental metadata.

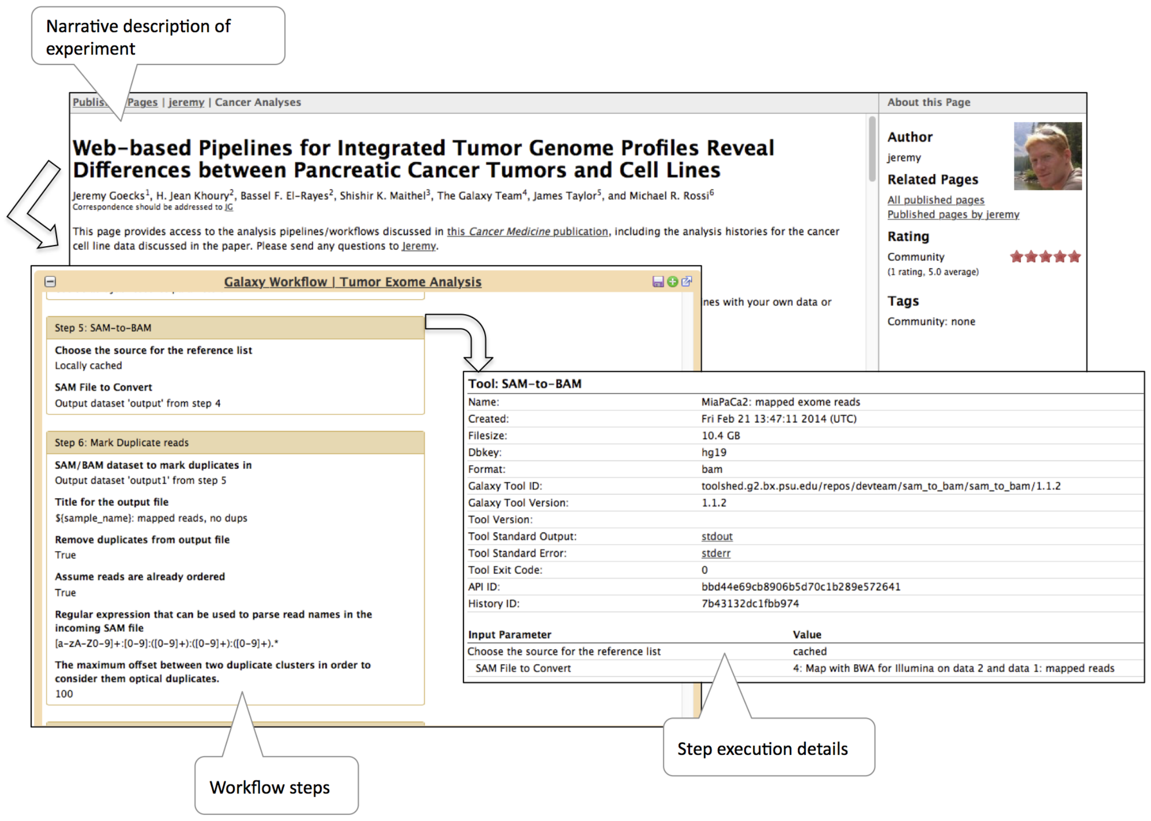
First category is **Experimental Bundles**. Bundling is a mechanism to package (where possible) all resources used within an experiment to enable its preservation for future re-enactment or audit by external parties. Bundles include data processing instruments (workflows, scripts, tools); provenance collected from instrument execution; datasets; narrative documentation describing hypotheses and findings; and finally metadata annotations over packaged artefacts. Support for bundles exists in state of the art scientific tooling. ReproZip [CSF13] is a system used to package command-line tool based scientific analyses. The Research Objects toolkit [BCG$^+$12] and the History Publishing feature of Galaxy [GNT10] system allow packaging of experimental artefacts, generated in workflow-based analyses. We provide the screen shot of a Galaxy published history in Figure 2.6. Here the analytical workflow and its step-wise execution logs and result data are presented through a set of interlinked web pages, which scientists use as online supplementary material to their manuscripts.

The second and third categories are the earlier mentioned **Experiment Reports**, which are curated metadata shared alongside datasets. The primary purpose of this metadata is to inform the scientist on the experimental context and help in interpretation of shared datasets. **It is for the creation of experiment reports, where we observe the impacted (or limited) use of trail-type provenance, which we call the provenance gap**. Detailed illustration of experiment reports will be given in Section 2.5 and a detailed formulation of the reasons underlying the provenance gap will be given in 2.7. Due to the provenance gap there is no push-button solution in scientific tooling for the creation of experiment reports, which typically take two forms:

- **Reports on Method:** The analytical method followed for data collection and analysis is an important piece of metadata that provides context. Guidelines on reporting [nat15] state that whenever scientists use a novel or previously unseen protocol they are expected to document it. Intuitively this corresponds to outlining analytical sub-steps, their dependencies and possible configurations. Workflow-based investigations are advantageous as they provide a codification of the analytical method. In recent years there has been a proliferation on publishing of workflows both as tools of data processing and as documentation of method followed in scientific studies [RG10] [GG11]. Workflows are published in workflow repositories. Examples are CrowdLabs [MSFS11], the public Galaxy portal [gal13] and myExperiment [DRGS08], which is the largest public repository of workflows. Despite the availability of infrastructure for both building and sharing workflows, using workflows for reporting is not instantaneous (hence depicted as partially supported in Table 2.2). The primary reason is workflows operate at a lower (computational data-processing) level, hence they are abundantly detailed to use in reporting. Scientists adopt two manual solutions for obtaining simpler reporting-friendly descriptions. One is designing workflows modularly and in a layered fashion using the workflow tooling and then employ the higher (abstracted) workflow layers for reporting. In certain cases however layered designs might still be too complex to convey the methodology, such as the case in the ENM workflow. In these circumstances scientists create further simplified depictions of the analytical method through experiment sketches (exemplified earlier in Chapter 1).

- **Reports on Data:** The experimental parameters is crucial contextual metadata required for interpretation of result datasets. This is of particular importance in cases where parameters are varied within the scope of an analysis; such as the parameter sweep interface of Vistrails in Figure 2.5. Intuitively a data report is a record containing descriptions of selected analysis results presented in association with descriptions of parameters and parameter values used for obtaining those results. Here analyses based on workflows are clearly advantaged over adhoc approaches as workflow execution provenance records and links up parameters and results. That said, to our knowledge there is no support in existing workflow-based data analysis tooling for the automated generation of data reports. The reasons that underlie this disconnect are several and has common roots with *Provenance Driven Data Selection* exemplified earlier. In current practice scientists manually inspect execution trails (through user interfaces exemplified earlier), they select report-worthy results and configuration values and transfer them to reports. To enable unambiguous interpretation scientists typically associate metadata with each transferred value.

To this end we introduced the investigation lifecycle and the roles that workflows


Narrative description of experiment
Workflow steps
Step execution details
Publis… Pages | jeremy | Cancer Analyses
About this Page

Web-based Pipelines for Integrated Tumor Genome Profiles Reveal Differences between Pancreatic Cancer Tumors and Cell Lines

Jeremy Goecks[1], H. Jean Khoury[2], Bassel F. El-Rayes[2], Shishir K. Maithel[3], The Galaxy Team[4], James Taylor[5], and Michael R. Rossi[6]
Correspondence should be addressed to JG

This page provides access to the analysis pipelines/workflows discussed in this *Cancer Medicine* publication, including the analysis histories for the cancer cell line data discussed in the paper. Please send any questions to Jeremy.

…nes with your own data or

Author
jeremy

Related Pages
All published pages
Published pages by jeremy

Rating
Community ★★★★★
(1 rating, 5.0 average)

Tags
Community: none

Galaxy Workflow | Tumor Exome Analysis

Step 5: SAM-to-BAM

**Choose the source for the reference list**
Locally cached

**SAM File to Convert**
Output dataset 'output' from step 4

Step 6: Mark Duplicate reads

**SAM/BAM dataset to mark duplicates in**
Output dataset 'output1' from step 5

**Title for the output file**
${sample_name}: mapped reads, no dups

**Remove duplicates from output file**
True

**Assume reads are already ordered**
True

**Regular expression that can be used to parse read names in the incoming SAM file**
[a-zA-Z0-9]+:[0-9]:([0-9]+):([0-9]+):([0-9]+).*

**The maximum offset between two duplicate clusters in order to consider them optical duplicates.**
100

**Tool: SAM-to-BAM**

| | |
|---|---|
| Name: | MiaPaCa2: mapped exome reads |
| Created: | Fri Feb 21 13:47:11 2014 (UTC) |
| Filesize: | 10.4 GB |
| Dbkey: | hg19 |
| Format: | bam |
| Galaxy Tool ID: | toolshed.g2.bx.psu.edu/repos/devteam/sam_to_bam/sam_to_bam/1.1.2 |
| Galaxy Tool Version: | 1.1.2 |
| Tool Version: | |
| Tool Standard Output: | stdout |
| Tool Standard Error: | stderr |
| Tool Exit Code: | 0 |
| API ID: | bbd44e69cb8906b5d70c1b289e572641 |
| History ID: | 7b43132dc1fbb974 |

| Input Parameter | Value |
|---|---|
| Choose the source for the reference list | cached |
| SAM File to Convert | 4: Map with BWA for Illumina on data 2 and data 1: mapped reads |

Figure 2.6: A Galaxy History Page

and workflow provenance play in it. We also surveyed the current end-uses of workflow provenance, and noted its use is limited in reporting. We will now take a closer look at this disconnect. We first provide the basic intuitions for provenance (Section 2.4), and later use these intuitions to observe the characteristics of experiment reports as provenance records (Section 2.5). We then use the same intuitions to understand workflow provenance (Section 2.6).

## 2.4 Provenance Basics

We will now identify fundamental characteristics of provenance information and provenance collection mechanisms. While doing this we establish terminology used throughout the dissertation. Some of our characterisations are drawn from an early, but landmark taxonomy by Simmhan [SPG05], and a follow-on extension of it by Serra da Cruz [dCCM09]. Freire [FKSS08] and Carata [CAB$^+$14] provide more compact and recent surveys, which analyse representative provenance systems with the goal of understanding major clusters of provenance-enabled computational systems. In this section we provide basic background on the following:

- the information captured in provenance
- the granularity of provenance
- the approaches in collecting provenance that affect its accuracy and informativeness, namely black and white-box provenance
- the perspectives of describing a process in provenance, namely prospective and retrospective perspectives
- models/vocabularies for provenance
- the major means of provenance representation and access

### 2.4.1 Subject of Provenance

Provenance is the documentation of a process starting from origins (sources) through to results obtained within that process [SPG05]. Consequently provenance has two components; *process* and *data*. A basic provenance record outlining data, processes and their relations is given in Figure 2.7. The trace here states that a *Filter* process

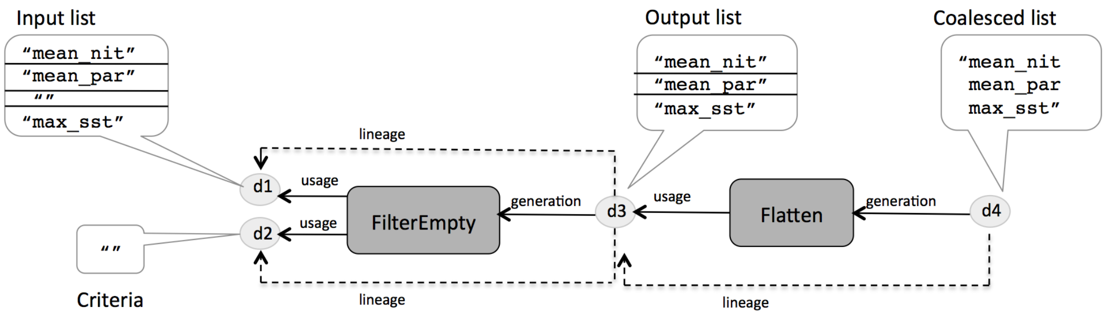


Figure 2.7: Basic Provenance of a Filtering Process

has occurred, which has used two data artefacts, $d1$, a list of layer names used in Biodiversity [1], and $d2$, a string value (in this case empty string). Process has generated one output $d3$, which is another list of layer names comprised of items that do not match the criteria. Provenance records qualify the role that data plays with respect to processes. This way we know that $d1$ acts as the *InputList* to the process, $d2$ is the *Criteria*. A follow-on process called $Flatten$ has used the output of filtering and has converted this list of layer names into a single coalesced string of layer names. A key relation that is implied from such a basic record is data lineage [SPG05] (depicted with dashed lines in Figure 2.7):

> *Data Lineage* corresponds to the ancestry relations among data artefacts that are linked via a process execution. The inputs used by a process make up the data ancestry for the process's outputs.

In this dissertation we further categorise lineage in two: as *Opaque* and *Transparent*. Opaque lineage informs us that there is an ancestry relation among two data artefacts but does not provide any further information on what kind of relation it is. The trace in Figure 2.7 contains opaque lineage. Transparent lineage on the other hand provides additional information about the nature of ancestry. For instance, for the case in Figure 2.7, lineage stating that the output list in is $copied\ from$ the input list, or, for a web page editing scenario, lineage stating that an edited page is a $revision\ of$ the older page are examples of transparent lineage.

The data artefacts generated by one process may later be used by another, as depicted in Figure 2.7 where $Flatten$ has used the data generated by $Filter$. This results in linking of processes, which we refer to as process causality [MGM+08a].

[1]Taken from the ENM example.

*Process Causality* corresponds to a causation relation between two processes that are linked via an information flow (or communication) among them.

Provenance information is conceptually a Directed Acyclic Graph (DAG) comprised of processes and data as nodes, and the usage, generation, and the induced lineage and causality relations as links among nodes. Note that the trace in Figure 2.7 conveys the most **basic information** about a (computational) process. Meanwhile there is no explicit information indicating that input/output data are lists of *Environmental Layer Names* or the activity is a kind of *String Filtering*. Bose and Frew identify this additional information as *Fundamental Metadata* [BF05], which is of importance when making use of provenance information. In this dissertation we adopt the term *Domain-Specific Metadata*.

*Domain Specific Metadata* corresponds to additional information that further characterises data and processes documented in a provenance trace using domain-specific terms.

Such Information can be found in implicit and explicit, hence machine processable, forms. An example of implicit information is, for instance, the values of data artefacts, e.g. layer name values, consumed and produced by the filtering activity. Domain specific information can explicitly be specified in different forms such as key-value pairs [PNNJ05], textual markup attached with generic Annotation Ontologies [COGC$^+$11], or domain-specific semantic characterisations [MSZ$^+$10] using ontologies from respective scientific domains.

## 2.4.2 Granularity of Provenance

Provenance can be collected at varying granularities for both data and processes. Particularly the granularity with which data is represented and its lineage probed is a critical factor in determining the informativeness of provenance. Scientific datasets are often of a granular nature. Data can be found in Collections (e.g. the input collection of different environmental layers to the ENM workflow ). Furthermore individual items in such data collections may also have granular structures (e.g. Raster-Vector based representation of environmental layer information).

With regards to granularity the provenance record filtering (in Figure 2.7) is coarse. The input and output of the process are modelled with singular items in provenance even though they are collections of strings. Similarly the entire filtering activity is recorded as a single process. An alternative recording of the same process where data is captured at a fine-grain is given in Figure 2.9. Here the collection structure of layer

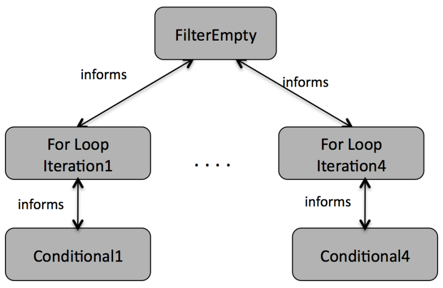


Figure 2.8: Granular Process Provenance For List Filtering

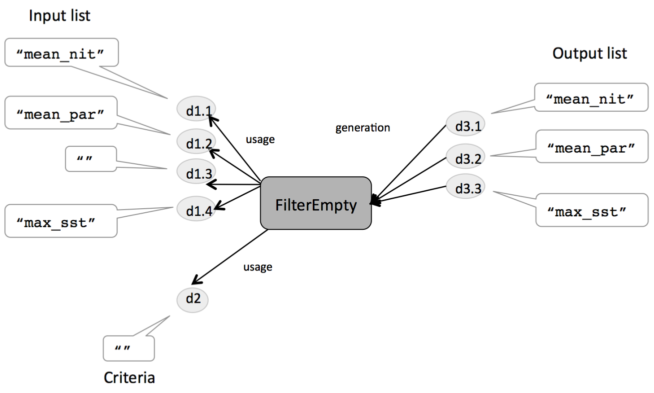


Figure 2.9: The $n - by - m$ situation

data is reflected to provenance and where each layer in the collection is modelled with a distinct data node in the provenance graph.

Processes can also be recorded at finer granularities depending on the layer in the software stack that provenance collection occurs. The provenance of Figure 2.7 is a coarse documentation created at the level of the workflow that uses the filtering data adaptation tool (from the local library of the Taverna workflow system). The underlying implementation for the tool is a Beanshell script [bea15] that iterates over the input list and conditionally transfers items to the output list. In case provenance were to be recorded at the (lower) script level, then execution of each control-flow block in the script, i.e. the iterations(loops) and the conditionals within), could be documented as (sub)processes (as illustrated in Figure 2.8).

When data is modelled in fine-grain, and yet the processes are documented coarsely, we may get to the situation known as the *n-by-m problem* [CAB+14]. Based on our study of provenance literature, in this dissertation we further categorise this problem into two:

- *Role-wise n-by-m* [GCP13] [DBK+14]: This is the problem of determining whether inputs of all different roles have actually contributed to outputs of all roles. In both examples in Figures 2.7 and 2.9 lineage would be induced from data items in role $OutputList$ to $Criteria$ as well as $OutputList$ to $InputList$. For this case those inferences would not result in inaccurate lineage, i.e. both the input list and criteria are actually used to produce the output. Imagine, however a variant of the filtering activity, which produces an output filtered list, but also produces a copy of the original list (say for debug purposes). In such a case the copy of original list would not have a dependency to the filtering criteria.

- *Data-wise n-by-m* [MPB10]: This corresponds to determining whether individual data items in a particular input role do actually contribute to individual items in an output role. For the example in Figure 2.9 lineage would be induced between all items in input layer list ($d.1.1$, $d1.2$,..) and all items in output list ($d3.1$, $d3.2$, ...) ( $n \times m$ lineage relations omitted for simplicity). As we know that the process is a list filtering, clearly not all of those lineage relations are accurate. When activities consume and produce data collections, in the worst case the n-by-m pattern causes lineage to be inaccurate in the order of size of input data collection(s).

Data and process granularity is highly dependent on the purpose of provenance collection. From the perspective of an experiment auditor/reviewer, a coarse grained depiction of input-output lists may suffice as evidence of successful execution for a scientific workflow. However from the perspective of a developer debugging Taverna library tools, or for a user who is interested in downstream products of a particular environmental layer (as in ENM) a more fine-grained provenance may be required. Besides these benefits fine-grained capture of provenance does have the side effect of making provenance more complex, and more costly to collect [dCCM09].

## 2.4.3 Black and White Box Provenance

A characterisation commonly made in literature for provenance collection is *black-box* versus *white-box* provenance [CJ10] [ADD+11]. These two categories differ in their assumptions on the (un)availability of actionable information regarding the inner-workings of the computational process, for which provenance is collected.

The *black-box* approach makes no assumptions on the inner execution details of processes, where provenance is collected by observation, documenting the inputs and outputs of a process. Earlier traces given in Figures 2.7 and 2.9 exemplify this approach. The advantage of this approach is that it is a flexible mechanism for the tracking of arbitrary (or unrestricted) computations. The disadvantage is that the lineage in black-box provenance is opaque lineage. Furthermore, for the cases of n-by-m data artefacts connected by a black-box process, the accuracy of lineage relations are not guaranteed.

In the *white-box* approach it is assumed that information on the inner-workings of the computational process is available to the provenance collection infrastructure. Consider that we know that the running example, the filtering process, is realised by a script that iterates over the input list, transferring the terms different than the criteria to the output list. Availability of such information means we can build an "inversion procedure" [SPG05], that we can use to trace every possible execution of the filtering process to record white-box provenance. Such provenance could go beyond black-box provenance in the following respects:

- **accurate lineage** that links items in the output layer list only to the equal data-valued items in the input layer list (rather than $n \times m$ number of relations)
- **transparent lineage** that denotes that each item in the output list has been $copied\ from$ corresponding items from the input list.

**According to the characteristics identified herein, provenance in workflows, command-line tools, and scripts fall into the black-box category, whereas database provenance is white-box.** Relational queries are built on operators that have well-defined set (or bag) theoretic semantics, such as Selection, Projection, Join or Union (SPJU) [AHV95]. Thanks to this exposition, even-though queries get executed over entire tables (with millions of records) it is possible to trace query results back to individual source records [CCT09]. Database provenance is studied in three categories. The core capability, i.e. the creation of fine-grained and accurate lineage between query result records and source records is called **Why Provenance**. On top of this comes **Where** and **How** provenance which add further resolution and qualification to lineage. **Where provenance** pushes lineage granularity to the level of individual cells in records. As relational query operators build result records by copying values of source records, where provenance qualifies cell-level lineage as value copying. **How Provenance** corresponds to specifying the nature of the process, individual SPJU operations that has led to the creation of result records.

Another important difference between black-box and white box provenance is in their end-uses. In black-box, provenance is treated as a structure, a DAG, that can be traversed, queried or compared with other DAGs, yet not interpreted. When lineage is transparent, i.e. it is informative on the nature of relation between source and result data, as in **where** and **how provenance**, it becomes possible to interpret provenance. This way one could build value-added capabilities over provenance traces. As reviewed in detail in Appendix A, database provenance has diverse end uses. **Where provenance** has been used to propagate attributes of source records to their copies in results [BCTV04] [WM90]. The extra information in **How provenance**, in the form of query operators has been used to create more sophisticated attribute propagation capabilities. Examples are computing confidentiality, quality [KIT10] or uncertainty scores [ABS$^+$06] for query results. An apparent drawback of database provenance is its restrictive assumption on the kinds of data processing steps that are supported. Clearly not all data is in relational form, and not all scientific data processing can be represented with relational queries.

There are also a number of approaches, characterisable as grey-box provenance. These approaches use "weak inversion" [WS97] [SBD$^+$09] procedures to compute lineage, accuracy of which is less than 100% percent correct white-box provenance, but better than $n-by-m$ black-box lineage.

### 2.4.4 Prospective and Retrospective Provenance

A common categorisation of provenance, originally observed by Clifford et al [CFV$^+$08] is *retrospective* versus *prospective*. Provenance is, by its nature, historical or backward looking. The example trace for list filtering given earlier in Figure 2.7 is a retrospective (or historical) provenance record. Such records can sometimes be accompanied with a prospective counter-part, called a "plan"[BDG$^+$12], a "recipe" [CFV$^+$08] or "method" [RG10]. Recipes outline the intended process; process's sub-steps, their dependencies and possible configurations. A query, a script or programme, a workflow description, or a lab protocol are examples of recipes. Returning to our example the beanshell script used for filtering corresponds to prospective provenance.

When compared to other application areas prospective provenance has had particularly strong adoption in data-oriented science due to the following benefits:

- the credibility of scientific results rely heavily on the transparency of working methods, prospective provenance is used to capture the method adopted in data processing. [RG10].

- when prospective provenance captures the method of data processing with a suitable abstraction (e.g. workflows) this abstraction can be used as a guide/roadmap to browse or query retrospective provenance traces [BCBDH08] [ABL10].
- prospective provenance can be used to verify retrospective traces and to check whether data processing has occurred as intended [MDB+13] [MG11].

Provenance is of increased utility when it is provided in an agreed upon and machine processable form, we review efforts in this direction in the next section.

### 2.4.5 Provenance Models

Providing models for provenance has been an active topic of study in recent years [GCG+10]. The purpose of having provenance models is:

- to have a commonly agreed upon vocabulary for stating provenance,
- to have structured machine processable representations,
- to have formal foundation for the inference of refined provenance (such as data lineage or process causality) from basic/crude observations,
- the ability of validating or checking provenance traces for consistency.

Models can be broadly viewed in two categories: (1) generic ones that attempt to capture the very essence of provenance and target widespread cross-domain applicability and (2) specialised ones that cater for the requirements of a use case or domain. Early EU Pasoa [MGM+08b] and EU Provenance [VSK+08] projects followed by a series of "Provenance Challenge" events have given way to the development of the pioneer generic model of provenance, namely the Open Provenance Model (OPM) [MCF+11]. Alongside OPM other models have emerged from specialised domains. Examples are the Provenir Ontology [SNB+11] for representing e-Science experiments, The Proof Markup Language [dSMRD08] for representing answer explanations in intelligent systems, the PREMIS model [pre11] for representing preservation metadata or the VOID vocabulary [ACHZ09] to reveal relations among interlinked datasets on the web. All these models, which are shown to have partial mappings with each other [GCG+10], have been input to the development of the next generation generic model of provenance, namely PROV. Similar to OPM, PROV is a technology independent model and is particularly targeted for interoperable provenance exchange. While

PROV and OPM's majority applications have been in the documentation of computational processes, these specifications can also be used for documenting physical processes such as lab protocols, or field work. Note that all models mentioned above are targeted to specify retrospective provenance.

In parallel to retrospective models, vocabularies have emerged for prospective provenance most specifically for the case of workflow provenance. OPMW [GG11], P-Plan [GG12], D-PROV [MDB$^+$13] and Wfdesc [BCG$^+$12] are such models that describe workflows in a manner abstracted away from workflow-system-specific details. These abstract descriptions are used in conjunction with retrospective provenance (PROV or OPM compliant traces) collected from execution of workflows.

**For the purposes of this dissertation two models have significance: PROV for retrospective, and Wfdesc for prospective provenance.** We selected these as the substrate over which we built our provenance abstraction and annotation techniques (described in Chapters 4 and 7 respectively). PROV is the state of the art standard, and a W3C recommendation for modeling provenance [GCG$^+$10]. On the other hand the Wfdesc model [BZG$^+$15], an output of the EU Wf4Ever project, has originated from efforts in creating experimental bundles for workflow based studies. This model is now being used as a foundation for the development of a Common Workflow Language (CWL) for scientific data analysis [AT15]. PROV and Wfdesc are the models of choice for representing provenance in the Taverna workflow system, which is the primary source of the analysis and test cohort for our techniques in this dissertation. We postpone the discussion of the details of these models till Section 2.6 where we illustrate Taverna provenance.

### 2.4.6 Provenance Representation and Access

An area of development that is orthogonal to provenance modelling and collection is choosing a provenance representation model and query language [HBM$^+$08]. Various data models such as XML, RDF or the Relational Model have been used for this purpose. Due to the Directed Acyclic Graph (DAG) nature of provenance information, graph data models have found particular applicability. Graph databases [Neo12] or RDF stores [CLFF10] are commonly used due to their native support for representing the transitive relations in a provenance graph (e.g.lineage) and querying these relations. Such capabilities are often realised with complex/non-intuitive queries and demanding query execution times in non-graph representations [ABML09].

Access to provenance information often happens the following ways:

- **Querying:** Node(s) of interest may be located by graph queries outlining desired patterns in provenance. Gadelha et al [GMWF11] have identified that application-independent provenance queries involve the following basic patterns:
  - seeking **nodes based on their attributes**. e.g. Process nodes of a particular name, Data nodes that have a designated value.
  - seeking **nodes that are involved in relationship(s)** (consumption, generation). e.g. Data generated by a particular Process.
  - seeking **a sub-graph of interest**. These are advanced queries that may involve regular path queries [DCVK$^+$13].
- **Traversal:** When a node of interest is identified, a typical follow-on pattern is to then recursively traverse lineage and causal dependency relations in the graph to obtain all descendants/antecedents of a data or process node.

An access pattern that is of importance for the purposes of this dissertation is achieved by a combination of Querying and Traversal. In this dissertation we call the access pattern of this type *Provenance Driven Data Selection*.

## 2.5 Experiment Reports

The increased emphasis given to the publishing of datasets has given rise to the proliferation of reporting guidelines and models for representing metadata. At the time of writing the UK's Digital Curation Centre (DCC) [RBR$^+$05], which is a cross disciplinary hub for guidance on data publishing, lists 34 vocabularies and 50 tools. Whereas a similar resource dedicated to bio-sciences, namely the Biosharing portal [MGBRS$^+$15] lists 339 domain-vocabularies, 71 guidelines, and 179 models and formats for scientific data. Ecological Metadata Language in environmental science [FAJS05], DarwinCore in Biodiversity [WDG$^+$15], ISA-Tab in life-sciences [SRSF$^+$12], are some prominent examples in their respective areas. Davenhall [Dav11] argues that metadata associated with scientific data supports three primary functions:

1. Search and discovery of data in a repository.
2. Programmatic reading or extracting of data from its physical representation.
3. Understanding and interpretation of data by (re)users.

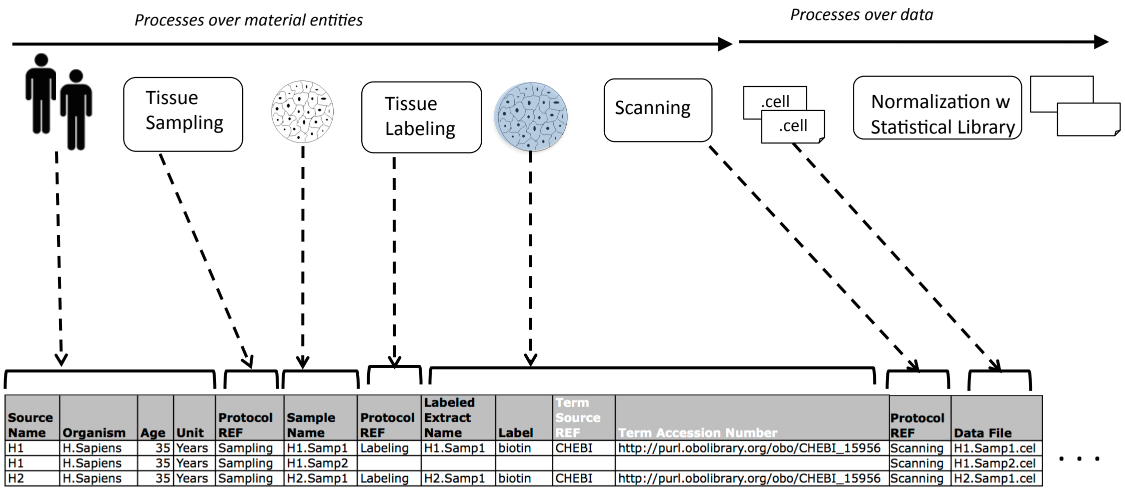
Some metadata standards cover multiple functions whereas others focus on a particular function. Our focus, i.e. metadata providing experimental context, is primarily targeted for the third category above. A thorough review of all metadata standards is out of scope for this dissertation. Instead in the following two subsections we illustrate experimental metadata with two real-world examples. The first one depicts standards-compliant reporting using the ISA-Tab specification. The second one presents how a study involving workflow-based analytical processes has been reported.

### 2.5.1 Reporting with ISA-Tab

Arguably the most mature approach in experiment reporting is the ISA-Tab specification. ISA-Tab is a widely adopted flexible standard used in Biological and Biomedical domains [SRSF$^+$12]. ISA-Tab is also the curation model for several life-science data repositories such as GigaDB [SZE$^+$14] and Metabolights [HSC$^+$13], and most recently adopted as such by Nature Publishing Group's *interdisciplinary* data journal "Scientific Data" [nat15]. We have therefore chosen this standard as an illustrator of experimental metadata used for the reporting of data.

ISA-Tab describes experiments through hierarchical Investigation (I), Study(S), and Assay (A) entities, where the Assay represents the smallest unit of experimentation. Figure 2.10 provides both the high-level process, and the corresponding ISA-Tab compliant tabular metadata for a DNA Microarray experiment (Assay) [GBMSRS14]. A distinctive aspect of ISA-Tab is its $end-to-end\ modeling$ capability covering processes over material entities, such as lab-work, as well as computational processes over data. Recall from Simmhan's definition that provenance had an *origin* and a *derivation history* component. ISA-Tab is significant as it highlights the importance of *origin*. Primary scientific data is often obtained from the physical world, in our example data comes from samples belonging to a human source of certain *age* and *gender*. These characteristics regarding data's ultimate source are also key differentiators for the data itself. These characteristics of the origin become a **provenance liability** on the primary data obtained such that every downstream data to be derived shall also be characterised with these attributes.

Another point of emphasis in ISA-Tab is the capturing of experimental variation and protocol configurations and the ability to discretely trace variation or configurations to resulting datasets. In Figure 2.10 some samples are labeled whereas others are not and using ISA-Tab metadata we can tell whether a result (scan file) originates from a labeled or unlabelled sample.



| Source Name | Organism | Age | Unit | Protocol REF | Sample Name | Protocol REF | Labeled Extract Name | Label | Term Source REF | Term Accession Number | Protocol REF | Data File |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H1 | H.Sapiens | 35 | Years | Sampling | H1.Samp1 | Labeling | H1.Samp1 | biotin | CHEBI | http://purl.obolibrary.org/obo/CHEBI_15956 | Scanning | H1.Samp1.cel |
| H1 | H.Sapiens | 35 | Years | Sampling | H1.Samp2 | | | | | | Scanning | H1.Samp2.cel |
| H2 | H.Sapiens | 35 | Years | Sampling | H2.Samp1 | Labeling | H2.Samp1 | biotin | CHEBI | http://purl.obolibrary.org/obo/CHEBI_15956 | Scanning | H2.Samp1.cel |

Figure 2.10: Reporting a Microarray analysis in ISA-Tab

The structural and semantic interpretation of the values in an ISA-Tab spreadsheet is enabled by extra (meta) information given in Study files. This extra information describes sources, samples, protocols, and their expected properties (e.g. inputs, outputs are attributes of a procedure). This information allows identifying the boundary between columns of information in an Assay spreadsheet to denote where description of one step finishes and the next begins. This way the standard allows each domain to define their own schema for a spreadsheet. In order to help in interpretation of attribute values ISA-Tab allows (optionally) qualifying values presented in the spreadsheet with a term in a *domain-specific controlled vocabulary*. This qualification is itself presented as an attribute, in the ISA spreadsheet. See the *TermSourceREF* column in Figure 2.10. The values in this column denote that each labeled extract is a material classified as *biotin* with respect to the definition for it given in *CHEBI* ontology [dMDE+10].

**ISA-Tab is a generic framework which does not prescribe what should be reported, but instead, helps scientists structure and annotate experimental information that is deemed report-worthy.** Experiment Reporting Guidelines we mentioned earlier advise scientists on what to report. For our example in Figure 2.10, this is determined with the Minimum Information About a Microarray Experiment (MIAME) Reporting Guideline [BHQ+01]. MIAME prescribes what columns of information need to exist in an ISA-Tab spreadsheet to report an experiment involving Microarray analysis.

As experiments reported with ISA-Tab could spread across lab work and computational analysis, the primary method for creating ISA-Tab compliant metadata is manual curation. However if scientists adopt strict record keeping, by recording each experimental activity and their inputs/outputs into a database, then such records can be exported as ISA-Tab spreadsheets [GPS+14]. In fact, the authors of the ISA-Tab specification state that if an experimental process that has occurred can be provided as a DAG of activity and entity nodes, then they provide a tool that can map the graph into an ISA-Tab spreadsheet, where each path in the experimental process is mapped into a row in the spreadsheet [BKSRS12]. Furthermore there is emerging work on integrating scientific databases, with workflow systems, where the former contains data collected from lab-work and the latter facilitates computational downstream analyses [GBLZ+15][GPS+14]. These efforts are all evidence of momentum building towards the exploitation of provenance to generate experiment reports.

### 2.5.2 Reporting Workflow-Based Experiments

Our second example is the earlier introduced ENM workflow, which is representative of an emerging set of data-oriented investigations, where the data analysis is comprised entirely of computational processes [BZG+15][PCB+14]. In the published article on ENM Obst et al [BOHS07] provide semi-structured information as experimental metadata to report both the method followed and the result data obtained. The method report for ENM is an informal flow diagram, an experiment sketch, given in Figure 2.11. This diagram presents a highly abstracted depiction of what is otherwise a complex and novel data analysis pipeline. The executable ENM workflow contains several sub-workflows, when expanded makes up 54 steps in total. Meanwhile, in the sketch (Figure 2.11), the protocol is depicted coarsely in terms of main analysis phases (*Create model*, *Test model*, *Project model*) and the high-level activities involved per phase. Note that the ENM workflow has 2 inputs and 19 outputs, whereas only a single input and output has been depicted in the sketch.

The data report of the ENM study is a metadata table given in Figure 2.12. The prime output of ENM is distribution maps for species. When each result map is introduced it is accompanied by row of metadata, comprised of contributing input factors, and outputs characteristics. This metadata organisation is very close in spirit to the ISA-Tab specification, although here there is no controlled structuring of data (multiple values are presented in a single cell) or there are no explicit semantic annotations (no taxa specified in table for species names). The first row in Figure 2.12 states that source occurrence data belonging to *Alexandrium minutum* of size of *3077* records has been subjected to the filtering protocol which has resulted in a dataset of *79* records,

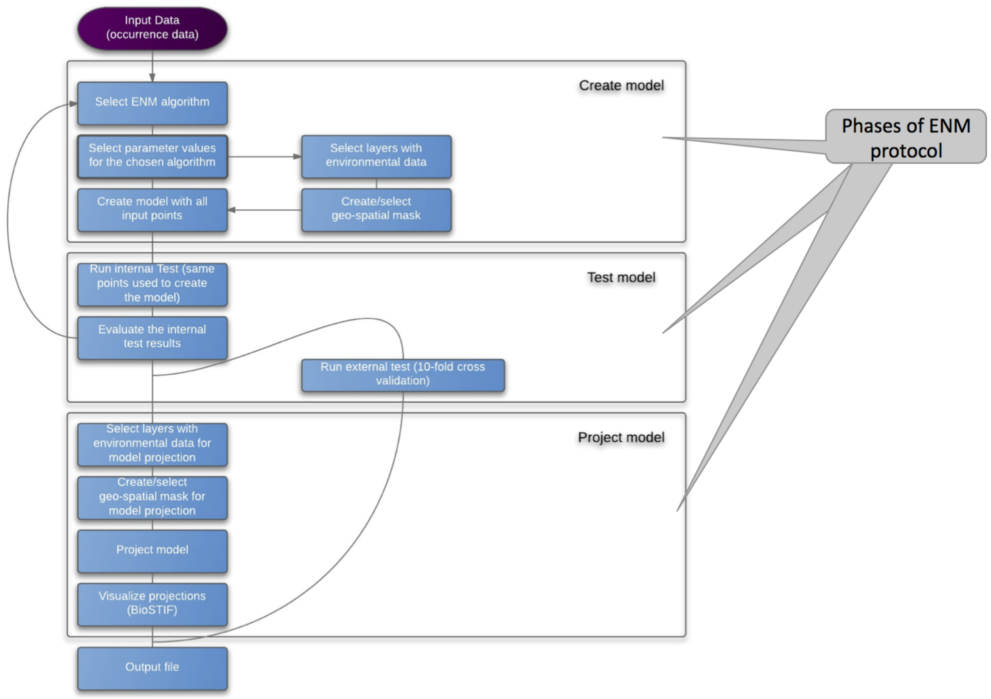

Input Data (occurrence data)
Create model
Select ENM algorithm
Select parameter values for the chosen algorithm
Select layers with environmental data
Create model with all input points
Create/select geo-spatial mask
Test model
Run internal Test (same points used to create the model)
Evaluate the internal test results
Run external test (10-fold cross validation)
Project model
Select layers with environmental data for model projection
Create/select geo-spatial mask for model projection
Project model
Visualize projections (BioSTIF)
Output file
Phases of ENM protocol

Figure 2.11: Workflow sketch depicting the ENM protocol

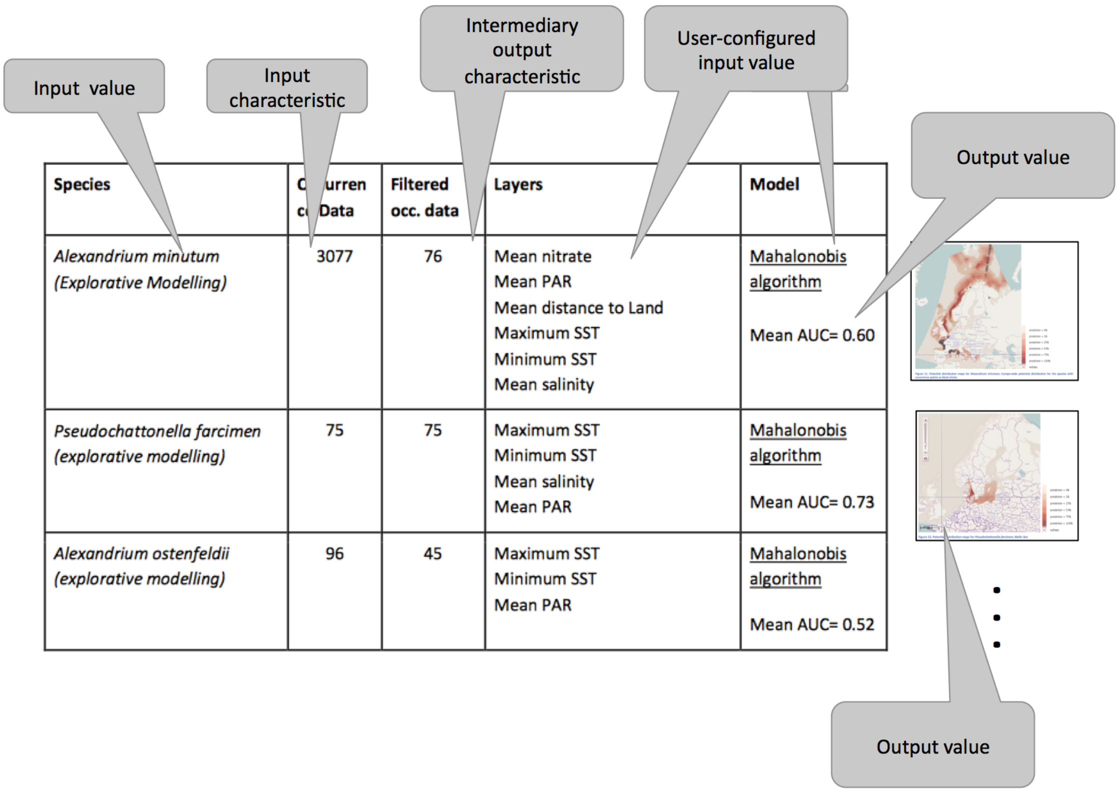
this filtered data, combined with 6 layers of environmental data has been provided as input to modelling, where the protocol has been configured to use the *Mahalonobis Algorithm*. The result of this particular projection has a mean accuracy value of *0.60* and it is visualised as a distribution map. **An important aspect of this metadata table is that it is curated using the data generated during the execution of the workflow.** For instance, the species names are extracted from first cell of each occurrence record input to workflow, whereas the record counts correspond to intermediary statistics generated as part of the workflow.

### 2.5.3 Observations on Experiment Reports

Drawing from the two examples just presented we make the following observations on experiment reports.

**Observation 1** Reports describe the factorial aspects of an experiments design.

For the purposes of this dissertation we adopt the term *Factorial Design* to refer to two patterns of experimentation:


Input value
Input characteristic
Intermediary output characteristic
User-configured input value
Output value
Output value

| Species | O[illegible]urren c[illegible] Data | Filtered occ. data | Layers | Model |
|---|---|---|---|---|
| *Alexandrium minutum (Explorative Modelling)* | 3077 | 76 | Mean nitrate<br>Mean PAR<br>Mean distance to Land<br>Maximum SST<br>Minimum SST<br>Mean salinity | Mahalonobis algorithm<br><br>Mean AUC= 0.60 |
| *Pseudochattonella farcimen (explorative modelling)* | 75 | 75 | Maximum SST<br>Minimum SST<br>Mean salinity<br>Mean PAR | Mahalonobis algorithm<br><br>Mean AUC= 0.73 |
| *Alexandrium ostenfeldii (explorative modelling)* | 96 | 45 | Maximum SST<br>Minimum SST<br>Mean PAR | Mahalonobis algorithm<br><br>Mean AUC= 0.52 |

Figure 2.12: Results and corresponding inputs from a run of ENM Protocol

- Parameter explorations, which corresponds to the repeated enactment of an experiment by using parameters from a multi-dimensional parameter space. (Different combinations of input layers or algorithm selections in the ENM study or the host gender and age in the ISA-Tab study are examples of parameters).

- Dataset sweeps, which corresponds to the repeated enactment of an experiment (with fixed parameters) over different datasets. (The enactment of the ENM analysis for different species data is an example).

The metadata tables in both the ENM and ISA-Tab examples reflect elements of factorial design. Input parameters, analytical steps and output data outline a column structure for the table. Meanwhile each individual enactment of the analytical process (corresponding to the exploration of a point in the parameter space, or a dataset sweep) populates the tables with rows.

**Observation 2** Reporting abstracts away the details of an experiment's design.

This principle applies to reports on both data and the method. When reporting data not all elements of an experimental process are reported. The ENM workflow contains several outputs such as temporarily urls and execution logs. Such results, however, have not made their way into the report as extra columns of information. Similarly in the Microarray example, the scanning instruments not only generate ".cel" files but they also generate logs of scanning completion stati, such details are deliberately omitted by the curator of the ISA-Tab table. Similarly a parameter can be omitted from a report if it stays constant for each enactment of the experiment. The abstraction principle also applies when reporting methods [LAB+06][HWB+12]. As the ISA-Tab example involves the application of standard (lab)protocols the report simply refers to them by name, and this is sufficient for common understanding among the scientific data publisher and consumer. In the ENM Workflow the data analysis protocol is novel hence needs further describing. The scientist has the option of using the workflow description for this task. As the ENM workflow is way too detailed for conveying the scientific method followed, the scientists has produced an informal sketch (diagram) depicting a simplified view of the workflow.

**Observation 3** Reporting is selective.

The factorial designs of experiments aid scientists in their explorations. Consider the ENM case where scientists run the analysis over multiple datasets belonging to different species. Results for all explored species, however, may not get reported. Some of these results may be deemed low-quality or they might simply be a test-benchmark aiding in the calibration of experimental parameters.

**Observation 4** Reports describe parameters and results with their domain-specific attributes to facilitate their understanding and interpretation by report consumers.

This principle applies to reports on data. Some attributes are inherent to an experiment's design, such attributes have a descriptive scope for all actual parameters and result obtained through the enactment of that design. For instance, stating that one output of the ENM protocol is a model accuracy statistic named *Area Under Curve (AUC)*, or stating that the output of labelling is a material named *biotin* are examples of these characterisations. While ENM adopt adhoc domain descriptions (embedded in column names or similar), ISA-Tab provides structural ontology based descriptions. Some attributes are inherent to individual elements produced by enactments of the experiment. For instance each taxonomic name in the species column in the ENM table, or each group of environmental layers are examples of such attributes.

To this end we have presented provenance as experiment reports. The ISA-Tab and ENM examples hopefully illustrated the content and structure of this category of provenance.

## 2.6 Workflow Provenance

As experiment reports make up one side of the provenance gap, workflow provenance makes up the other. We will now give general background on workflow provenance, followed by the description of Taverna system and its provenance features. Figure 2.13 presents the archetypal architecture for provenance-enabled workflow systems. In the discussions that follow we make references to this architecture and its impact on workflow provenance.

### 2.6.1 Characteristics of Scientific Workflow Systems

The contribution of scientific workflows to provenance is in two forms:

- Workflow descriptions provide the codification of a data analysis process as an executable program. As such they provide the *prospective provenance* (or the recipe) of an experiment.

- Workflow systems can trail a particular execution of a workflow to create *retrospective provenance* documenting the derivation process for results obtained from a particular enactment of an experiment.

These two forms of provenance are ultimately shaped by two factors 1) the requirements of data-oriented science as an application-area and 2) characteristics of shared scientific resources. We will now outline characteristic of scientific workflow systems that have significance in the context of our dissertation research. When discussing each characteristic we refer to the underlying reasons and how that characteristic shapes workflow provenance.

**The key contribution of workflows to scientific data processing is the systematic coordination of dataflow among analytical steps underpinned by external, and often heterogeneous resources.** As we exemplified with the ENM study workflows can bring together analytical capabilities delivered from web-services, command line tools, or local libraries. The internal details of an analytical step realised by such resources are opaque to the workflow system.

Impact for provenance: Outsourcing data processing to external opaque resources result in black-box provenance collection, opaque data lineage for which accuracy guarantees cannot be given.

**Workflow systems contain add-on components for provenance collection.** Scientists build workflows using the design interfaces in workflow systems (see Figure 2.13). Workflows are executed by execution engines, which exploit local or remote resources through appropriate accessors. During execution the workflow engine generates events (such as start and completion of each analytical step, creation and transfer of data). Provenance collection framework observes events fired and records this information as retrospective provenance.

Impact for provenance: As workflow execution and provenance collection are decoupled via an event layer there can be variations on the granularity of representations of processes and data depending on how closely the execution of workflows are monitored [MG11]. We refer to this potential differences as the granularity discrepancies between execution provenance and workflow design (prospective provenance).

**Workflows are built on data-oriented programming models.** Traditionally a program is comprised of a set of statements bound by a control specification prescribing statement execution order. What sets scientific workflows apart from conventional programs is their data-orientation. Informally a scientific workflow is a network of data *Processor* steps [2] that are related by *dataflow dependencies*, where processor invocation order is determined by the availability of data. Data driven execution is what sets workflows apart from traditional business process languages such as BPEL [LR06] and YAWL [vdAH05]. Scientific workflow systems lack the majority of complex control-flow constructs, found in process languages such as branching to alternate execution paths, splitting or synchronisation of parallel executions [MGRtH35].

Impact for provenance: Data-orientation of workflows results in a deterministic structure of computation, consequently workflow execution trails can have a predictable structure [MG11]. This brings the possibility of using the workflow descriptions (prospective provenance) as a structure to query over the execution traces (retrospective provenance).

**Workflow languages provide constructs for supporting factorial-designs.** These can be identified in three categories:

- modelling constructs for parameters and data.
- structured (collection) data-types for the modelling of data/parameter collections.
- control constructs to enable the repetitive enactment of analyses. Earlier we mentioned that scientific workflow languages lack complex control constructs. On the other hand they often provide, basic control-like features, most notably iteration (looping) constructs targeted to realise parameter explorations and dataset sweeps.

Given a workflow system supports all these constructs, then encoding factorial design **within** a workflow description may be possible [LPVM15] [DF08]. This way the workflow system becomes an infrastructure for defining parameter spaces and parameter space exploration strategies. In this dissertation we focus on this case. Meanwhile for systems weakly supporting above constructs, factorial design is to be represented and managed **outside** of the workflow. Here factorial design is either entirely undertaken by resources or this task is delegated to the users, who are expected to run workflows with changing parameters and keep track of each parameter configuration

[2] Here we adopt the terminology *Processor* from the Taverna system

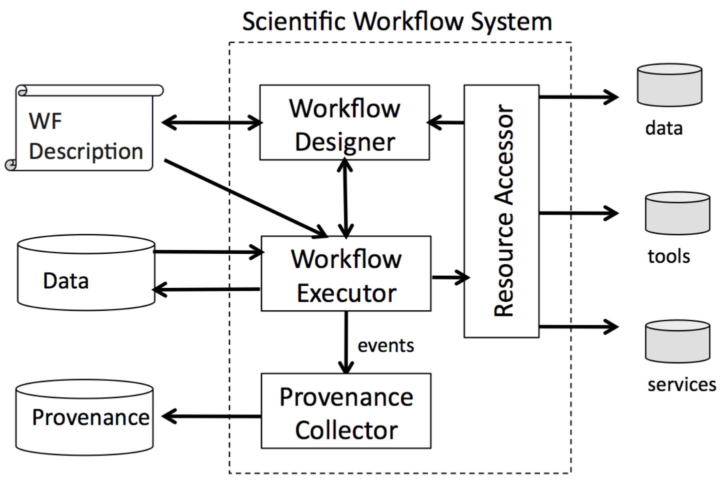


Figure 2.13: The archetypal architecture for provenance-enabled workflow systems

and each corresponding result. An comparative assessment of provenance-enabled workflow systems in this regards is given in Section 2.8.

Impact for provenance: We would expect factorial design in a workflow description to manifest itself also in execution provenance as fine-grained lineage linking each results back to individual input data and parameter combinations that are used in data's derivation. Note however, as provenance is collected by observation, there can be granularity discrepancies between workflow execution provenance and workflow descriptions. These discrepancies can affect the truthful representation of the factorial design in workflow execution provenance.

**Workflow systems with an open-world resource perspective adopt loose data typing.** A workflow system with an open-world viewpoint is one that offers the flexibility of incorporating a prior unseen resource into a workflow. In such cases, the amount of data type restrictions that can be imposed at the workflow language are minimal. Scientific resources are typically exposed on the web via (SOAP or REST) services that consume and produce data as textual message attachments (commonly typed as $text/plain$ MIME type). Or similarly, command line tools consume $string$ parameters and return $strings$, or $files$. This loose typing approach checks superficially whether the inputs are of the correct shape e.g. whether a parameter is indeed a $string$ instead of, say, $binary$. However it does not impose further constraints on the validity of data values. Taverna is the most prominent example of open-world workflow systems.

Impact for provenance: Loose typing means that the workflow engine is oblivious to the data it carries around, consequently workflow provenance provides no extra information on data. Loose typing combined with black-box collection means that provenance automatically collected from workflow executions is generic, lacking in domain-specific metadata regarding processes or data.

**Workflow systems in a controlled resource environment can impose stricter data typing.** For certain domains of science the kinds of shared resources can be enumerable. This allows the workflow systems to assume a set of domain-specific types for representing data and analytical steps. In state of the art workflow systems stricter typing is realised in two ways. The first approach is having types in the workflow specification language and type checking data values during workflow execution [GNT10] [CFS$^+$06]. Another approach is to have loose typing in the workflow language but impose domain types as an additional layer of information, often in the form of semantic annotations over elements of a workflow description [WKM$^+$10] [GRD$^+$06]. Additional type information is used to check at workflow design time for validity of combinations of data processing steps (e.g. a step expecting a genomic sequence as input can only be preceded by a step producing one).

Impact for provenance: The additional type information in the workflow description can be transferred to provenance traces as domain-specific metadata, characterising what the actual data and processes documented in provenance are [MSZ$^+$10].

**Most Workflow Systems adopt a separated scheme for storing provenance and data.** Scientific data comes in diverse forms and sizes. Workflow systems, especially those that are oblivious to data content during workflow execution use external storage facilities for data. The file system, content repositories or opaque BLOBs in databases are common approaches [CF12] [MRHBS06]. Note that in non-workflow approaches, such as biomedical software systems, joint storage of provenance and data is a recommended practice [CMD$^+$14]. Therefore our observation here is exclusively on scientific workflow systems.

Impact for Provenance: In a separated scheme data values are absent from the provenance graph, they're represented by proxy, with identifiers that are pointers to data in its associated storage system.

To this end we highlighted characteristics of workflow languages or workflow systems. The next two items are characteristics observed on existing workflow sets.

**Heterogeneity of resources cause workflows to be complex.** Complexity is a commonly observed attribute of workflow descriptions [BCBDH08] [ABL10]. The workflow for the ENM study illustrates the extent complexity can reach. Complexity is rooted in the role of workflows as integrators of heterogeneous resources [GAB$^+$12]. Workflows systematise resource wrapping and integration by making explicit the effort needed to deal with heterogeneity through adapter steps. These data grooming processes are commonly referred to as "shims" in literature [HSL$^+$04]. The abundance of adapters significantly increases complexity of the workflow description [LRL$^+$12].

Impact for provenance: complex workflow descriptions lead to complex execution provenance. There are two basic manifestations of complexity in workflow execution provenance (discussed in detailed in Chapter 4). The first is having long/deep lineage traces, and the second is having several distinct traces. Deep traces are often a manifestation of complexity of the workflow descriptions, which can be attributed to the abundance of adapter steps in workflows. Another consequence of adapters is that not only is resulting lineage deep, but also it connects multitude intermediary datasets, which are often content-wise redundant (e.g. a list of layer names followed by the same list with empty lines removed, followed by the same list coalesced into a single string and so on).

**Opaqueness of resources hinders the end-to-end encoding of factorial designs into workflows.** Earlier we identified that iteration constructs and collection types in workflow languages are intended to be used for encoding factorial design. Resources that underpin analytical steps also have such features built-in. E.g. An analysis web service may accept multiple parameters and data all at once, perform sweeping analyses and return results all at once. Such coarse integration of resources into workflows are particularly favoured to reduce resource access costs or improve performance of workflow. However, as the inner workings of resources are opaque, the sweeps within the resource are invisible to workflow execution engine.

Impact for provenance: Iteration constructs and collection typed parameters allow for fine grained modelling of processes and data in provenance and allows lineage to be used to trace from parameters to resulting datasets. However coarse integration of resources causes the earlier identified $n-by-m$ pattern to occur in workflow execution provenance.

To this end we identified certain characteristics of scientific workflows. Note that these observations are intended to motivate our research rather than provide a complete and rigorous characterisation of workflows. Scientific workflows as computational artefacts have distinctive features in their scheduling, fault tolerance mechanisms [YB05], design constructs [CG08] or design patterns [MGRtH35], which are beyond the scope of our background discussion.

### 2.6.2 Taverna Workflow System

Herein we will describe the Taverna system. The research described in Chapters 3, 4 and 6 and 7 of this dissertation introduce solutions with applicability to workflows in general, as well as solutions specific to Taverna system. Taverna provides an important part of our analysis cohort in Chapter 3 and our entire evaluation cohort for Chapters 4 and 5 and 7. Our reasons for choosing Taverna are:

- Taverna is a significant contributor to the publicly available scientific workflows. The myExperiment repository lists more than 2000 Taverna workflows at the time of writing of the thesis.
- Taverna adopts standard, technology independent vocabularies rather than custom representations for provenance [BDG+12] [BZG+15].
- The dissertation author is co-located with The Taverna development team. This provides easy access to local knowledge and tooling.

Taverna [MSRO+10] is an open-source and domain-independent workflow system used in several domains, such as genomics, environmental science, chemistry and astronomy. In its default configuration Taverna adopts an open-world approach allowing the access and composition of shared resources into analysis pipelines. Taverna workflows are represented in the $t2flow$ format which has its roots in the Simplified Conceptual Unified Flow Language (Scufl) [AFG+03]. $t2flow$ has an XML based syntax that allows the definition of the following structural elements for a workflow (pictorially depicted in Figure 2.14):

- A set of *Processors*. Processors are granules of data processing capability that ingest data from input ports and produce data to output ports.
- *Ports* for processors and workflows. Ports are named placeholders for data, they define a structure and type for the data they are expected to hold during execution. The Taverna environment (in its default configuration) provides a loose-typing approach, where a port can hold data of following types:

- Singleton values of a set of MIME types (text/plain, image/*).
- Nested collections of singletons.

- *DataLink* definitions, that designate how data flows among steps of analytical processing. In Taverna datalinks can connect workflow or processor ports to each other.

- Technical grounding information for Processors, known as *Processor_Types*. Processor Types inform the workflow execution engine on the kind of resources that underpin Processors. During execution workflow engine is responsible for delegating the execution of the Processor to the suitable resource accessor, the selection of which is done using Processor Type information. *SOAP* and *REST* services, *Command Line* tools, *Beanshell* scripts, *Subworkflows*, *String Constant* emitters, *R scripts* [R C13] are examples of Processor Types. In addition to entirely computational steps, a processor may be underpinned by a *Human Interaction*, where the user performs the data processing.

- Control specifications in the form of:
  - *Run − after* links among two processors that control execution order by delaying the start of the latter until the completion of former.
  - *Iterations* specification for processors that configure the repeated application of processor over input collections. This feature is in wide use among Taverna workflows and is the powerful feature to specify factorial designs. When a processor accepts multiple inputs that are collections, Taverna gives the possibility of configuring selection of input combinations from lists using list *Dot* and *Cross* product operators. These operators help the workflow designer to outline a strategy to explore the parameter space.

The distinguishing feature of Taverna that also underlies its widespread adoption is its openness in resources (data and analysis steps) allowed into workflows. The default Processor Types of Taverna allows access to any resource that supports a standard access interface (such as command-line or web service). Taverna's domain neutral typing (comprised only of basic MIME types) and very basic data-structuring (through collections-items) renders any kind of data potentially processable via workflows. In addition to the (default) open approach, Taverna supports the development of controlled resource approach through Plugins and Components.

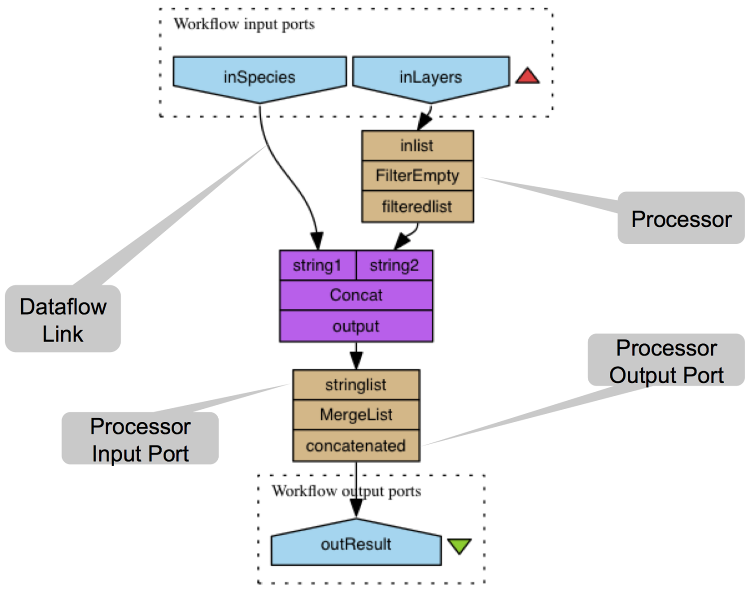


Figure 2.14: Constituent of a Simple Taverna 2 Workflow

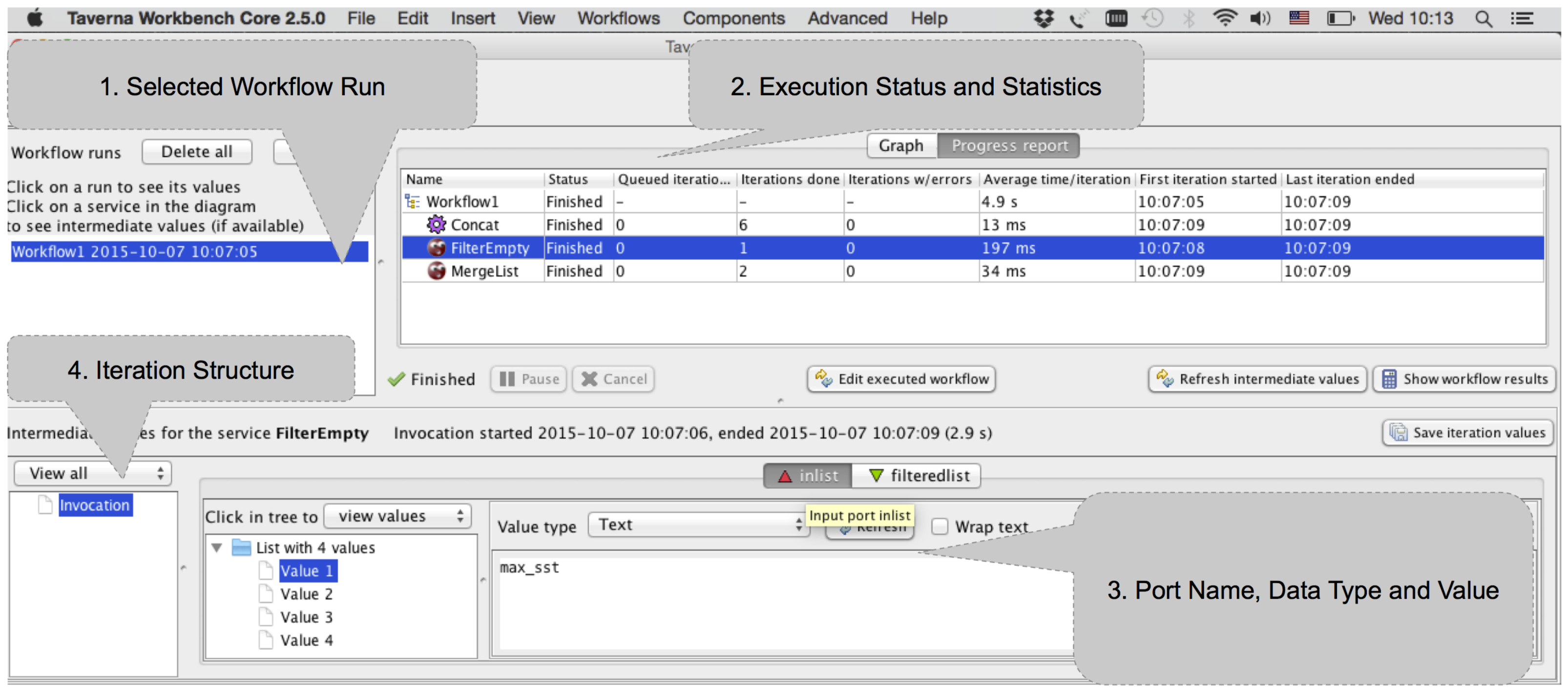
Taverna allows extending the set of supported Processor Types, through Plug-ins [MSRO$^+$10] and Components [3]. These serve a multitude of purposes:

- They enable easy integration of specialized resources such as Grid job submission interfaces into the workflow environment [TMN$^+$10] [KMB08].
- They enable controlled typing to be imposed over processors. Plug-in based processors are marked up, with domain-specific vocabularies to denote their function and the types of data they produce and consume. Using this information the plug-in infrastructure emulates a strongly-typed environment during workflow design by performing validation of processor compositions [WKM$^+$10].

The Taverna system provides user interfaces for designing and executing workflows. In Figure 2.15 we provide a screenshot of Taverna Results Panel displaying execution information for the earlier workflow of Figure 2.14. Execution is initiated by the assignment of data values to workflow input ports. Each such assignment and subsequent execution is consider a particular *Run* of that workflow (Box 1 in Figure 2.15). Depending on iteration configurations, each processor in a workflow may execute a different number of times within one run of the workflow. Taverna allows

[3]The primary difference between plug-ins and components is that the former is implemented with local Java libraries, whereas the latter with sub-workflows.

the user to view the number of iterations per processor, their completion status and statistics (Box 2). The port names and port types, and the values of data that appears are delivered through those ports can be seen (Box 3). Through the result pane users are given the ability to save result data and export execution provenance in various representations including the OPM [MCF+11] and PROV [BDG+12] models.



Click on a run to see its values
Click on a service in the diagram
to see intermediate values (if available)

Workflow1 2015-10-07 10:07:05

| Name | Status | Queued iteratio... | Iterations done | Iterations w/errors | Average time/iteration | First iteration started | Last iteration ended |
|---|---|---|---|---|---|---|---|
| Workflow1 | Finished | – | – | – | 4.9 s | 10:07:05 | 10:07:09 |
| Concat | Finished | 0 | 6 | 0 | 13 ms | 10:07:09 | 10:07:09 |
| FilterEmpty | Finished | 0 | 1 | 0 | 197 ms | 10:07:08 | 10:07:09 |
| MergeList | Finished | 0 | 2 | 0 | 34 ms | 10:07:09 | 10:07:09 |

Intermedia... es for the service **FilterEmpty** Invocation started 2015-10-07 10:07:06, ended 2015-10-07 10:07:09 (2.9 s)

```
max_sst
```

Figure 2.15: The Screenshot of Taverna Execution Panel Showing Intermediary and Final Results.

The formal syntax and semantics for Taverna was first given by Turi et al [TMG$^+$07] using Computational Lambda Calculus. In this formalisation a workflow is defined in functional terms, where each data processing step is represented by an opaque function accompanied by constructs for list building, applications of functions to items in lists. Sroka et al [SHMG10] have later provided custom formalism for the Taverna version 2 system to cater for its data streaming capability. Finally in [MPB10] Missier et al have provided a more practical functional notation for selected parts of Taverna's behaviour. For the purposes of our research, in Chapter 6 we will adopt Missier's approach to denote the operational behaviour of Taverna and show how our workflow analysis rules exploit this behaviour.

## 2.6.3 Provenance Traces from Taverna

Taverna uses technology independent vocabularies, namely PROV [BDG$^+$12] and Wfdesc [BZG$^+$15] to represent its provenance. We provide an illustration with our running example. When list filtering is realised as a processor in a Taverna workflow, the provenance recorded for a particular invocation of this processor would be as given in Figure 2.16. The figure follows W3C's layout conventions for presenting PROV graphs [Gro12] that dictates shapes for different provenance elements, and an order for their layout. PROV builds on the core elements of *Activity*, *Entity* and *Agent* (corresponding shape conventions are given as legend in Figure 2.16). A processor's invocation is documented as an *Activity* with designated start and end times. Data is recorded as *Entities*. The usage and generation relationships, are further qualified by specifying the *role* that the each has played in the computation ( $d1$ and $d2$ have fulfilled the roles of *criteria* and *inList* and $d3$ has fulfilled the role of *outList*). Note that these role qualifications come from the input/output port names of the filtering processor in the workflow description.

An important benefit of using PROV for modelling is its well-defined set of constraints and inference rules that allow validating provenance and inferring refined provenance such as *process causalities* and *data lineage*. PROV provides the $wasInfluencedBy$ relation to assert opaque lineage among (data) entities (depicted with dashed arrows in Figure 2.16).

The trace in Figure 2.16 also informs us who performed the activity through PROV's *Agent* construct. Here the filtering activity is undertaken by the Taverna workflow engine. It is also stated that Taverna engine performed filtering using a plan named *FilterEmpty*. The *Plan* construct of PROV corresponds to prospective provenance.

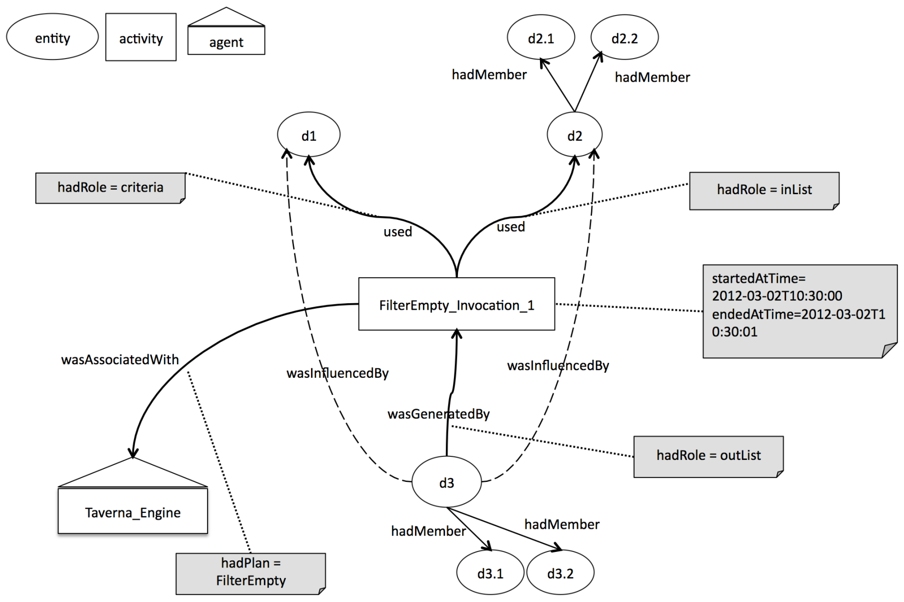


Figure 2.16: Retrospective Provenance of List Filtering represented in PROV

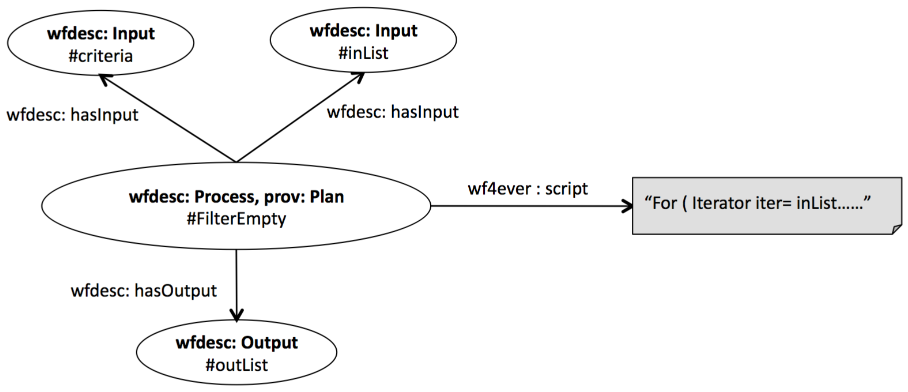


Figure 2.17: Prospective Provenance of List Filtering represented in Wfdesc

Note that PROV provides the *Plan* as an atomic construct and has no further vocabulary to outline plan details. This is where the Wfdesc model comes in. Wfdesc allows the abstract specification of a workflow's details through the use of *Process*, *Input*, *Output* and *DataLink* constructs. The Wfdesc representation of the list filtering processor is given in Figure 2.17. The figure displays a "node and directed arc" diagram, a method adopted for visualising RDF graphs [WLC14], where an RDF statement is denoted with a "node-arc-node" pattern in the graph. Note, however, for brevity we use a more contact depiction of information in Figure 2.17. RDF statements denoting *type* information are given in compact form as additional labels over nodes (in bold font). This compact form has also been adopted by the developers of the Wfdesc model [BZG+15].

As this processor is realised with a Beanshell script, its details are not further elaborated, for information purposes only the source script is given as a literal (see Figure 2.17). In cases a process is realised with a sub-workflow, Wfdesc allows us to model its sub-processes as distinct nodes in the RDF graph with their own properties.

Within Taverna system there are no granularity discrepancies between prospective and retrospective provenance. Data appearing at workflow/activity ports of collection type are modelled in execution provenance as entities that are PROV *Collection*s linked to their items with PROV *hadMember* relation. Similarly for processes, execution provenance is recorded at a fine grain reaching into iterations and sub-workflows.

To sum up, Taverna inherits all characteristics identified in Section 2.6.1. It provides:

- prospective and retrospective provenance,
- opaque data lineage,
- fine-grained modelling of data and processes in provenance,
- generic provenance, without any domain specific metadata characterising the data and processes,
- a separated storage scheme for data and provenance.

In addition, when we look at provenance cohorts from Taverna [BZG+13] we observe they exhibit the complexity characteristic, and the coarse integration of resources into workflow design.

## 2.7 Formulating the Provenance Gap

To this end we introduced characteristics of provenance on either side of the gap, namely Experiment Reports and Workflow Provenance. We now revisit earlier observations on reports, take them as requirements and discuss the opportunities and challenges that workflow provenance has in meeting these requirements.

**Observation 1. Reporting Factorial Designs.**

- Opportunities: When identifying workflow systems' general characteristics we stated that they (in varying levels of support) provide constructs to support factorial designs. This creates an opportunity to use workflow execution traces in reporting data.
- Challenges: Note however, we also stated that **granularity discrepancies** between prospective and retrospective provenance may affect the truthful representation of factorial design in workflow execution traces.
- Our Challenge here lies in understanding the varying levels of support in scientific workflow systems, and to identify which systems support a truthful representation of factorial design. A comparative survey aiming to provide an understanding of the level of support in different workflow systems is provided in Section 2.8.

**Observation 2. Selectivity in Reporting.**

- Opportunities: Assuming that we have the constructs in place for encoding factorial designs and its truthful representation in workflow execution provenance, then provenance provides traceability among experimental parameters or input datasets to results. This provides an opportunity to selectively create reports via (the earlier mentioned) *Provenance Driven Data Selection* (PDDS). This is a provenance access pattern, which involves (1) Identification of parameter/data node(s) of interest (2) Using lineage to reach to result nodes descending from identified data or parameter.
- Challenges: Existing workflow systems have weak support for PDDS. The reasons are two-fold. First obstacle comes from **separate storage scheme of data/-parameters and provenance**. This blocks the immediate accessibility of values

of data or parameter nodes, hindering their identification in a provenance graph. The second challenge is the **coarse integration of external scientific resources into the workflow**, which hinders the end-to-end modelling of factorial designs. Such integration causes the $n - by - m$ pattern in lineage reducing its accuracy in linking parameters with descendant results. There exists no capabilities in workflow tooling to detect or anticipate such patterns.

- Our Challenge here is two fold. First is to assess the fitness of standard workflow provenance in supporting PDDS. In Chapter 5 we provide a case study that aims to provide such an assessment. The second challenge is for a particular system, specifically Taverna, 1) to understand the circumstances in which the n-by-m pattern affects PDDS, and 2) to develop techniques to detect broken factorial design in a workflow description. Chapter 6 describes analysis techniques that target this challenge.

**Observation 3. Abstraction in Reporting.**

- Opportunities: Workflow descriptions due to their simple data-oriented programming models have the potential to be used as reports on experimental method. Workflow descriptions also provide an anticipated structure, a blueprint for all future execution traces to be obtained per workflow execution.

- Challenges: We identified that workflows, especially those operating in open resource environments can be **complex** due to heterogeneity of resources, and the data adapter steps involved. One common solution adopted by scientists for abstraction, in Taverna as well as other systems we later review in Section 2.8.2 is to design workflows in layers (of sub-workflows), where lower layers contain the detail of resource integration, and the upper layers abstract away from detail. Meanwhile, **abstraction through layered design is a manual, scientist-driven activity**. The lack of support in tools for automated abstraction is due to the black-box nature of individual steps in workflows. As steps are black-boxes the workflow system has no actionable indicator on the (in)significance of steps.

- Our Challenge here is twofold; first is to understand in detail the contributors to complexity by analysing existing workflow sets and to understand whether we can break the black-box assumption so as to obtain indicators of activity significance. We tackle this challenge with an empirical analysis reported in Chapter

3. The second challenge is to see whether additional (grey-box) information on workflow activities can be used in computationally assisted workflow abstraction. We describe our abstraction techniques in Chapter 4.

**Observation 4. Using Domain-specific Attributes in Reporting.**

- Opportunities: Earlier we identified that due to loose typing and black-box provenance collection, workflow provenance is inherently generic. Workflow descriptions provide a vocabulary of analytical processors, data and parameters, as well as low-level implementation-oriented type information for those. A similarly generic viewpoint is adopted in workflow execution provenance. This characteristic of provenance stands as an important blocker for its use in reporting. When we look beyond provenance we observe that domain-specific information is often implicitly available, it is buried in processor names or data values.

- Challenges: Domain specific information is needed for various purposes, not just reporting. We identified that one way (in the context of workflow validation) to supply this information is to semantically annotate processors, and ports in a workflows description to denote their domain-specific types [WKM$^+$10]. And as we surveyed existing workflow tooling in Section 2.8.2 this annotation process is primarily manual. Such annotations make explicit the information implicit within the workflow description (e.g. processor names). For reporting, these annotations could be used as attributes that are inherent to the experiment's design. Note, however a second category of attributes we identified in reports are those inherent to entities that get created per enactment of experiments. This information is typically buried in parameter or data values. Making this information explicit would mean annotating individual results produced from workflow execution by interpretation of parameter and data values. In our survey of workflow systems we were not able to find any manual **annotation features that targeted workflow execution provenance**. Understandably so, as the number of elements within a workflow design are fixed and poses a manageably sized task for a user to annotate. Whereas the number of elements in an execution trace can be very large (considering parameter explorations and data sweeps) prohibiting their manual annotation.

- Our challenge here is the provision of domain-specific attributes that describe elements in a workflow execution provenance. Due to the voluminous nature of this layer of provenance any potential solution should minimise human involvement. Any automated solution would require 1) indicators on the sources of implicit domain-specific information and 2) an annotation logic that would make this information explicit. Consequently our challenge here is to devise techniques to systematically obtain these two pieces of information and to devise an automation approach and architecture that uses this information to annotate workflow provenance.

## 2.8 State of the Art in Workflow Provenance

In the literature review provided herein we survey state of the art workflow systems to assess their level of provenance support against the requirements of reporting. For completeness we also surveyed provenance in other data processing instruments [BF05], namely Scripts, Command-Line Tools and Databases. In order not to disrupt the flow of text the non-workflow categories are in given in AppendixA. We assess workflow systems against the following:

- What are the distinctive aspect of each system, and how is provenance currently used?
- What is the level of support for encoding factorial designs? What are relevant constructs? At which granularity are processes and data represented? Are there granularity discrepancies between retrospective and prospective provenance?
- Is provenance driven data selection supported? Is the data-wise $n-by-m$ pattern observed in traces? What are the storage schemes for data and provenance?
- Is there support for the addition of domain-specific metadata to provenance and how it is achieved?
- Are there and what are the abstraction mechanisms for provenance in these systems?

### 2.8.1 Workflow Systems

We review workflow systems with provenance support, namely Taverna, Kepler, Galaxy, Wings/Pegasus and Vistrails. The KNIME system is notably excluded as, despite having provenance-like features, such as execution monitoring and result caching, it does not provide any means to export or access provenance information. Also notably excluded are KARMA [SPG08] and PASOA [MGM$^+$08b]. These are pioneer provenance systems with particular focus on workflows in Service-Oriented environments. KARMA can trace execution of BPEL workflows [LR06], and can record message exchanges among services participating in a workflow [Apa09]. We excluded KARMA due to BPEL not being a workflow language of choice for most scientists. PASOA, on the other hand, is not a workflow system per se. It provides a protocol for collecting and collating provenance from multiple disparate parties involved in service-based computing scenario. Very brief overviews of the surveyed systems, other than Taverna (which was given in Section 2.6.2) are as follows. The survey's narrative follows from Table 2.3. (We use circles to denote level of support for reporting requirements. Full circle denotes full-support, half circle denotes partial support and empty circle denotes no support.)

**Wings** [GRK$^+$11] is a "semantic" workflow system that has found use in Life Sciences, Text Analytics and Geo-science domains. Wings adopts a tightly controlled world-view of resources as it requires them to be made part of a catalogue prior to use. During cataloging Wings expects semantic descriptions and constraints about resources to be provided. An example description could state that an analytical step is a $BayesianClassifier$, a constraint could state that input and output data of the classifier should be about the *same* study subject. Wings combines semantic resource descriptions, with workflow descriptions to check for validity of resource compositions in workflows.

**Galaxy** [GRH$^+$05] is a web/cloud-based data-analysis platform used widely in biomedical disciplines. Galaxy facilitates the combination of analytical command line tools with datasets in workflows. Similar to Wings, Galaxy has a tightly controlled resource environment and is similarly backed by a catalogue. During catalogue introduction, Galaxy expects scientists to supply domain-specific, structured descriptions denoting dataset types or tool functions. Galaxy utilises these descriptions when publishing workflow execution histories as online supplementary material (experimental bundles discussed in Section 2.3.3).

**Vistrails** [CFS$^+$06] supports workflows for Scientific Visualisation. An inherent characteristic of such workflows is that they are the output of an iterative and exploratory design process. The distinctive feature of Vistrails is the tracking of this exploratory process. By keeping track of all intermediate designs Vistrails allows scientists to compare and contrast execution results of different design alternatives to further progress with the design.Vistrails adopts a partially-controlled resource environment. It provides a built-in library of visualisation functions and also supports the incorporation of prior unseen functions (e.g. web service base analyses) into workflows.

**Kepler** [LAB$^+$06] is another workflow system that has found use in multiple scientific domains. Kepler uses Ptolemy II as its execution engine, which is a framework with support for various (sub)modes of computation under the umbrella of the Dataflow Architecture [NLG99]. Each flavour of dataflow computation is manifested in Kepler as a Director, that dictates the mechanics of dataflow among analytical steps (e.g. streaming vs synchronised clock). Similar to Vistrails, Kepler adopts a hybrid perspective on resources by providing a built-in library as well as allowing incorporation of external resources. Kepler utilises provenance to perform smart workflow re-runs [ABJF06], where sub-parts of a workflow can be selectively re-executed using cached outputs of upstream portions of the workflow.

| System | Factorial Design | | | Provenance-Driven Data Selection | | | Abs. | Domain -Specific Metadata | | End-Use |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Process Granularity** | **Data Granularity** | **L.o.S.** | **Prone to n-by-m** | **Storage Scheme** | **L.o.S.** | | **Wf Description** | **Wf Execution** | |
| Taverna | Processor<br>Iteration<br>Sub-workflow | Fine | ● | Y | Unified Singleton Parameters,<br>Separate Collection Parameters<br>Separate Data | ◓ | Y | Semantic Annotation<br>Text Annotations<br>Parameter Values | N | Audit<br>Experiment Bundles |
| Kepler | Actor<br>Ramp, Repeat, Loop<br>Sub-workflow | Coarse Data,<br>Coarse Parameter | ○ | N | Separate Data,<br>Unified Parameter | ● | Y | Semantic Annotation,<br>Library Datatypes,<br>Parameter Values | N | Efficient Re-run |
| Kepler (COMAD) | Actor<br>Data Bindings | Coarse Parameter,<br>Fine Data | ◓ | Y | Separate Data,<br>Unified Parameter | ◓ | Y | Key-value pairs<br>Library Datatypes | N | Debugging |
| Galaxy | Steps | Coarse Data,<br>Coarse Parameter | ○ | N | Separate Data,<br>Unified Parameter | ● | N | Attributes | Propagated Attributes | Experiment Bundles |
| Wings | Component<br>Component Collection | Coarse parameter,<br>Fine Data | ◓ | Y | Separate Data,<br>Unified Parameter | ◓ | N | Semantic Annotation,<br>Parameter Values | Propagated Annotations | Workflow Validation |
| Vistrails | Modules<br>Parameter exploration,<br>map/fold<br>sub-workflow | Fine Parameter,<br>Coarse Data | ◓ | N | Separate Data,<br>Unified Parameter | ◓ | Y | Text annotations,<br>Library Datatypes,<br>Parameter Values | N | Comparison<br>Exploratory Design |

Table 2.3: Comparative Table of Workflow Provenance (L.o.S and Abs. stand for Level of Support and Abstraction respectively).

### 2.8.2 Assessing Systems Against Reporting Requirements

#### 2.8.2.1 Support for Factorial Designs

A common observation for all systems is that they record process provenance at the level of granularity of individual analytical steps embodied in the workflow. These are called *Processors* in Taverna, *Actors* in Kepler, *Modules* in Vistrails, *Components* in Wings and *Steps* in Galaxy. In addition, provenance collection in all systems can see into control constructs, be it iteration, or sub-workflows and record the provenance of steps within. Meanwhile, the granularity of modelling of data within workflow descriptions differ as well as its representation in execution provenance, and it is a critical factor in determining a system's support for encoding factorial designs.

**Taverna** provides full support in this category. As discussed earlier in Section 2.6.2 Taverna supports fine-grained modelling of workflow inputs and outputs as nested collections both at prospective and retrospective provenance layers. Collections are the main driver of iteration in Taverna. Input cardinalities determine whether a *Processor* is to iterate or not. Iterations over multiple input collections can be configured via *dot* and *cross* product operations, which gives scientists the flexibility to define their own strategies to explore a parameter space. Taverna does not differentiate between parameters and data, which means both dataset sweeps and parameter explorations are built using the same language constructs.

**Wings** provides partial support in this category. Wings permits fine-grained modelling of input data as *Files* and *FileCollections* and supports (coarse-grained) parameters as first class modelling elements in workflow design. Granularity of modelling data is uniform across workflow specification and execution traces in Wings. This system provides *ComponentCollections* for iteration and uses this feature primarily for dataset sweeps. Unlike Taverna, parameter exploration strategies cannot be implemented as parameters are coarse entities and iterations are not configurable.

**Vistrails** provides partial support in this category. Vistrails system has parameter exploration capability built-in. More specifically it supports the definition of a multi dimensional parameter space, upon which the entire workflow is iterated. On the other hand custom strategies for exploring the parameter space is not possible. In addition to parameter collections, Vistrails supports structured typing for data with *list* types. Iteration (within a workflow) over lists is supported by using higher-order modules such as *map* and *fold* that encapsulate analysis modules and input lists to be swept.

While both components of factorial design, i.e. the dataset sweeps and parameter explorations are possible in workflow design, this does not get fully reflected to execution provenance as list typed data is recorded coarsely.

**Kepler**'s support in this category changes depending on different Directors used.

- Kepler with its default set of Directors provides weak support for encoding factorial design into a workflow specification. As the workflow language Kepler supports structured types (*arrays*) for data, however, such structured data gets modelled coarsely in execution provenance. Kepler supports the modelling of Parameters as distinct workflow design elements, however collection typed parameters are not supported. Kepler provides iterated analyses via $ramp$, $repeat$ and $feedback-loop$ constructs, however as both data and parameters are modelled coarsely, this capability is insufficient on its own to represent factorial designs.

- The Collection-Oriented Modelling and Design (COMAD) director of Kepler provides improved, yet partial support for factorial designs. COMAD approach is intended to improve data granularity modelling of Kepler by providing support for nested collections of datasets to be recognised as first class elements in Kepler workflows. The COMAD director provides a very simplistic model for modelling analytical steps, lacking constructs for iterations or sub-workflowing [BMWL07]. On the other hand alternative mechanisms, called $databindings$ are in place. In this approach inputs, intermediary and final output data is to kept in one single hierarchical data structure that is passed through every analytical step in the workflow in an assembly line manner. Each analytical step has a data binding specification (somewhat similar to an XPATH query over XML) that specifies the data of interest for that step. When a binding specification is bound to multiple data sub-hierarchies this can be used to sweep a particular analysis over collections of input data. While data is modelled at a fine-granularity, parameters are coarsely represented, hence parameter explorations cannot defined and managed within workflows.

**Galaxy** system does not provide support for encoding factorial design into workflow descriptions. Here data/parameters are modelled coarsely at both the workflow description and workflow execution levels. Galaxy also lacks control-constructs such as iteration or sub-workflows.

#### 2.8.2.2 Support for Provenance Driven Data Selection (PDDS)

The level of support in PDDS corresponds to the ability of the provenance consumer to 1) locate data/parameter nodes of interest and 2) access data products that are derived from the designated parameters/data via lineage. These two capabilities are dependent on storage schemes, and factorial design support respectively.

We observe that level of support for factorial designs becomes a double-edged sword when it comes to PDDS. Systems that support factorial design within workflows, such as Taverna, Wings, Kepler (COMAD) and Vistrails bear the potential of creating fine grained lineage linking parameters/data to result data. On the other hand, in systems like Wings, Taverna, Kepler (COMAD) factorial design is to be created by the user, and there are no restrictions in place that would prevent the user from coarsely integrating resources into a workflow design. As discussed earlier, coarse integrations cause the data-wise $n-by-m$ pattern to occur in provenance. Meanwhile Vistrails does not leave the encoding of factorial design to the user, instead it provides this as a built-in and restricted feature, thereby guaranteeing that, for parameter explorations, the n-by-m pattern is avoided within the execution provenance of a workflow. Due to their lack of support for factorial design, the Galaxy and Kepler systems are not prone to the data-wise $n-by-m$ issue (within workflows).

Another determinant of PDDS support is the storage scheme adopted. This is most critical for storing parameters. When parameter values are stored jointly with provenance, these values can be predicated on in provenance queries. All workflow systems support unified storage schemes when it comes to parameter values. Taverna system presents a disadvantage here. Despite having support for collection-typed parameters, values of those are not stored together with provenance.

#### 2.8.2.3 Support for Abstraction

Complexity manifests itself in both workflow descriptions and workflow provenance. Our comparative survey shows that Taverna, Kepler, Vistrails systems support the sub-workflowing construct to allow the user to manually create abstractions in the form of layered workflow designs. Abstraction at the design level helps cope with complexity of workflow execution traces. Note from earlier that we discussed the complexity of execution provenance manifesting as 1) long lineage traces or 2) multiple distinct/interlinked traces among results. Workflow systems provide interfaces ( often called execution or result panels) to display result datasets and their lineage as captured in workflow provenance. In these interfaces the workflow descriptions is typically used as a layer of abstraction (a browsing guide or an access index) over traces. Such interfaces are available in Taverna, Wings, Vistrails and Galaxy systems.

In addition to the support built into workflow tooling. There has been research-oriented efforts on provenance abstraction in the context of scientific workflows. The Provenance Browser [ABL10] and ZOOM [BCBDH08] are the two notable systems in this regard. Provenance Browser exploits, in addition to lineage, the hierarchical structure present in (nested) data collections and nested processes to give dimensionality to visual exploration workflow provenance graphs. Capabilities of Provenance Browser has been showcased over traces of the Kepler (COMAD) workflow system. ZOOM on the other hand focuses on creating on computationally assisted creation of layered designs and showcases how layering leads to shorter lineage traces. The state of the art in provenance abstraction in general is provided in detail in Chapter 4, where we discuss our workflow abstraction techniques.

#### 2.8.2.4 Support for Domain-Specific Information

Domain-specific information can be provided with workflow provenance in two layers; the workflow description and the execution trace. At the description layer metadata outlines the design-bound aspects of experimental elements. Examples of such characteristics are the type of function performed by an analytical step or the domain types of parameters and of prospective input/output datasets. Domain specific information at the workflow description layer is conveyed in three basic ways 1) via built-in type information, 2) by add-on annotations, 3) by capturing design-bound (static) parameter values. Vistrails and Kepler systems provide a comprehensive type space that allows domain specific typing of workflow elements. All workflow systems support annotation of workflow descriptions. These annotations range from basic textual mark-up as in Taverna and Vistrails to semantic mark-up created with domain-ontologies, as in Taverna, Kepler and Wings systems. Workflow systems, with the exception of Taverna and Kepler (COMAD), mandate that all parameter values are hardcoded into a workflow design. These values also contribute as domain-specific information at the workflow description layer.

A distinctive aspect of Galaxy and Wings is their mandate on domain-specific metadata due to their controlled viewpoint on resources. Here, both the input datasets and analytical tools carry annotations created prior, during resource introduction (to respective catalogues). A mandate on metadata can be seen a drawback as metadata creation requires scientist's time and effort, both limited resources [TF08]. On the other hand, while collecting descriptive metadata about resources these systems also collect configurations/constraints regarding metadata that resources should have when brought in composition with other resources. Wings and Galaxy puts this information to semi-automate resource annotation. During the execution of workflows annotations on input datasets are propagated to newly created output datasets.

Galaxy supports the propagation of annotations from input datasets to derivative datasets (depending on tool configurations). The motivation for having domain-specific metadata in Galaxy is to be able to later use it when publishing decorated provenance traces together with output data. Whereas in Wings annotations on workflow description is used for validation of processor compositions, or processor vs input data compatibility. We survey the provenance annotation problem and the state of the art closely in Chapter 5.

## 2.9 Chapter Conclusion

In this chapter we introduced the backdrop for our dissertation research. We elaborated on data-oriented investigation lifecycle and identified the different categories of provenance in it. We identified that **Experiment Reports** and **Workflow Provenance** are the two types of provenance information that have gained increased research and development attention. Experiment reports are a by-product of efforts to systematise the reporting of datasets, meanwhile workflows and workflow provenance result from efforts that systematise the way computational data analyses are performed. Even though both attempt at systematisation, reporting makes limited use workflow provenance.

We then looked closely to both categories of provenance. We observed that although both attest to the same philosophy of provenance (i.e. describing the origin and derivation history) they operate over different levels of abstraction. Experiment reports require a domain-oriented high level view point whereas workflows adopt a generic implementation oriented view point.

We projected characteristics of experiment reports on to workflow provenance. We observed overlaps between characteristics as opportunities and differences in characteristics as challenges. We then performed a much detailed analysis by assessing provenance capabilities of individual scientific workflow systems against reporting requirements.

# Chapter 3

# Scientific Workflow Motifs

## 3.1 Chapter Introduction

This chapter reports the results of an empirical analysis of real-world workflows. The major output of this analysis is Workflow Motifs which can be defined as:

> a domain-independent, non-exhaustive taxonomic categorisation of the nature of activities and data within workflows. Motifs characterise activity function, role of data, and high-level workflow design patterns.

The analysis depicts the current landscape of workflow-based scientific data processing and provide empirical evidence that a) substantiates some of the observations made on workflow provenance in Chapter 2 and b) informs the techniques that we present in Chapters 4, 6 and 7 on the kind of treatment workflow provenance requires to overcome its shortcomings. We layout the analysis setting in Section 3.4 by introducing the study cohort and methodology. This is followed by the introduction of Motifs and Motif occurrence statistics in Section 3.5. Related work is given in Section 3.9. Finally we revisit our observations on workflow provenance from Chapter 2 in light of Motifs in Section 3.8 and discuss how Motifs could inform techniques on workflow provenance abstraction and annotation.

Parts of work described herein has been published as a conference and a journal paper.

- D. Garijo, **P. Alper**, K. Belhajjame, et al. Common motifs in scientific workflows: An empirical analysis. In Proceedings of the 8th eScience Conference, pages 1-8, Chicago Illinois USA, October 2012, IEEE.

- D. Garijo, **P. Alper**, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble. Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems, 36:338351, 2014, Elsevier.

The conference paper reports our analysis over a cohort limited to workflows from the Taverna [MSRO$^+$10] and Wings [GRK$^+$11] systems. This analysis has later been extended to include workflows from Galaxy [GRH$^+$05] and Vistrails [CFS$^+$06] and has been published as a journal paper. Both papers share equal joint contribution of authors P.Alper and D. Garijo, the journal paper has been marked explicitly to denote joint first authorship.

## 3.2 Motivation for Motifs

From a technical stand-point, the prime cause of provenance genericity and also the major bottleneck in the development of automated solutions for provenance annotation and abstraction is the **black-box assumption** in workflow provenance collection.

For the development of computational techniques to aid in provenance annotation and abstraction, one would require actionable information on the inner workings of activities. This information is required;

- **during abstraction to elicit whether an activity documented in a provenance trace is report-worthy or not.** Workflow descriptions constitute experimental metadata that outlines the analytical method/protocol followed. In cases of analyses involving heterogeneous resources from distributed and autonomous providers the workflow's integration role becomes too elaborate (with several adaptation steps required for gluing together resources or preprocessing datasets) to the degree that it overwhelms the scientific data analysis protocol, making it less apparent in the workflow description. In a black-box approach there is no differentiation between an adapter versus a scientifically significant analysis step in a workflow.

- **during annotation to identify which (data/activity) elements in a provenance trace hold information on data's creational context.** The creational context refers to the parameter values, activity configuration settings used in conception data produced from workflow executions. Workflows are a medium that witnesses first-hand the process of creation or collection of scientific datasets. Yet their ignorance to the inner workings of activities provides a dry documentation lacking any explicit (domain specific) descriptors of context. Ignorance also prevents us from understanding whether all data artefacts within a trace are minted afresh or whether they are copies of some other data.

This thesis is comprised of investigations on the treatment of workflows and their provenance towards provisioning experimental metadata [ABGK13]. More specifically, we investigate in Chapter 4 the use workflow descriptions themselves as source of experimental metadata and tackling complexity with **workflow abstraction** and in Chapter 7 the use of workflow provenance as a medium to **collect and propagate domain specific annotations** over data.

A pre-requisite for the development of such solutions is to have an understanding of data processing inside workflows that is beyond the black-box. Breaking the black-box assumption has been an attractive topic in workflows and provenance research. We shall also comment here on our chosen path to break the black-box. Two main strategies can be adopted here:

- Assuming white-boxes, where data is processed with a restricted but well-defined set of operations, such as relational queries or dataflow operators [ADD+11] [ICF+12]. Operations, whose execution semantics are known by a provenance collection framework allow precise tracing procedures to be built. A major downside of this approach is that it is highly restrictive on the structure of data and the kinds possible operations. As shall become apparent from our empirical findings such a restrictive assumption is clearly not possible over real-world workflows.

- Assuming grey-boxes, which is our chosen approach where data processing is partially characterised by an abstraction of activity function and for certain cases implied relations among inputs and outputs, but leaving out details as to the specific execution semantics. Note that such a transparency may not guarantee accurate fine-grained traceability or may not fully qualify every possible lineage relation in a provenance trace. However it may be used to characterise a wider variety of activities that have a visible footprint in real-world workflows. More importantly the transparency provided may permit the detection of points in provenance from which metadata can be provisioned and also the propagation of such metadata among datasets, which are interlinked with partly qualified relations.

To this end we identified above our own motivations in breaking the black-box assumption. As the analysis reported in this chapter was done in collaboration with other scientists (we elaborate in Section 3.4.1), we shall also briefly comment here on their motivations. Workflows as automation artefacts are the outcome of a non-trivial design process that requires significant user expertise and effort. As a consequence a common aspiration in workflow research is for workflow design to be incremental, where users build on or re-use workflow descriptions of scientists within their domain of study [SBCBL14] [GFG+09]. In order to devise techniques that could help users in finding similar workflows one needs evidence of commonly encountered activities and activity combinations and evidence of incremental development and insight into how it happens in practice. Our collaborators [GCG13] were seeking such evidence and insight in the joint analysis.

## 3.3 Anticipating Motifs

Our empirical analysis has not occurred without precedent. As observed earlier data-oriented investigations build on the exploitation of shared heterogeneous resources, which bring scientists the responsibility to glue together these resources. Interviews with scientists have shown that this effort takes up to 50% to 80% of scientist's time [Loh14]. And as mentioned earlier the need to automate this integration is an important motivation for scientists for having workflows. Note however, that this particular role that workflows fill strongly determines how workflows are shaped. Hull has been first to characterise these gluing (adapter) activities in workflows (in bioinformatics) as "Shims" [HSL+04]. Adapter activities are required for "resource plumbing" [LAB+09] and "data grooming". From a high level perspective:

- *resource integration* corresponds to the effort required in accessing a resource; differences in access requirements need to be catered for here, such as messaging formats (e.g. JSON, XML, text), or access protocols (e.g. grid job submission protocols, or REST-based web services) [LAB+09].
- *data grooming* corresponds to all efforts that groom data prior to its use in a scientific analysis. Guo [Guo13] observes that this preparatory stage involves formatting, cleaning, partitioning datasets.

We therefore performed our analysis with anticipation to observe and to understand in more detail above kinds of activities in scientific workflows. More specifically our objectives were:

- Do analytical and adapter type data processing manifest as separate activities in workflows?
- What kinds of adapter steps are there and what is their extent in workflows?
- Is an enumeration of activity function possible?
- What is the current practice of encoding experimental context (factorial designs, configuration settings) into workflow design?
- What is the current practice of scientists in dealing with workflow complexity?

## 3.4 Analysis Setting

The analysis corpus is comprised of workflows from Taverna [MSRO+10], Wings [GRK+11], Galaxy [GNT10] and Vistrails [CFS+06] systems. The choice of these systems as our cohort is due to the availability of shared workflows through repositories [MSFS11] [DRGS08] and portals [gal13]. A workflow is shaped by several factors including 1) the design constructs available in the workflow language 2) the world-view of available resources, i.e. data and analytical tools composed by the workflow system 3) the scientific domain(s) where the system is used. In this chapter we are aiming to investigate the characteristics of workflows that are imposed by external factors rather than the innate characteristics that come with the supported workflow modelling language. Consequently we introduce the four systems in our cohort by discussing the world-view adopted and the domains in which they operate. A summarised view of the discussion is given in Table 3.1.

Table 3.1: Cohort Workflow Systems' Characteristics

| **Workflow System** | **Open Environment** | **Controlled Environment** |
|---|---|---|
| Taverna | Data & Activity | Activity |
| Wings | - | Data & Activity |
| Galaxy | - | Data & Activity |
| Vistrails | Data & Activity | Activity |

All cohort workflow systems were introduced in Section 2.6 while surveying the state of the art in workflow provenance. Here we recite their certain characteristics for text coherency.

**Taverna** [MSRO$^+$10] is a widely-used cross-domain workflow system applied in bioinformatics, astronomy, medicine, high-energy physics, chemistry, solar physics, ecology and digital library domains. The distinguishing feature of Taverna that also underlies its widespread adoption is its openness in both the data and the analysis activities that can be incorporated into workflows. On the activity side, any tool with a standard invocation interface such as a web service endpoint or a command-line tool can be integrated into a workflow. On the data side, Taverna's domain neutral typing (comprised only of basic MIME types) and very basic data-structuring (through collections-items) renders any kind of data potentially processable via workflows. Most typically scientists adopt this open approach to integrate (remote) third party resources with local ones to build analysis pipelines. An alternative to openness is to exert control over resources. With control, one could restrict the kinds of analysis tools and datasets that can be used. Taverna allows creation of controlled environments with a component framework. A component family defines a world view of all allowed activities and the types of data that can be consumed and produced by those activities. The advantage of having control is the simplification of workflow development for the scientist and the prevention of design errors.

**Wings** [GRK$^+$11] is a "semantic" workflow system that allows workflow development by combining activities (called components) and datasets selected from respective catalogues. Wings adopts a controlled world-view to both activities and datasets as it requires these resources to be made part of a catalogue prior to use. The defining feature of Wings is the ability to store semantic descriptions and constraints about resources during cataloging. Whenever those resources are composed in workflows, constraint checking is performed to ensure valid combinations. The Wings system has been used in Life Sciences, Text Analytics and Geosciences domains. A notable feature of Wings is its ability to delegate the execution of validated workflows to the Pegasus [DShS$^+$05] system. Pegasus specialises in the execution of workflows on grid environments or compute clusters.

**Galaxy** [GRH$^+$05] is a web-based data-analysis platform specifically designed for use in biomedical disciplines. Galaxy tightly controls resources, analysis activities called tools, and dataset by necessitating that they are made part of the Galaxy backend prior to use. Hence a common pattern in use of the Galaxy system is to have customised

Galaxy deployment for each sub-discipline it is used in. During the introduction of resources to catalogues, Galaxy prompts scientists to supply domain-specific resource metadata. Galaxy only supports basic dataflow definitions and lack support for any any further design construct. On the other hand, Galaxy backends can be deployed on cloud computing infrastructures where compute-intensive analyses can be effectively executed.

The **Vistrails** [CFS$^+$06] system supports the creation of workflows for scientific visualisation in Medical Informatics and Environmental Sciences and Physics domains. Vistrails workflows can be built by composing activities in the open, such as web services and also controlled such as modules from a rich embedded visualisation library. While both modes are supported the defacto approach is the controlled approach where activities are annotated to denote their accepted input/output types (e.g. geometric shapes, numeric types) and further domain specific characteristics. Similar to Wings annotations are later used for validity checking of module compositions.

### 3.4.1 Methodology

We performed a **bottom-up and primarily manual** analysis regarding both *activities* and the *data* consumed and produced by activities. As summarised in Table 3.2 the identification of activity motifs and the measurement of their occurrences in the workflow cohort was performed jointly by three analysts, namely Pinar Alper (dissertation author), Daniel Garijo and Khalid Belhajjame. The identification of data motifs has been exclusively performed by Pinar Alper. All three analysts are computer scientists who have familiarity with Hull's categorisations of Shim services in bioinformatics [HSL$^+$04] and basic dataflow constructs, such as relational query operators [CCT09]. None of these analysts have in-depth information or affinity to any particular scientific domain. P. Alper has analysed the Taverna cohort, K. Belhajjame has analysed the Vistrails cohort, and D. Garijo has analysed the Wings and Galaxy cohorts.

The activity analysis was done in two dimensions 1) Functional, where we outline **what kind of data-processing** occurs within a workflow activity and 2) Non-functional, where we identify **how activities or workflows are realised**. E.g. a visualisation step can be implemented in different ways: via a stateful multi-step invocation, through a single stateless invocation, or as a sub-workflow. We categorised each activity with at most one functional motif. This restriction has emerged from our observations on the cohort rather than being set upfront in our analysis method. We observed that it is rarely the case that an atomic activity had multiple functional motifs.

Table 3.2: Work performed by each analyst

| **Analysis Of** | **Workflow System** | **Performed By** |
|---|---|---|
| Data Motifs | All Systems | P. Alper |
| Activity Motifs & Motif Occurences | Taverna | P. Alper |
| | Galaxy | D. Garijo |
| | Wings | D. Garijo |
| | Vistrails | K. Belhajjame |
| | Cross-Validation w/ Selected Taverna Astronomy WFs | P. Alper & D. Garijo |

Analysis was based on manual inspection of workflow descriptions. In addition to workflow descriptions, additional documentation in the form of workflow annotations and textual descriptions and the underlying implementations of workflow activities such as scripts were also inspected as necessary to determine motifs. The only automated part of the data collection was associated to the workflow size statistics for Taverna workflows. The myExperiment repository that hosts workflow descriptions from several workflow systems provides a REST API that allows retrieving detailed information on Taverna workflows. Using this facility we were able to automate the collection of statistics regarding the number of workflow steps and the number of input/output parameters of Taverna workflows.

The analysis was bottom-up in the sense that rather than hypothesising a catalogue of expected motifs up front, analysts accumulated the set of observed motifs and their number of occurrences as they progressed with the analysis. In order to minimise misinterpretation and human error, categorisation made by P. Alper over a selected subset of workflows (43 Astronomy workflows in Taverna) has been cross-validated by D. Garijo. A cross validation of K. Belhajjame's categorisation with other analysts' has not been performed.

Workflow cohort has been obtained from the following sources at the time of the analysis (January 2013):

- Taverna workflows were obtained from myExperiment [DRGS08], which is the largest repository of scientific workflows. myExperiment contains workflows from a multitude of domains. We determined the target domains for workflows by browsing the myExperiment *group* tags [mye13]. These tags correspond to either generic concepts such as "workflow", "example" or "component" or to specific domains such as “text mining”. We eliminated generic tags and aggregated specific tags into the domains they signify. Taverna system has the largest public collection of workflows, in order to obtain a feasible subset for manual analysis, we made random selections.

- Galaxy workflows constituted a subset of those that were publicly available at the Galaxy deployment [gal13] at the time of the analysis. A selection was made to eliminate those lacking documentation.
- Vistrails workflow systems comes pre-packed with a set of tutorial type workflows that provide commonly used visualisation pipelines, additionally the CrowdLabs portal [MSFS11] allows Vistrails users share workflows. We have made a selection based on the availability of documentation for workflows.
- Wings system does not have a public repository of workflows. For Wings, we used a subset of workflows known by the Wings developer team at the time of analysis. The selection was made based on the availability of basic workflow documentation.

Table 3.3: Number of workflows analyzed from Taverna (T), Wings(W), Galaxy (G), Vistrails (V)

| Domain | No. of workflows | Source | | | |
|---|---|---|---|---|---|
| | | T | W | G | V |
| Drug Discovery | 7 | 0 | 7 | 0 | 0 |
| Astronomy | 51 | 51 | 0 | 0 | 0 |
| Biodiversity | 12 | 12 | 0 | 0 | 0 |
| Cheminformatics | 7 | 7 | 0 | 0 | 0 |
| Genomics | 90 | 38 | 28 | 23 | 1 |
| Geo-Informatics | 6 | 6 | 0 | 0 | 0 |
| Text Analysis | 45 | 11 | 31 | 3 | 0 |
| Social Network Analysis | 5 | 0 | 5 | 0 | 0 |
| Medical Informatics | 7 | 0 | 0 | 0 | 7 |
| Domain Independent | 30 | 0 | 18 | 0 | 12 |
| TOTAL | 260 | 125 | 89 | 26 | 20 |

### 3.4.2 Workflow Cohort

The domains of analysis and number of workflows from each domain are given in Table 3.3. We have chosen 260 workflows from various domains. This workflow set can be downloaded from the supplementary material pack we shared in myExperiment [Alp13]. Note that the distribution of workflows to domains is not even, as it is also the case in myExperiment for Taverna workflows. When selecting the workflows we included those developed with the intention of backing actual scientific investigations. We refrained from including toy or dummy workflows, which are developed to demonstrate a workflow system's design constructs. The workflows analysed are as follows (we use a convention "*X* out of *Y* workflows" where *Y* denotes the number of all workflows available from a particular system available at the time of analysis):

- 89 out of 132 Wings workflows were analysed. The domains constituted Drug discovery, Text mining, Generic (domain-independent), Genomics and Social Network Analysis.
- 125 out of 874 Taverna workflows (those compatible with Taverna version 2) were analysed. The domains constituted Cheminformatics, Genomics, Astronomy, Biodiversity, Geo-Informatics and Text Mining.
- 26 out 145 Galaxy workflows were analysed. Since Galaxy has wide application in the biomedical sciences, most of the workflows are from Genomics, although a few are designed for generic Text Mining capabilities.
- 20 out of 274 Vistrails workflows were analysed. Majority of workflows in Vistrails are generic pipelines that could be used in any scientific domain involving visualisation based investigation. Additionally we included workflows from Medical Informatics and Genomics domain.

## 3.5 Motifs

This section introduces motifs identified in our analysis. Motifs are divided in three categories: Activity Functional Motifs (given in Section 3.5.1), the Workflow Non-Functional (given in Section 3.5.2) and the Data Motifs (given in Section 3.5.3). An overview of all motifs is provided in Table 3.4. Throughout the introduction of motifs we will use two illustrative workflows:

- The Taverna workflow given in Figure 3.1 is from the Metabolomics domain [AHM10], and it analyses data obtained via Mass Spectrometry instruments that captures the unique chemical fingerprints that specific cellular processes leave behind. The workflow follows a process of data transfer to a remote server and preparation of input parameters needed for the analysis, running the analysis by invoking a remote stateful web-service, which requires multiple invocations for the completion of a single analysis. The final step in the workflow is the downloading of results from the server.

- The Wings workflow fragment given in Figure 3.2 is from the Drug Discovery domain [GKX$^{+}$11], and involves an analysis known as sMAP that can quantitatively characterize the geometric properties of proteins. In our example a comparison analysis is performed on two different input datasets. The results are then sorted and merged.

### 3.5.1 Activity Functional Motifs

**Data Retrieval** Certain activities are responsible for bringing-about the data into the workflow pipeline. Data from both local and remote sources can be obtained through relational queries, imports of tabular data (e.g. CVS, XLS files), or through web service invocations. Recall from our discussion of the data-oriented investigation lifecycle in Chapter 2 that some investigations re-use data from previous studies shared in community databases. Web services is the primary access mechanism for obtaining shared datasets. Parameters of such data-access services allow scientists to define a scope for the data of interest (a particular geographic location, a time frame or a particular subject). Data access could also be incremental and dynamically parameterised. For instance a first retrieval activity could result in a set of data items, which are then used as parameters to retrieve further data. Such linked retrieval activities form the so called "Data Integration" chains [MPL11].

**Data Analysis** This motif is a rough categorisation for those activities that perform some form of analysis over data. Analysis may be as basic as creating data statistics or as complicated as running data mining algorithms or simulations. A key differentiator for this category is that analysis activities mint data afresh, i.e. the results from these activities are previously not existent as activity inputs. The Taverna workflow in Figure 3.1 exemplifies this motif with its activity named *warp2D*. This activity accepts as input Mass Spectrometry data that contains indicators of the amount and type of chemicals found in a specimen. Using indicators warp2D computes various statistics and runs correction models to address shifts in data.

Table 3.4: Scientific Workflow Motifs

- Activity Functional Motifs
  - Data Preparation
    - Format Transformation
    - Augmentation
    - Extraction
    - Filter
    - Group
    - Sort
    - Split
    - Combine
      - Combine-Heterogeneous
      - Combine-Homogeneous
      - Merge
  - Data Analysis
  - Data Cleaning
  - Data Movement
  - Data Retrieval
  - Data Visualisation
- Workflow Non-Functional Motifs
  - Inter-Workflow Motifs
    - Atomic Workflow
    - Composite Workflow
    - Overloaded Workflow
  - Intra-Workflow Motifs
    - Internal Macro
    - Human Interaction
    - Statefull (Asynchronous) Invocation
- Data Motifs
  - Promoted
  - Intermediary
  - Configuration Parameter
  - Data Value
  - Data Reference
  - Design-Bound Value
  - Execution-Bound Value

**Data Cleaning** Datasets may require cleaning prior to their use. A cleaning step essentially preserves the primary content of data but improves data quality through removing noise data points, or fixing ill-formed data values. Cleaning can be automated

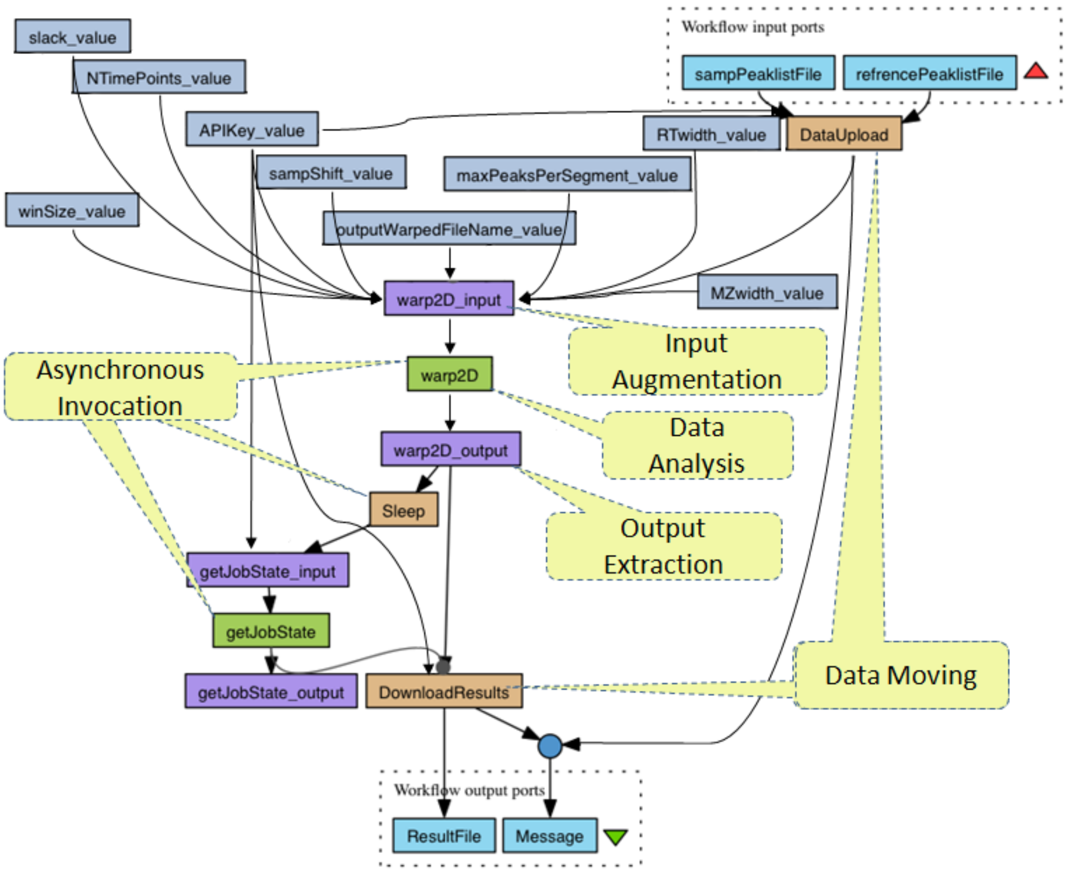


Figure 3.1: Motifs in a Taverna workflow from functional genomics.

(e.g. preprocessing a text corpus to remove punctuation) or it could be human-driven.

**Data Visualisation** In data-oriented science visualising a result dataset is often as important as obtaining/deriving that dataset. Visualisation has become a key ingredient of the analytical reasoning process that scientists employ [RFP09] as insights over data are often made during scientists inspecting and comparing visualisations. Moreover scientists use visualisations to understand the impact of experimental parameters and to re-iterate the workflow design process as necessary. Another use is in the communication of findings (in manuscripts or reports). Activities with the data visualisation motif typically create plots/charts from input datasets. Visualisation can be realised using both local and remote resources (e.g. R, Matlab libraries or web accessible resources such as Google charts).

**Data Movement** Certain analysis activities that are performed via external tools such as command line scripts or web services require the submission of data to a location accessible by the tool. In such cases the workflow contains dedicated step(s) for the upload/transfer of data to these locations. The same applies to the outputs, in which case a data download step is used to feed the data to the follow-on steps of the workflow. In Figure 3.1, the *DataUpload* and *DownloadResults* activities ship data to and from the server prior to and after the warp2D analysis.

#### 3.5.1.1 Data Preparation Motifs

**(Input) Augmentation** Data retrieval and analysis activities that are handled by external services or tools typically require as input well formed query strings or formatted requests such as web service input messages. Some activities in workflows aggregate multiple input parameters together with a format padding to create well-formed queries or service requests. In Figure 3.1 for each service invocation (e.g. *getJobState*) there are steps (e.g. *getJobState_Input*) that are responsible for creating the correctly formatted inputs for the service.

**(Output) Extraction** Raw outputs of data retrievals and analyses are typically in structure and form adopted by the external resources. Some activities in workflows are responsible for extraction of data from raw outputs by stripping of resource-specific padding and leaving data in a resource-neutral form. This motif can be considered the functional inverse of the augmentation motif. An example is given in Figure 3.1, where extraction activities (e.g. *getJobState_output*) are responsible for parsing the result XML message returned from the service (*getJobState*) to return a singleton value containing solely the job state.

**Format Transformation** Heterogeneity of data formats is a known issue in many scientific disciplines [Li11] [MMH$^+$15]. Workflows that bring together multiple access or analysis activities have an obligation to cater for format heterogeneity. This is achieved by format transformation. This motif could be observed as a combination of extraction and augmentation motifs where the padding around data for one format is stripped only to be replaced by padding required by another format.

**Combine** This category refers to activities that combine multiple threads of data during the execution of scientific workflows. Separate threads can originate from iterated and parameterised execution of activities or from execution of parallel branches inside the workflow. As motifs are a high-level characterisation, we cannot ascertain the structure of data nor the semantics adopted by the activity. This means that when an activity has the combine motif we can only assume that it relays the data at its input port(s) to its output port. We observe three sub-motifs in this category

- **Combine-Heterogeneous** This motif can be likened at a high level to relational *Join* operator. Recall that this operator (with the exception of self-join) is used to combine data from two tables typically representing distinct relations (differing information). Similarly the Combine-Heterogeneous motif is an abstraction to represent activities that combine data of different nature.

- **Combine-Homogeneous** Activities with this motif bring together data threads of similar nature. At a high-level this can be likened to the relational $Union$ operator. Appending one CSV dataset to the end of another is a common example for this motif.

- **Merge/Flatten** Recall from Chapter 2 that workflow systems like Taverna and Wings support the structuring of data in the form of nested collections. This motif refers to a specific form of data combination where an activity un-nests collections. A flattening activity would take as input a collection with nesting level and $n$ and would return a data with nesting level $n-1$, the un-nesting would coalesce all items at level $n$ in to a singleton data item.

**Filter** In scientific workflows the primary scoping of data, i.e. acquiring the data of interest, is typically done during Data Retrieval. Meanwhile datasets may require further narrowing especially in the cases of eliminating erroneous/invalid data. In this regard Filtering is a specialised data cleaning activity used by scientists. Examples are eliminating punctuation characters prior to text mining, or clearing empty lines from a CSV dataset.

**Group/Aggregate** Rare as they are (as we shall see in Section 3.6) certain activities in workflows organise data into groups so as to compute group-wise aggregates or to expand data items by calculating simple aggregate attributes out of existing ones.

**Sort** To cater for the expectations of data analysis or organisation activities input datasets may need to be sorted prior to use. Figure 3.2 shows an example where the data resulting from the $SMAPV2$ analysis are sorted with the $SMAPResultSoterV2$ component prior to their merging in a subsequent step.

**Split** Activities with the split motif perform the inverse of the *Merge* motif. Our analysis has shown that many steps in the workflows separate an input into different outputs. For example, the splitting of a dataset in different subsets to be processed in parallel in a workflow, or a dataset of nesting level $n$ may further be split into a dataset of nesting level $n+1$.

## 3.5.2 Workflow Non-Functional Motifs

We divide this category in two sub-groups, namely Inter- and Intra-Workflow motifs depending on whether motifs are observed by analysing the relations among multiple workflows, or within one workflow.

### 3.5.2.1 Inter Workflow Motifs

**Atomic Workflows** Our review observed that certain workflows perform an atomic, self-contained unit of functionality, which effectively requires no dependence on or derivation from another workflow. Typically these workflows are designed to be included in other composite workflows. Atomic workflows are the main mechanism of modularising functionality in scientific workflow development.

**Composite Workflows** Scientists re-use other workflows to create increasingly complex units of function, and eventually, whole-analysis pipelines. The Composite Workflow motif refers to those workflows that build-upon other workflows. There are two mechanisms of composition.

- Through the use of sub-workflowing construct provided in certain workflow systems (e.g. Taverna and Vistrails systems). This is an explicit way of creating composite workflows and results in nested workflow designs.

- By using adhoc methods of incremental development. In Galaxy and Wings systems the sub-workflow construct is not supported, therefore the only mechanism to build upon other workflows is to transfer these workflows into larger workflow designs via copying and pasting workflow scripts.

**Workflow Overloading** Our analysis has shown that authors tend to deliver variants of a workflow, each performing similar function, but operating over different input data formats. An example is performing a text mining analysis over an input of *PDF* or

$TXT$ file. Informally this motif can be likened to method overloading in programming languages, where multiple methods of the same name (denoting similar function) can be written and where each overloaded method differs from others by input parameter arity or supported input formats. Overloading is a way to deal with heterogeneity of data formats and it is adopted by developers, who aim for maximum uptake/reuse of their workflows.

#### 3.5.2.2 Intra Workflow Motifs

**Internal macros** This category refers to repeated patterns of activity combinations (fragments) within workflows. This is achieved in a way similar to adhoc composites. A workflow design with the internal macro motif is one that has copied-pasted fragments within. The Wings workflow in Figure 3.2 has this motif. There are two branches of the workflow that contain the same fragment ( $SMAPV2$ followed by $SMAPResultSorterV2$).

**Human interactions** Workflow systems increasingly involve human-interaction driven activities in addition to traditional entirely-computational activities. Human interactions are typically needed for data cleaning or for the dynamic/run-time selection of activity configuration parameters.

**Stateful/asynchronous invocations** One way to realise data-analysis activities is through creation of stateful analysis jobs. Such job-based analyses are typically long running and compute intensive and are often undertaken in grid environments, compute clusters or cloud environments. A stateful analysis require multi-step interactions to allow data submission, job initiation, job status inquiry, and result access. The Taverna workflow in Figure 3.1 has the stateful invocation motif, where the service invocation *warp2D* causes the creation of a warping job. The call then returns a JobID, which is used to inquire about the job status (*getJobStatus*), and to retrieve the results (*DownloadResults*). Note that the alternative to stateful invocations is stateless-invocations, in which a data analysis or retrieval function is performed in a single step.

To this end we have described the activity motifs which are descriptive of the nature of data processing inside otherwise opaque (black-box) workflow activities. We will now introduce Data Motifs; these are characterisations informative on either the innate nature of data or those characteristics of data only observable in the context of a workflow.

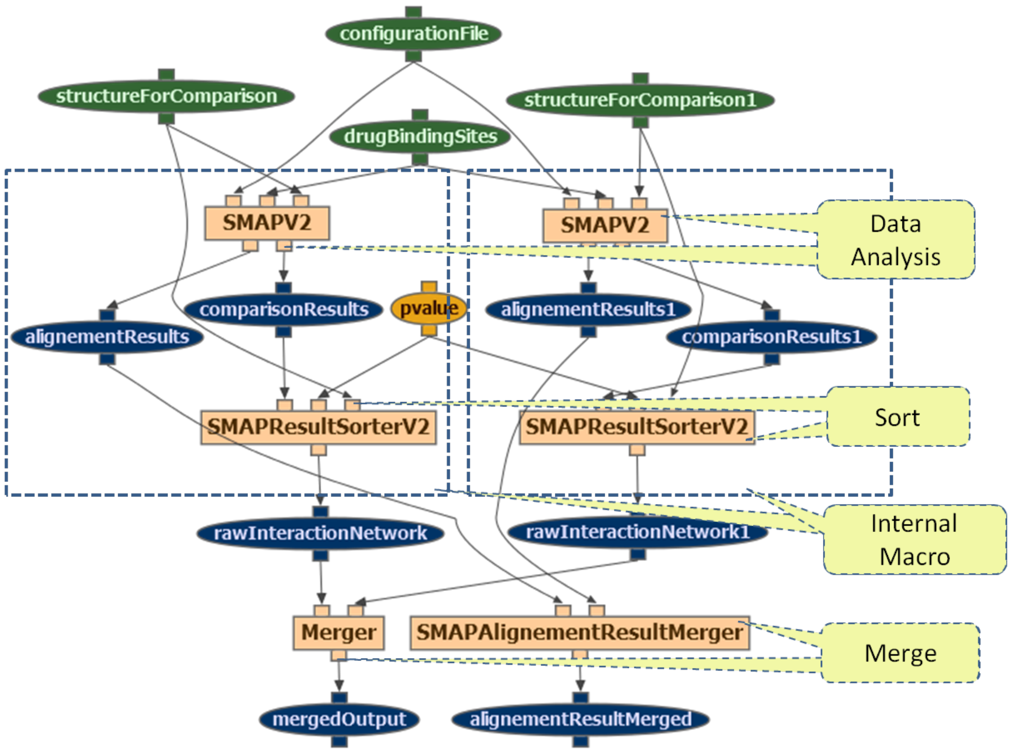


Figure 3.2: Motifs in a Wings workflow fragment for drug discovery.

### 3.5.3 Data Motifs

**Configuration Parameter vs Data** The data artefacts that are input to activities can be viewed in two major categories, 1) the actual data to be processed and 2) the configuration parameters. This separation is empirically evident in all systems' workflows. Wings and Galaxy and Vistrails allow this separation to be explicitly modelled as part of workflow design, whereas in Taverna and Vistrails implicit mechanisms are in place. In Vistrails users are given the option to supply default values for activity inputs. Our observation in Vistrails is that default values are most commonly used for inputs of configuration type. Taverna users are provided with a constant value emitting activity, which is commonly used for modelling configuration parameters; in the example of Figure 3.1, the inputs *samplePeakListFile* and *referencePeakList* are carriers of data whereas all other inputs such as *slack_value*, or *winSize_Value* are configuration parameters. In the Wings example in Figure 3.2 rather than encoding each configuration as an input to *SMAP*, the activity has been designed to accept the location for a file, input named *configurationFile*, which contains all the configurations for the *SMAP* analysis.

**Collection and Item** In our comparative review of workflow systems in Chapter 2 we identified that most of them, in our study cohort Wings, Kepler and Taverna, support the modelling of data as (nested) collections. The *Collection* and *Item* motifs conceptualises this feature of workflow languages. The purpose of including this feature as a motif is to be able capture the characteristics of data in a manner that is independent from any particular workflow language. Note that in addition to *Data*, *Configuration_Parameter*s can also be collections (as illustrated in Taverna and Vistrails systems in Chapter 2).

**Reference vs Value** Recall from the activity motifs, we described *DataMoving*, for those that move data between the workflow execution environment and external environments, such as remote servers, or the file system. When the data is processed in an external environment its referred to by reference inside the workflow. The input of the *DataUpload* activity in the example of in Figure 3.1 has the *Value* motif, whereas its output has the *Reference* motif.

**Intermediary** Within a workflow a dataflow link among two activities denotes that the output of one activity will be used as input of another. Data artefacts that fulfil the roles of both being generated by an activity and being consumed by another have the intermediary motif.

**Promoted** In addition to activities having inputs and outputs workflows themselves can have inputs and outputs. Workflow Inputs/Outputs are design constructs facilitating the communication of data in and out of a workflow execution context. A workflow typically contains several activities and those activities may have multiple outputs. Among all data generated by activities within a workflow, some can be promoted to become outputs of the overall workflow. In the Taverna workflow in Figure 3.1 we can see that the output of *DownloadResults* activity has been promoted to become a workflow output. Among the cohort systems Galaxy and Wings do not have a separate modelling constructs for workflow inputs/outputs, whereas Vistrails and Taverna does. This difference in modelling constructs appear to be closely related to the workflow system's ability to manage multiple nested execution contexts. Recall that Galaxy and Wings do not support sub-workflows, whereas Taverna and Vistrails does. The capability in the Galaxy system for promoting activity outputs is the ability for users to mark uninteresting outputs as hidden, even though during execution data gets generated at these ports they will be hidden from view when displaying results.

**Constant vs Dynamic** The values of inputs to activities may be bound to a constant value by-design or may take dynamic values per execution. When discussing the Configuration Parameter motif we mentioned *string constant* inputs in the Taverna system or default values of Vistrails, these are examples of the Constant motif for inputs. Note that even though the Constant and Configuration Parameter motif is most commonly observed together on inputs, they are not mutually dependent. Through the use of human interactions or by reading from files parameter values may be determined at workflow execution time.

In the following section we present the occurrence statistics for activity motifs in our study cohort.

## 3.6 Motif Occurrences

Figure 3.3 illustrates the distribution of Activity Functional motifs across domains. The figure shows **the predominance of the data preparation motif**, which accounts for 57% of all functional motifs in the entire dataset (labeled "all domains" ). Note that this measurement is, to our knowledge, the first such quantification of the extent of data preparation within workflows. In each domain this category is the most common, surpassing those that perform a scientifically-significant function such as analysis or visualisation. When Data Preparation is considered jointly with the Data Movement motif, these steps account for 63% of all functional motifs in scientific workflows. The Social Network Analysis workflows (from the Wings system) stand out as an exception, with no occurrence of data preparation. This is because these workflows make strict assumptions on the input data structure, the data model, and the protocols with which to access data so that the need for data grooming prior to analysis is eliminated. Another observation is that Data Retrieval operations are as common as visualisations and analysis. Within domains such as Genomics, Astronomy, Medical Informatics or Biodiversity, where curated common scientific databases exist, workflows are used as data retrieval clients against these databases.

Drilling down to Data Preparation, Figure 3.4 shows **the dominance of Augmentation and Extraction motifs** for most domains. These activities can be seen as adapters that help plug data analysis capabilities into workflows. Their occurrence is more visible in domains relying on third party services, most notably, in Taverna domains such as Biodiversity, Cheminformatics and Geo-Informatics, Genomics, and Astronomy. Format Transformation, the most widely mentioned example of data grooming

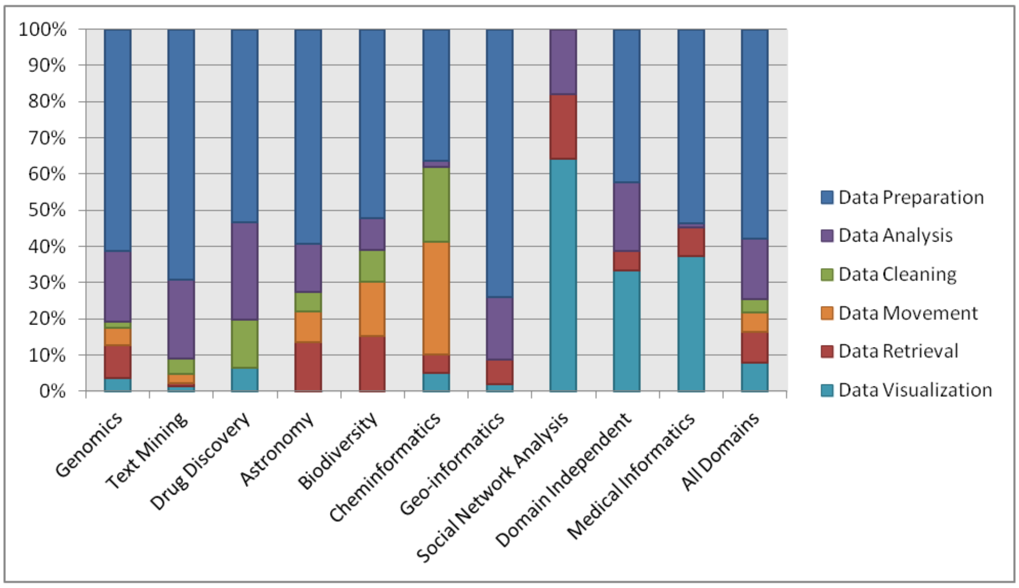


Figure 3.3: Distribution of Activity Functional Motifs per domain

accounts for (on average) 20% of such activities. Looking at the varying need for format transformation in Figure 3.4, we observe that in a domain where a widely used common data format exists, such as the VOTable [OWD+04] format in Astronomy, this reduces the need for such data grooming. Filtering appears as another motif that is as common as format transformation. While filtering is commonplace, other data organisation motifs such as sorting or grouping are much rare.

**Nature of workflow systems and their environment have their effect on motifs**. In order to analyse this effect in a manner undisturbed by differences of domains, we focus on a particular subset of the data belonging to a meta-domain, called the Life Sciences, which includes Genomics, Drug discovery, Biodiversity, Chemical Informatics and Medical Informatics. This meta-domain is common in all systems, and constitutes a significant share of workflows in each ($50+\%$). The number of workflows in each life science sub-domain is given in Table 3.5.

Recall from Section 3.4 that an important differentiating aspect of Galaxy and Wings systems is that they adopt the controlled-data approach, where they expect data to be made part of the workflow environment prior to processing. As a result in Figure 3.5 we observe that Wings and Galaxy workflows contain minimal-to-none data retrieval or movement steps, as data is pre-integrated into the workflow environment. On the other hand in the Taverna and Vistrails systems workflows carry dedicated steps for retrieving data from external sources ($10+\%$ in Taverna and 10% in Vistrails). The Data Moving motif is also visible in Taverna workflows but absent in Vistrails, as

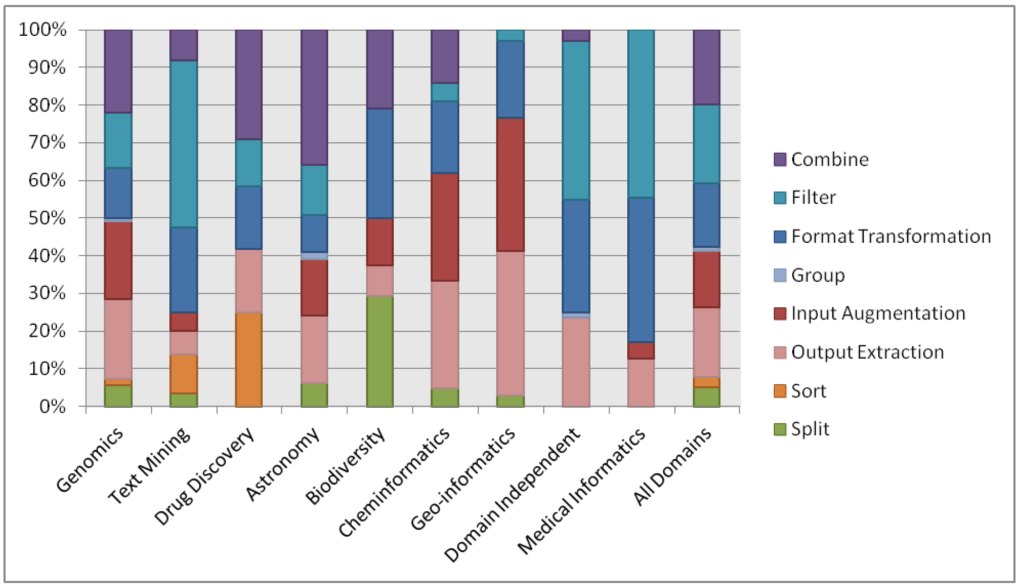


Figure 3.4: Distribution of Data Preparation motifs per domain. The Social Network Analysis domain is not included, as it doesn't have any data preparation motifs.

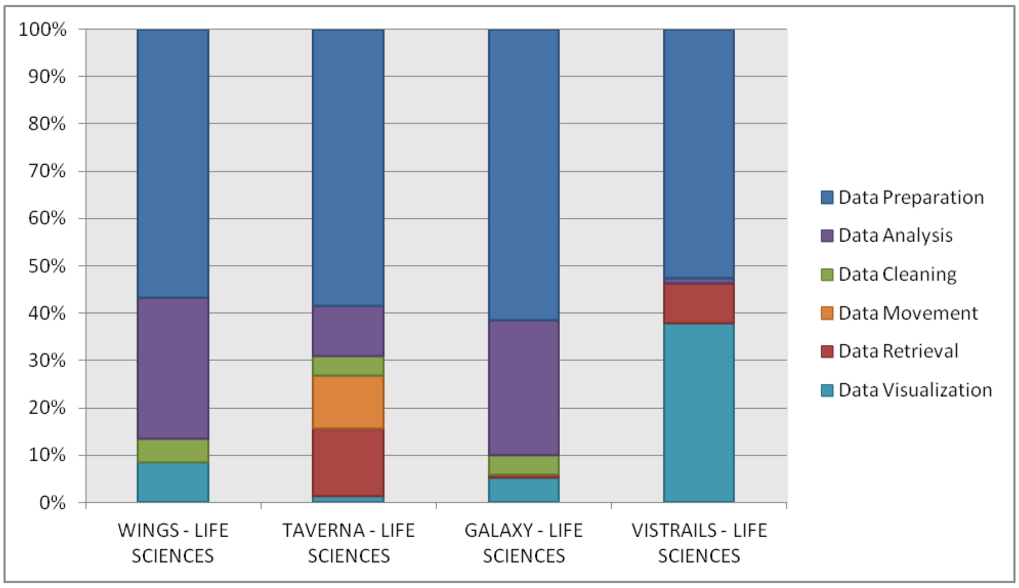


Figure 3.5: Activity Functional Motifs in the Life Sciences Workflows

Taverna is open in terms of analysis activities involved and has to ship data to locations accessible by third-party analysis tools. Vistrails adopts a library of controlled activities, so once data is retrieved for analysis, further moving is not necessary.

Another observation we make from Figure 3.5 is on data visualisation. Finding the right visualisation for results is often a challenge in itself. The Vistrails system allows

Table 3.5: Distribution of workflows from Taverna (T), Wings(W), Galaxy (G) and Vistrails (V) in the Life Sciences domain

| Domain | No. of workflows | Source |
|---|---|---|
| Drug Discovery | 7 | W |
| Biodiversity | 12 | T |
| Cheminformatics | 7 | T |
| Genomics | 90 | T (38), W(28), G (23), V (1) |
| Medical Informatics | 7 | V |
| TOTAL | 123 | |

scientists to captures this exploratory process ( in search of a suitable/acceptable visualisation) with workflows, consequently this motif has a large occurrence in Vistrails workflows (almost 40%). The Galaxy toolkit, on the other hand, utilises an interactive visualisation tool [GCT+12], separate from the workflow environment. Scientists use this tool to find the correct visual presentation for their data. Once this visualisation is found, it can be incorporated as a visualisation activity into the analysis pipeline, around 5% activities in Galaxy workflows have this motif. Taverna and Wings systems do not have dedicated mechanisms for the exploratory or interactive search for a right visualisation, yet they allow scientists to apply a pre-determined visualisation operations to their findings within workflows, the percentage in these systems is 2% and 10% respectively.

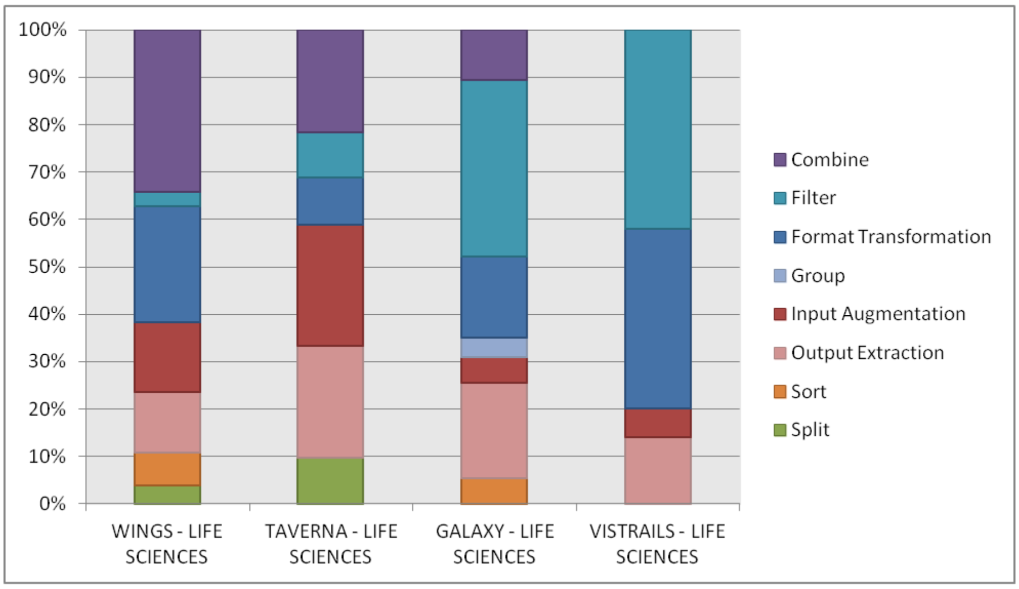


Figure 3.6: Data Preparation Motifs in the Life Sciences Workflows

The percentage distribution of Data Preparation motifs for the Life Science domain is given in Figure 3.6. We observe that the distribution for this domain is similarly affected by difference of workflow environments (open vs controlled). Augmentation and Extraction motifs account for the largest share of data grooming activities in Taverna (almost 50%), and less so in Wings (30%), Galaxy (20%) and Vistrails (20%).

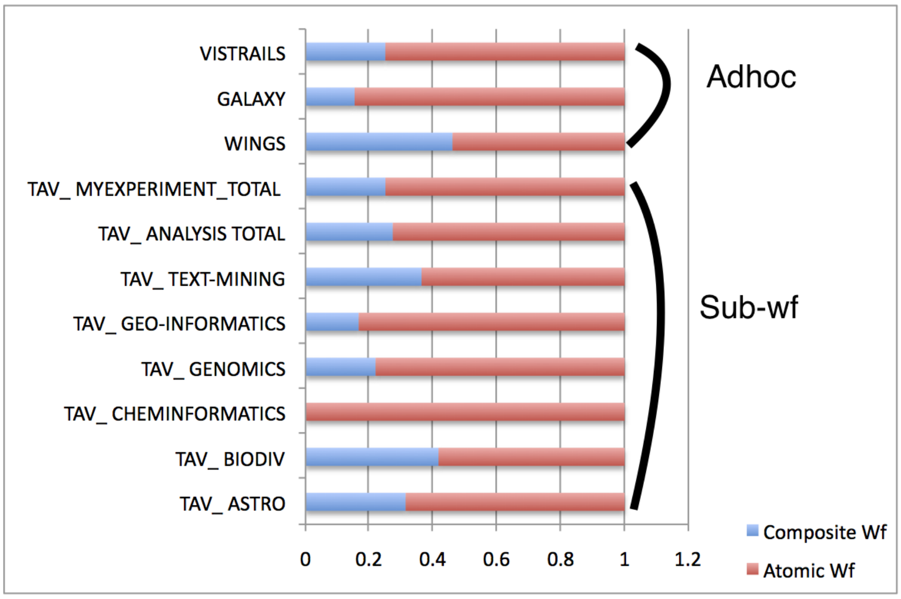


Figure 3.7: Percentage of Atomic versus Composite Workflows

We now move on to the analysis of workflow non-functional motifs. We provide numbers on a frequency/possibility per workflow basis rather than percentage distribution among motifs. The reasons for this presentation are: the non-functional motif category is not particularly tied to activities but mostly observable over workflows.

We begin with Composite Workflows, which are those that build on other workflows either with the sub-workflow construct or by adhoc means such as copy-pasting fragments. The Taverna dataset exhibits the sub-workflow approach whereas Wings, Vistrails and Galaxy datasets exhibit the adhoc approach. Figure 3.7 provides the percentage of atomic versus composite workflows in each system. Our manual analysis over 128 Taverna workflows showed that 73% of workflows are atomic (flat), i.e. contains no sub-workflows, whereas the remainder 23% are composite (nested). Because Taverna workflows exhibit explicit sub-workflowing, an occurrence analysis over the entire dataset is possible using the myExperiment query interface. Our inquiry via this interface over the entire Taverna-2 workflow set revealed 25% composite and 75% atomic workflows (given in Figure 3.8), which showed that, for this motif, our sample set is representative of the whole set. Figure 3.8 also shows us that the majority among nested workflows are only 1 and 2 levels deep and it is very rare for workflow designs to have nesting levels greater than 2.

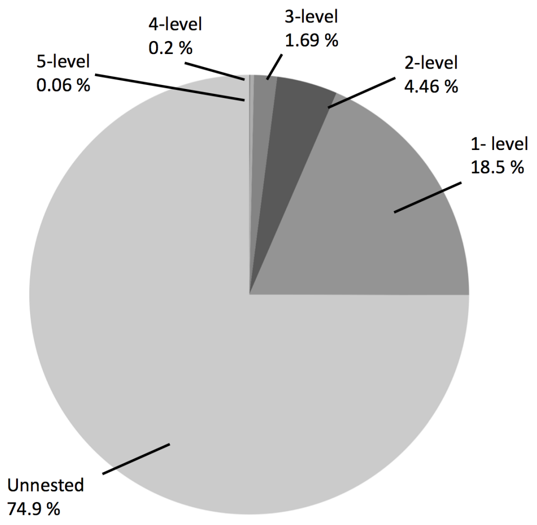


Figure 3.8: Nesting Levels of Taverna Workflows in myExperiment repository.

For the other systems, that adopt the adhoc composition approach, manual analysis over the cohort shows 46% to 54% composite to atomic distribution as for Wings, 15% to 85% for Galaxy and 25% to 75% for Vistrails. Among all the Taverna workflows in our manually analysed sample 58% are modular components designed to be included in other workflows (note that it is not possible to automate this analysis over the entire Taverna set due to a lack of unique identification and referencing of workflows in the Taverna system). This result combined occurrence of Composite Workflow motif shows that modular design of workflows and building workflows by re-using parts of other workflows for development [GFG$^+$09] have become an empirically observable practice in scientific workflows.

The sub-workflow construct, i.e. the ability to create nested workflows, is a critical factor in managing workflow size. **The difference in resource environments affects sizes of workflows in respective environments.** Systems that adopt an open resource approach have larger workflows that are crowded with adapter steps needed. This affect is presented in Table 3.6. Here we review the size of workflows in our study cohort in terms of number of minimum, maximum and average activities per workflow. (In Table 3.6, for Taverna, the size depicts the number of activities at the top-most layer of design (where workflows are nested). From Table 3.6 we observe that the total average size for each system's workflows (bottom row) is close to the human working-memory limit of 7(+/-2) [Mil56]. In the first outlook Taverna systems workflows, even though it adopts an open resource environment, seems close in size to workflows from controlled systems like Wings and Galaxy. On the other hand the high upper bound on the size of Taverna workflows shows us how complex workflows can get. For the Taverna cohort the simpler workflows are intended as components to be included in larger workflows.

The difference of controlled versus open environment becomes more stark when we expand nested Taverna workflows and count the total number of activities within. This is presented in Table 3.7. This table also shows us how very complex workflows in the domains of Astronomy and Biodiversity have been scaled down to levels of workflows from controlled-resource systems through the use of sub-workflowing constructs (recall the Taverna domains that displayed the composite workflow motif in Figure 3.7)

Among the Taverna domains that use sub-workflowing, we can clearly observe a negative correlation between the rate of occurrence of the composite workflow motif and (top-level) workflow size. In Figure3.7 we can see that Biodiversity and Text Mining domains have composite workflow motif more frequently than any other Taverna domain. Accordingly in Table 3.6, we see that these two domains have the most compact (top-level) workflow designs in Taverna. The ability to create layered workflow designs is also dependent on the capabilities of workflow designers. Due to our close-connections with Taverna user base, we are aware that workflows in Biodiversity or Text mining domains are developed by the help of computation-savvy expert workflow designers rather than pure domain scientists. Meanwhile in Genomics workflows are often developed by domain-scientists, and therefore workflow complexity remains unmanaged (intact).

The Stateful Invocation motif is observed only in the Taverna system, given in Figure 3.9 and absent in others. This motif is representative of single analyses that are performed by executing multiple consecutive activities. Systems with a controlled approach like Wings and Galaxy each have specialised activity scheduling/queuing frameworks [DShS$^+$05] [Sta06] that perform job creation, monitoring and result access under the hood, hence such details are absent from workflow design. In Taverna the activity execution at disparate providers may differ and the workflow design has to cater for these mechanisms. Note, however that stateful interactions is a rare motif in Taverna workflows, only in Genomics, Cheminformatics, Biodiversity and Astronomy this motif is observed with rate 0.13 occurrence per workflow .

Our analysis revealed that Human-Interactions are increasingly used in workflows,

Table 3.6: Minimum, Maximum and Average number of activities per workflow, in different domains and workflow systems.

| Domain | Source | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Taverna | | | Wings | | | Galaxy | | | Vistrails | | |
| | Mn | Av | Mx | Mn | Av | Mx | Mn | Av | Mx | Mn | Av | Mx |
| Drug Discovery | | | | 11 | 8 | 18 | | | | | | |
| Astronomy | 1 | 7 | 33 | | | | | | | | | |
| Biodiversity | 1 | 4 | 12 | | | | | | | | | |
| Cheminformatics | 1 | 8 | 20 | | | | | | | | | |
| Genomics | 1 | 11 | 53 | 1 | 5 | 17 | 1 | 7 | 29 | 7 | 7 | 7 |
| Geoinformatics | 3 | 8 | 14 | | | | | | | | | |
| Text Analysis | 1 | 4 | 6 | 1 | 4 | 15 | 4 | 6 | 11 | | | |
| Social Network Analysis | | | | 3 | 6 | 7 | | | | | | |
| Medical Informatics | | | | | | | | | | 8 | 14 | 29 |
| Domain Independent | | | | 1 | 2 | 5 | | | | 4 | 11 | 20 |
| TOTAL | 1 | **8** | 53 | 1 | **5** | 18 | 1 | **7** | 29 | 4 | **12** | 29 |

Table 3.7: Minimum, Average and Maximum number of activities in expanded composite workflows in Taverna (T)

| Domain | Number of Activities | | |
|---|---|---|---|
| | Mn | Av | Mx |
| Astronomy | 1 | 11.1 | 65 |
| Biodiversity | 1 | 12.16 | 78 |
| Cheminformatics | 1 | 9 | 23 |
| Genomics | 1 | 13.8 | 53 |
| Geoinformatics | 3 | 8.5 | 14 |
| Text Analysis | 1 | 6.6 | 19 |
| TOTAL | 1 | 11.45 | 78 |

most typically observed over data cleaning activities or workflow configurator activities for the selection of run parameters. Biodiversity makes extensive use of human-interactions (see Figure 3.10), this motif is observed almost once (0.83) per workflow. The interaction interface may be as simple as a multi-choice list generated by an activity script or it may be as elaborate as the spreadsheet editing interface of a data refinement tool [vHVDW13]. This motif is absent in Wings and Vistrails workflows. Galaxy is an environment that is heavily based on user-driven configuration and invocation of analysis tools (some parameters and inputs of the workflows can even be overridden even after the execution of the workflow has started). However, based on our definition of Human Interactions, i.e. analytical data processing or decision undertaken by a human, the Galaxy workflows also lack this motif.

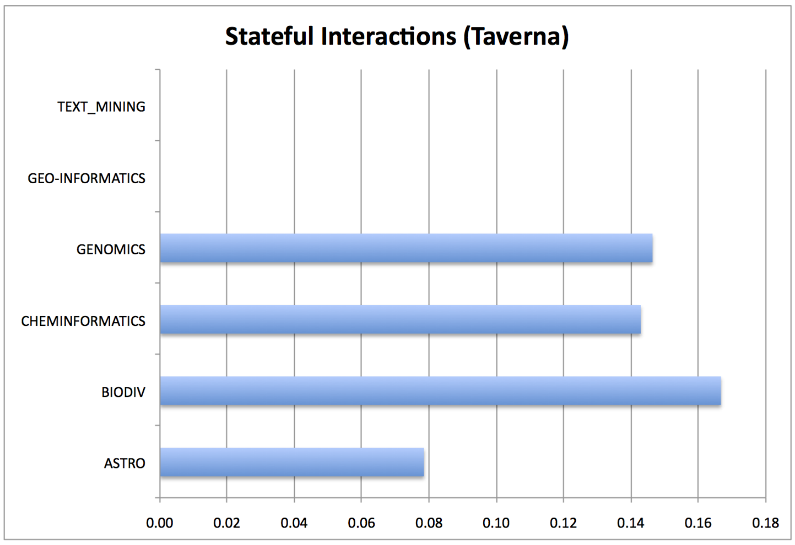


Figure 3.9: Occurrence of Stateful Invocation Pattern

The Workflow Overload motif is observed in all Wings domains and the Biodiversity and Text Mining workflows of Taverna. Similar to the layered workflow designs observed in these domains the ability to create overloaded workflow designs is also dependent on the computation-savvyness of workflow designers. An alternative approach to overloading would be to have single workflows with polymorphic parameter(s). We did not observe this approach in our analysis dataset, yet we observed very few cases [McW08] from Taverna system's earliest (now obsolete) releases. In the polymorphism case the workflow is generic and within it contains multiple distinct implementations that handle each differing input type. While we observe overloading as a good practice, use of polymorphic parameters is a poor practice as it complicates workflow designs.

## 3.7 Motif Ontology and Annotations

The first step in operationalising the information coming out of our empirical analysis is to capture it in a formal representation. Ontologies are a good match here as they allow the definition of "formalised vocabularies of terms" [W3C12]. We use the W3C Standard Web Ontology Language (OWL) [HKP+09] to capture motif categorisations in the Workflow Motif Ontology [GAB13]. Structurally OWL provides three major categories of definitions:

- It allows the definition of *Classes*, *Individuals* that belong to classes, and *Properties*

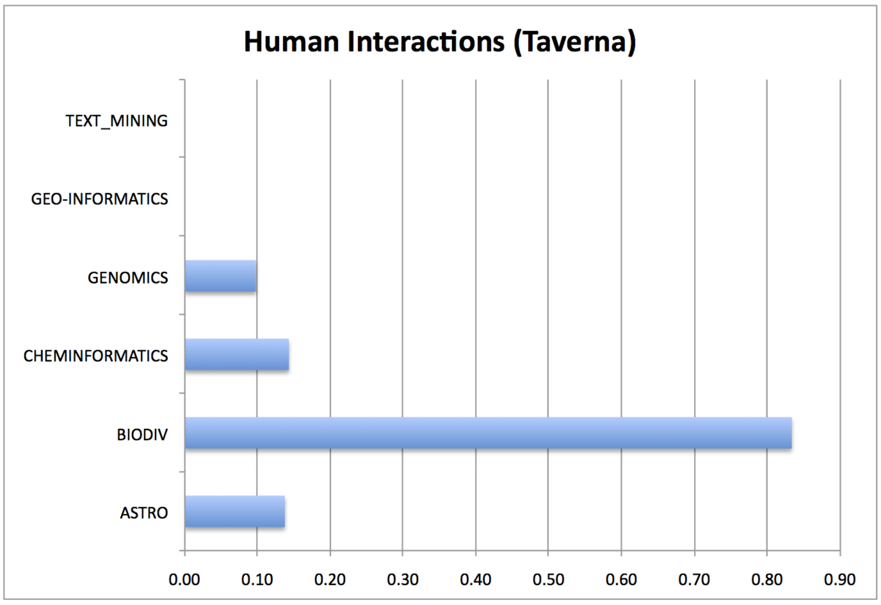


Figure 3.10: Occurrence of Human Interaction Pattern

that individuals can have. Properties either connect individuals together (*Object Properties*) or they connect an individuals to a literal values (*Datatype Properties*).

- As a knowledge representation mechanism that has its roots in Description Logics [KSH12], OWL allows classes to be described rather than simply asserted. A class description, also known as a *Class Expression* outlines the necessary features an individual has to have in order for it to belong to a class. Class expressions are built incrementally from simpler ones through use of primitives such as conjunction, disjunction and existential and universal quantification.

- It allows the specification of what is accepted as *true* in a domain through *Axiom* definitions. Axioms can be about other structural elements including classes and properties. Most typical examples are the *Sub Class* or *Class Equivalence* axioms for denoting relations among classes.

In addition to structural definitions, OWL comes with semantics that allow for reasoning over those definitions. Reasoning can provide new information, such as the inferred hierarchy of described classes. Or it could also be used for consistency checking by inferring whether a described class is satisfiable, in other words whether it is possible for this class to have individuals or not.

From these panoply of features that come with OWL we use a restricted subset in our motif ontology. More specifically, we define asserted classes for each motif, and

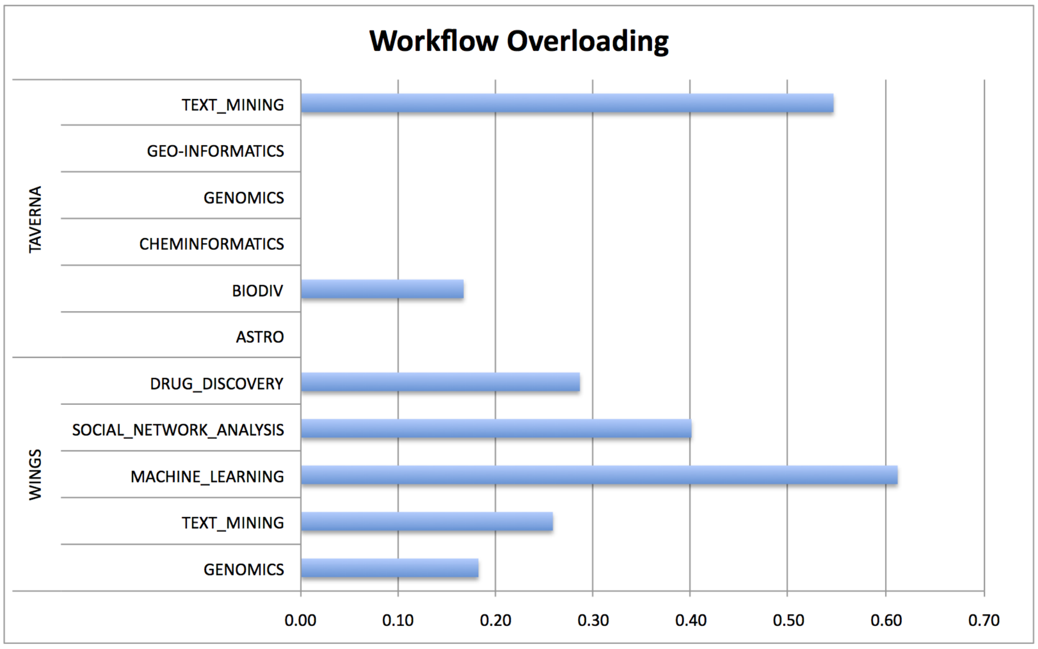


Figure 3.11: Occurrence of Workflow Overload Pattern

we use sub-class axioms to define a taxonomy of these classes as per Table 3.4. In addition to class definitions we define three object properties

- *hasDataOperationMotif* to associate activities with their functional motifs
- *hasWorkflowMotif* to associate workflows/activities with their non-functional motifs
- *hasDataMotif* to associate ports with their data motifs

We also define a more general property *hasMotif* and make it a super property for those above using sub-property axioms. A common practice when defining properties is to define domain and range restrictions. A domain restriction allows us to pinpoint which classes of individuals can be the source (holder) of the property in question. Recall from our review in Chapter 2 that there are several vocabularies for representing scientific workflows in an abstract manner as with the OPMW [GG11], P-Plan [GG12], D-PROV [MDB$^+$13] and Wfdesc [BCG$^+$12] models. In order for our motif ontology to be used in conjunction with several alternative workflow representations, both abstract and concrete, we refrained from putting domain restrictions on our properties.

Annotation is our chosen mechanism to convert activity black-boxes into grey-boxes. We use the motif ontology to annotate workflows and constituent activities and ports. The choice for annotation as a mechanism to bring transparency into activities has certain advantages. First of all it is *non-intrusive*, which allows our dissertation research, i.e. the development of experimental metadata provisioning techniques, to be at peace with real-life practices and to be applicable to existing workflows. Non-intrusiveness, combined with minimal restrictions on class and property definitions in the motif ontology allows annotations to be made *over workflows from different systems*. Secondly, annotating activities with their domain specific function and annotating activity input/outputs with their the domain-specific types of is already a *widely-adopted approach* for the solution of problems in workflow research. The most common use of annotations is for discovery of analysis activities, as in Taverna [LAWG05], and, data as well as activities, as in Wings [GRD+06] and Galaxy [GRH+05]. Another common use is the prevention [GRD+06] [WKM+10] or fixing of invalid activity compositions [CFS+06].

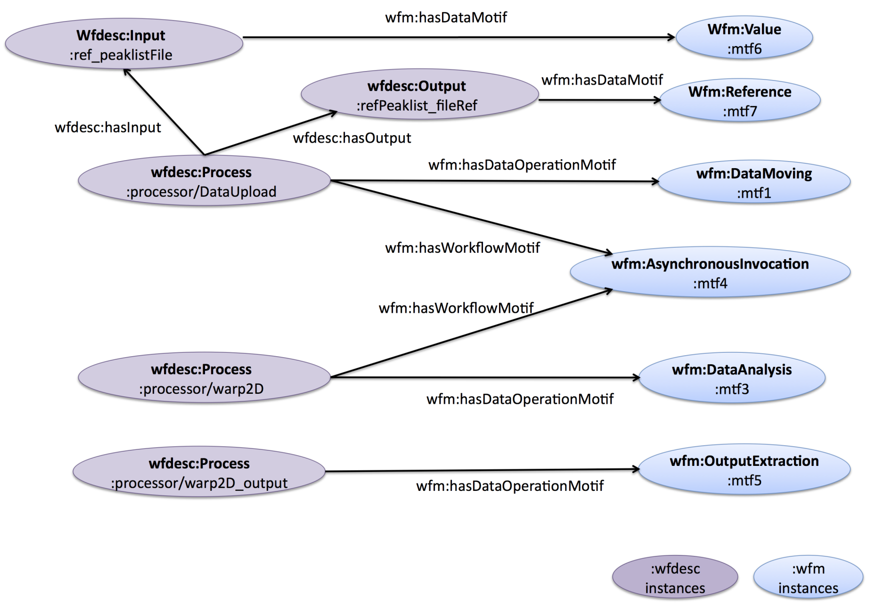


Figure 3.12: Subset of the annotations of the Taverna workflow shown in Figure 3.1 using the Wfdesc model.

A sample annotation of the earlier Taverna Metabolomics workflow is given in Figure 3.12. While providing an abstract and consistent representation of the workflow is not a pre-requisite to the usage of the Motif Ontology, we consider it a best-practice to use a model that is independent from any specific workflow language or technology. In this example the Taverna workflow is represented in the Wfdesc model describing its constituent elements (activities and ports on the left-hand side of figure). The object properties defined in the Motif ontology are used to link-up workflow elements with their associated Motifs (on right-hand side of figure).

## 3.8 Revisiting Anticipations and Goals

We now revisit analysis goals and expectations in light of the our findings.

**By manual analysis of workflow descriptions we were able to arrive at a domain-independent categorisation of activity functions, and their implementations in scientific workflows, called Motifs.** Motif occurrences show that a certain minority group of activities perform the scientific heavy lifting in workflows. These steps **mint** data through *Data Analysis* or *Data Visualisations* or *Data Retrieval*. The remainder activities can be broadly categorised as *Data Preparation* steps, which are data adapters that strive to format or organise the data in a way consumable by the scientifically-significant steps. Adapters do not mint scientific data afresh, instead they are **relayers**.

**The analysis showed that workflows are rife with *Data Preparation* steps, which account up to 70% of all activities.** This substantiated our observation from Chapter 2 that workflows are artefacts of implementation/automation as they systematically make explicit the effort needed to deal with resource heterogeneity. Another finding is: while a workflow system's controlled approach to resources and datasets may reduce the need for certain types of adapters (*Format Transformation*, *Data Movement* steps) it does not eliminate the need for adaptation entirely. We observed that Wings and Galaxy and Vistrails workflow cohorts, which are both from controlled-resource workflow systems included *Data Preparation* motifs nearly as much as other systems.

**Experimental context is captured in workflows, albeit context is spread-out and sometimes indirect and dynamically determined.** Recall from Chapter 2 that part of what is considered experimental metadata is domain-specific attributes of datasets, describing the context in which data is obtained. These attributes are expected to describe *data origin*, such as the external repository, or the local tool it comes from; data subject, such as a particular species or a stellar object, that the data is about; or values of experimental parameters used for data's derivation. It is apparent that the scientifically significant activities in the workflow, i.e. *Data Retrieval*, *Data Analysis* and

*Data Visualisation* activities and their inputs of *Configuration Parameter* motif are the hotspots for collecting such experimental metadata. Meanwhile:

- Our observation showed that while parameters are modelled individually in a workflow description these distinct parameters are not immediately co-located with the analytical activities that make use of those parameters. Distinct parameters are often bundled-up, wrapped with resource-specific protocol padding in *Augmentation* steps before reaching their point of use in an analytical step.
- Contextual information may be indirectly captured when activity inputs and outputs are *References* instead of *Values*. Consider for instance the case where an analytical task is configured with parameters accessed from a file, here the workflow description and its execution provenance will not be able to record the parameters that contributed to data, but will instead record the name of parameter file.
- With the *Dynamic* data motif we observed that parameters of analytical activities may not always be encoded as constant values in workflow design and they get determined at run-time. A typical example of this is the Data Integration chains comprised of multiple *Data Retrieval* activities, where the output of one becomes a parameter for the follow on activity.

**We observed that scientists use layered designs to manage workflow complexity.** The high upper bound on the size of workflows (recall Taverna workflows in Table 3.6) shows that analyses can be large and complex, due to the presence of *Data Preparation* and *Data Movement*. *Stateful Invocations* is another factor contributing to complexity where a single analysis is done through the invocation of multiple calls to external services. These steps dealing with the technical detail of resource access or data organisation obfuscate the scientifically significant/critical operations of the workflow. Obfuscation degrades reportability of the scientific intent embedded in the workflow. In our analysis we observed that a common way to overcome obfuscation, adopted by scientists, is to group related steps within sub-workflows to build up higher-order *Composite Workflows*. Another practice towards abstraction is to use the *Promoted* motif where selected intermediary results are made workflow outputs.

### 3.8.1 Using Motifs for Abstraction

We argue that motifs can be utilised in abstraction in the following ways:

- Motifs reveal the prime cause of complexity in workflows as the adapter steps, thereby signalling that these activities should be the target of abstraction.

- Motif categories reveal that adapter or significant steps are not entirely arbitrary in terms of their function. Our motif categorisation provides one high-level enumeration of functions of activities. This provides potential for developing abstraction mechanisms that operate at the level of characterisations rather than individual workflows and individual steps within those workflows.

- Motifs have surfaced from the current user practices on abstraction (sub-workflows or promoting intermediaries). These practices have two potential uses 1) we can develop abstraction mechanisms informed by these practices 2) the set of abstractions created by scientists in existing can be used as a benchmark on what a user desired abstractions should be.

### 3.8.2 Using Motifs for Grey-Box Provenance

To this end we have identified Motifs without having a critical discussion of the transparency they would bring to lineage. We will now discuss this by 1) identifying what information the motifs lack that would have been needed for white-box [CCT09] transparency, 2) identifying the grey-box transparency that certain motifs bring by comparing Motifs to Hull's categorisation of "Shim" activities [HSL$^+$04] and 3) highlighting a unique characteristic of workflow-based scientific data processing that makes a grey-box approach plausible.

Motifs provide only a partial understanding of the inner workings of activities. The following information is notably excluded from Motifs:

- The Activity Functional Motifs provided herein is a task-oriented characterisation that tells us what gets done in an activity. The core Motif ontology does not explicitly describe what the activity function implies in terms of relationships between the activity's inputs and outputs. However later in this Section we discuss the kinds of relations certain Motifs imply. In Chapter 7 we exploit these relations in annotation propagation.

- With the *Collection* Data Motif we capture the high-level organisation of scientific datasets. Beyond collections Motifs do not put any restrictions on the structure and format of data We intentionally left such a characterisation out of scope of our analysis as our observations revealed a innumerable amount of formats (CSV, XML, unstructured text, binary, various text-based domain specific formats).

- The Activity Functional Motifs also do not specify the detail on the inner workings of activities. Especially for the case of the data preparation motifs, where our classification is more specific (deep), we do not enumerate all possible procedures to *Filter* or *Combine* data. This is due to the diversity of data formats and the endless variation in procedures in such data preparation activities (e.g. coalescing two text fragments, line-wise join of two CSV files, value based filtering of an XML dataset using XPATH).

Due to the above omissions Motifs do not provide the transparency for a white-box approach, which we earlier characterised (in Section 2.4.3) as one in which 1) assumptions can be made on the structure of data and 2) each activity (e.g. matrix operations, relational select/join) has a fixed and well-defined semantics that wold allow accurate and qualified lineage relations among activity input(s) and output(s) (in the case of relational queries the qualification is *value-copying* from inputs to outputs). Recall from Chapter 2 that the accuracy and lineage qualification that white-box database provenance brings can be exploited in building provenance-enabled capabilities, such as propagation of annotations from source to result records

Despite the omissions, motifs still has the power to identify an anticipated lineage between the inputs and outputs of certain activities. Hull et al [HSL+04] has been the first to study common activities in workflows and the input-output relations inferred from activity functions. In this work authors have identified the existence of mediator type activities (they call "shims") used when gluing together analytical activities in bioinformatics workflows. Table 3.8 provides Shim categories, Shim-implied relations as identified by Hull et al, and our corresponding Motifs. Note that some shims map to *Data_Preparation* motifs, more notably some map to *DataAnalysis* or *Data_Retrieval*, which are activities we categorise as non-adapter type. Hull et al have identified relations implied by some of the shims as follows (the authors have represented relations in a custom ontology no longer available, here we cite more recent literature that caters for these relations):

- *semantic equivalence* of biological identifiers or differently formed but equivalent data [SL08],

- *part-whole* relations among data outputs built up by copying extracted part of inputs [AFGP96] ,
- *identifier-to-data* relations among biological identifiers and corresponding data [MBL+14].

Rather than representing these relations as qualified lineage, the main focus of Hull et al has been on categorising shim input/output types and using this information for discovering suitable adapter activities during workflow design, where activities with mismatched input/output types need to be linked-up.

Table 3.8: Shims and Corresponding Motifs

| **Shim** | **Example** | **Shim Implied Relations** | **Motif** |
|---|---|---|---|
| Dereferencer | GenBank ID replaced with GenBank record | I *uniquelyIdentifies* O | Data Movement, Data Retrieval |
| Syntax translator | SeqRet translates between representations of sequence data. | I and O are representations of the same thing | Format Transformation |
| Semantic translator | Translate DNA into protein | – | Data Analysis |
| Mapper | Maps between IDs. E.g. GenBank to EMBL | I *isEquivalentTo* O | Data Retrieval |
| Parser | Parse BLAST report. | – | Format Transformation |
| Iterator | Iterate over members of a given set | – | Not Motif, WF Language Construct |
| Comparer | Comparing BLAST reports notifies of new sequences | – | Data Analysis |
| Accessor/Extractor | Access a subset | I *hasPart* O | Extraction |

When we perform a similar exercise to determine implied relations for our data preparation motifs we obtain the result presented in Table 3.9. The most prominent relation is the part-whole relation [AFGP96]. We also have a reference-value relation for the *Data_Moving* motif. For the *Group* or *Sort* motifs, while we can assume that there is some value-copying from the inputs of the activity to the outputs, we cannot confidently identify the resulting relation as a part-whole relation.

Motifs provide transparency into workflow provenance 1) by differentiating between scientifically significant activities and adapter steps and 2) by providing qualified lineage for adapter steps. We argue that this partial (grey-box) transparency presents a

Table 3.9: Data Preparation Motifs and Implied Relations

| **Motif** | **Motif Implied Relations** |
|---|---|
| Format Transformation | I *hasPart* interim *isPartOf* O |
| Augmentation | I *isPartOf* O |
| Extraction | I *hasPart* O |
| Combine | I *isPartOf* O |
| Filter | I *hasPart*O |
| Group | – |
| Sort | – |
| Split | I *hasPart* O |
| Data Movement | I *isReferenceFor* O |

case worth exploring whether it can be exploited in building a provenance-enabled annotation generation and propagation capability (described in Chapter 7). More specifically:

- Scientifically significant activities with the *Data_Analysis Data_Retrieval* and *Data_Visualisation* motifs, and their *Configuration Parameter* inputs signal a source of information to be used to provision domain-specific metadata.

- Adapter activities with the *Data_Preparation* and *Data_Moving* motif signal a partonomy-type lineage relation among derivative datasets linked-up in a workflow execution trace. This information can then be used to propagate domain-specific annotations among derivative datasets in a provenance trace.

## 3.9 Related Work

Some of our motifs (non-functional ones) can be seen as higher-level patterns observed in scientific workflows. ”Workflow patterns” have been extensively studied [vdAtHKB03], where inventories of possible patterns are developed based on workflow constructs that are possible in different languages, along with the ways to combine those constructs. Scientific workflows typically use a dataflow paradigm rather than a control flow paradigm that is more typical of business workflows. As observed by a recent study [MGRtH35], scientific workflow systems largely support data-flow patterns[1] and even bring-about new patterns with their varied handling of data tokens.

[1]http://www.workflowpatterns.com/patterns/data/

Data-flow patterns outline ways of managing data resources during workflow execution, such as data visibility and transfer methods. These patterns, which are in essence mechanisms to steward data among activities, are orthogonal to the data-oriented function of those activities within workflows. Our work with motifs largely focuses on this latter orthogonal aspect (i.e. the data-oriented nature of activities) and therefore complements the workflow patterns research. Our work is also based on an analysis of empirical evidence of how data-intensive activities have been implemented against different environments, rather than specifying what is theoretically possible with the given constructs.

In Software Engineering, the term "pattern" refers to established best practices to solve recurring problems. In this regard patterns represent good and exemplary practice. In [CBCM$^+$13] authors outline anti-patterns in scientific workflows, namely redundancy and structural conflicts. The authors go on to provide a solution to address the redundancy anti-pattern. Particularly due to this perception of the term "pattern", in this thesis we opted to use the term "motif" for our classification of tasks. Our objective is to take a snapshot of the existing set of activities in workflows, rather than try to prescribe a best practice.

Our Activity Functional motifs can be seen as a domain-independent classification of tasks within scientific workflows. Similar analyses have been done in a domain-specific manner in areas such as bioinformatics, based on user studies [SGBB01]. Combined with such-domain specific classifications, motifs can make way for specification of abstract workflow templates, which can be elaborated to concrete workflows prior to their execution [GGRF09].

Another work, somewhat closer to our study in spirit, is an automated analysis of workflow scripts from the Life Science domain [WVW$^+$09]. This work aims to deduce the frequency of different kinds of technical ways of realising workflow steps (e.g. service invocations, local "scientist-developed" scripting, local "ready-made" scripts, etc.). This work also drills down into the category of local ready-made scripts, to outline a functional breakdown of their activity categories such as data access or data transformation. While this provides an insight into the kind of activities undertaken in workflows, it focuses on characterising local task types. Our approach is different from this work as we focus on detecting multi-step activities with many realisations (not just individual steps).

Finally, Problem Solving Methods (PSMs) is another area of related work. PSMs describe the reasoning process to achieve the goal of a task in an implementation and domain-independent manner [PB99]. Some libraries aim to model the common processes in scientific domains [GPEG+10], which could be further refined with the motifs proposed in this work.

## 3.10 Chapter Conclusion

In this chapter we reported on the results of an empirical study performed over 260 workflows from a cohort of workflows from 4 systems and 10 scientific domains. The analysis has resulted in a domain-independent classification of workflow elements, namely activity and data motifs. We introduced the Motif ontology that can be used to annotate workflow elements with their associated motifs. The identification of motifs in this Chapter is a pre-requisite to the development of experimental metadata provisioning techniques on Workflow Abstraction in Chapter 4 and Provenance Annotation in Chapter 7.

# Chapter 4

# Workflow Abstraction

## 4.1 Chapter Introduction

In this chapter we present our investigations towards tackling **provenance complexity**, and show how the activity characterisations identified in Chapter 3 can be put to use for abstracting workflows.

We begin in Section 4.2 by recalling the dual role of workflows as both documenters of analytical protocols and implementers of protocols in heterogeneous resource environments. We illustrate with an example that the implementer role inherently embodies complexity while the documenter role requires simplicity. In this dissertation research we target a specialised case of provenance complexity, more specifically we target structural complexity of workflow descriptions. In Section 4.2.1 we depict the landscape of provenance complexity in our context and identify how complexity of workflow provenance affects complexity of experiment reports. In Section 4.3 we revisit the two design practices, earlier identified as Motifs in Chapter 3, that scientists use to encode abstraction into workflow designs. Abstraction, particularly for the purpose of security, is an area active of research. In Section 4.4 we provide a generic blueprint to outline what constitutes the defining characteristics of a provenance abstraction system. We use this blueprint in Section 4.5 to comparatively review the state of the art.

Our approach to abstraction is based on the manipulation of workflow description graphs with well-defined primitives so as to reduce them to structurally simpler forms. We introduce required formal background on graph based representation of workflows and the algebraic notation to represent graph transformations in Section 4.6. In Section 4.7 we declaratively specify three primitives for transforming workflow graphs.

This is followed in Section 4.8 by the characterisation of our approach against the abstraction blueprint, and details on its procedural realisation. In Section 4.9 we discuss the results of our assessment of the reductive capabilities of our primitives and our assessment on how much abstractions created automatically overlap with abstractions created manually by scientists.

Earlier versions of work described herein has been published as a conference paper:

- **P. Alper**, K. Belhajjame, C. A. Goble, and P. Karagoz. Small is beautiful: Summarizing scientific workflows using semantic annotations. In Proceedings of the 2nd International Congress on Big Data (BigData 2013), pages 318-425, Santa Clara, CA, USA, June 2013, IEEE.

## 4.2 Motivation

In Chapter 2 we discussed that the transparency provenance brings into analytical processes is a desired property and facilitates understandability, experimental audit and repeatability. On the other hand transparency can be a double-edged sword when provenance traces are too *revealing* or too *detailed*. *Provenance Abstraction* is a current and active thread of research [CP14]. When we look at literature from this area we observe two major motivations for abstraction. First one is *privacy and security*, where exposing the entire provenance record may compromise data confidentiality, or may degrade the security of a system by exposing vulnerabilities [CP14]. The second driver is *simplicity* [BCBDH08]. Provenance records, especially those automatically collected from instrumented or monitored execution of systems -be they curated databases, workflow engines or file systems- are known to be complex, often to the degree that the size of the provenance metadata far surpasses the data itself [CJR08].

Simplicity is also the driver of abstraction in our context, when reporting workflow based experiments. Here scientists are expected to provide a truthful but not overwhelmingly detailed account of their analysis and results. The experiment reports we studied in Chapter 2 showed that abstraction is needed when reporting both the data and the method. When formulating the provenance gap we identified workflow descriptions as an important opportunity for scientists to report the scientific method, on the other hand our analysis of existing workflow sets in Chapter 3 showed us that workflows are also artefacts of implementation, and they therefore contain an abundant number of steps dealing with resource integration and data grooming. Empirical evidence showed the average proportion of such steps may be as much as 70% for the case of the Taverna workflows. Figure 4.1 illustrates a typical example of a workflow abundant in adapter steps. This workflow is part of the Huntington Disease (HTTD) case study previously introduced in Chapter 1. HTTD workflow uses the Anni web services for its analysis. First a set of gene ids is mapped to a set of concepts from a community-agreed vocabulary. Afterwards the Anni literature mining service is used to obtain association scores between these gene concepts, and a pre-determined meta-concept of interest (such as the *diseases* concept). For each matching disease concept, the underlying literature references that contribute to the association score are retrieved; finally the details on each matched concept, i.e. particular disease, is also retrieved. These are four high-level tasks performed by four web-service calls to an Anni end-point. Meanwhile the scientist has used an additional 27 adapter activities in order to cater for the protocol and format expectations of these web services. While it is necessary to have these steps so as to deal with heterogeneity in a systematic manner, they obfuscate the scientific methodology implemented with the workflow.

Abstraction is desired here as the sheer number of adapters obfuscate the scientific intent of the analysis [GAB$^+$14]. Adapters not only complicate the process, they also complicate the result data space when a workflow is exected. For example data grooming steps lead to several content-wise redundant data artefacts. On the other hand external resources, such as analytical tools or web services have a resource specific footprint in the result space, such as status logs, timestamps, protocols specific wrapping, attachments, which are crucial in debugging analyses but needs to be abstracted away during reporting. In short, without any abstractions workflows and their execution traces are ill-fit for use in reporting.

In the following subsections we first outline what provenance complexity stands for in our research context, and afterwards we identify possible strategies for dealing with complexity.

### 4.2.1 Layers of Provenance Complexity

Prior to discussing abstraction, we need to establish an understanding of the objective of abstraction, which is reducing complexity. Complexity of an artefact is a quality characteristic, that is often defined with respect to a particular end-use scenario such as the artefact's ease of maintenance, or its understandability. Our focus in this dissertation is provenance in the form of workflow provenance and experiment reports; therefore we need to establish our understanding of complexity for these artefacts. We

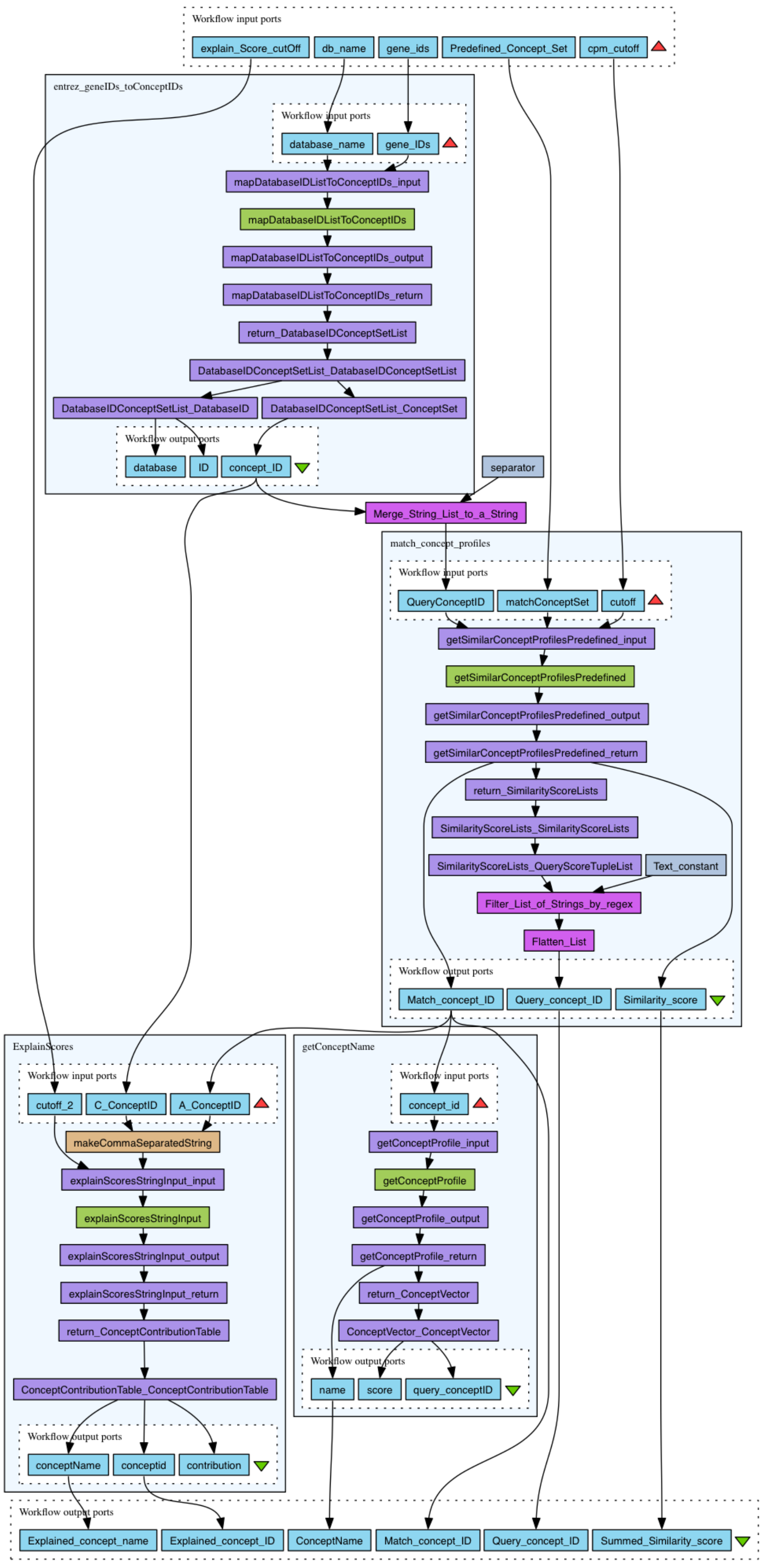

Workflow input ports
explain_Score_cutOff
db_name
gene_ids
Predefined_Concept_Set
cpm_cutoff
entrez_geneIDs_toConceptIDs
Workflow input ports
database_name
gene_IDs
mapDatabaseIDListToConceptIDs_input
mapDatabaseIDListToConceptIDs
mapDatabaseIDListToConceptIDs_output
mapDatabaseIDListToConceptIDs_return
return_DatabaseIDConceptSetList
DatabaseIDConceptSetList_DatabaseIDConceptSetList
DatabaseIDConceptSetList_DatabaseID
DatabaseIDConceptSetList_ConceptSet
Workflow output ports
database
ID
concept_ID
separator
Merge_String_List_to_a_String
match_concept_profiles
Workflow input ports
QueryConceptID
matchConceptSet
cutoff
getSimilarConceptProfilesPredefined_input
getSimilarConceptProfilesPredefined
getSimilarConceptProfilesPredefined_output
getSimilarConceptProfilesPredefined_return
return_SimilarityScoreLists
SimilarityScoreLists_SimilarityScoreLists
SimilarityScoreLists_QueryScoreTupleList
Text_constant
Filter_List_of_Strings_by_regex
Flatten_List
Workflow output ports
Match_concept_ID
Query_concept_ID
Similarity_score
ExplainScores
Workflow input ports
cutoff_2
C_ConceptID
A_ConceptID
makeCommaSeparatedString
explainScoresStringInput_input
explainScoresStringInput
explainScoresStringInput_output
explainScoresStringInput_return
return_ConceptContributionTable
ConceptContributionTable_ConceptContributionTable
Workflow output ports
conceptName
conceptid
contribution
getConceptName
Workflow input ports
concept_id
getConceptProfile_input
getConceptProfile
getConceptProfile_output
getConceptProfile_return
return_ConceptVector
ConceptVector_ConceptVector
Workflow output ports
name
score
query_conceptID
Workflow output ports
Explained_concept_name
Explained_concept_ID
ConceptName
Match_concept_ID
Query_concept_ID
Summed_Similarity_score

Figure 4.1: The Gene Annotation Pipeline of the Huntington's Disease Study

depict the layers of provenance information and the dependencies among their complexity in Figure 4.2.

The **Experiment Reports** we exemplified were tables outlining experimental results and metadata, or diagrams outlining the analytical process. In addition to these, scientific workflows in their raw (workflow-system specific) or technology-independent forms published in experimental bundles [HDZ$^+$14] or in repositories [DRGS08] also stand-in as reports on method. Artefacts at this layer are to be inspected and analysed by scientists to make sense of experimental outputs. Therefore their understandability is critical. Prior research has shown that there is a negative correlation between the structural complexity of diagrams[1] and their understandability [CLMG$^+$10] [MCGP08]. Therefore structural complexity is an important target for our research.

**Workflow descriptions** are both documenters of analytical method, and they are executable programmes. Complexity for workflows can therefore be defined in the context of these two roles. As documentations of the method to be analysed and understood by users the structural complexity of workflow descriptions is important. The structural complexity can be defined in term of the elements that make a workflow description graph, such as of number of steps, ports, dataflow links, dataflow paths. The structural complexity of workflow descriptions is also determinant in the structural complexity of provenance traces obtained from workflows. A structurally complex workflow will ultimately yield a structurally complex execution trace where each port foretells data artefacts, analytical steps foretell activities and the anticipated lineage among output and inputs of activities. A second type of complexity for workflows is their cyclomatic or control-flow complexity, which is used to assess the ease of maintainability of a program. This is a well-studied topic particularly for artefacts that adhere to the traditional Von Neumann (i.e. control-flow) architecture of computing; such as software programs [McC76] or business processes [GL06] [Car05]. Despite having few control constructs, those few constructs in workflow descriptions (such as iteration, looping features) is a significant determinant of the structural complexity of provenance traces obtained from their execution. e.g. Consider two workflow description graphs that involve the same number of steps, ports and dataflow connections, but one involving an iteration configuration over the activity, whereas the other not. In this case the workflow with the control-flow construct would result in structurally more complex traces. Depending on the type of iteration constructs, lineage can be longer/deeper (the unfolding of each execution a feedback loop construct as in Kepler)

[1]Specifically, UML state-chart and class diagrams

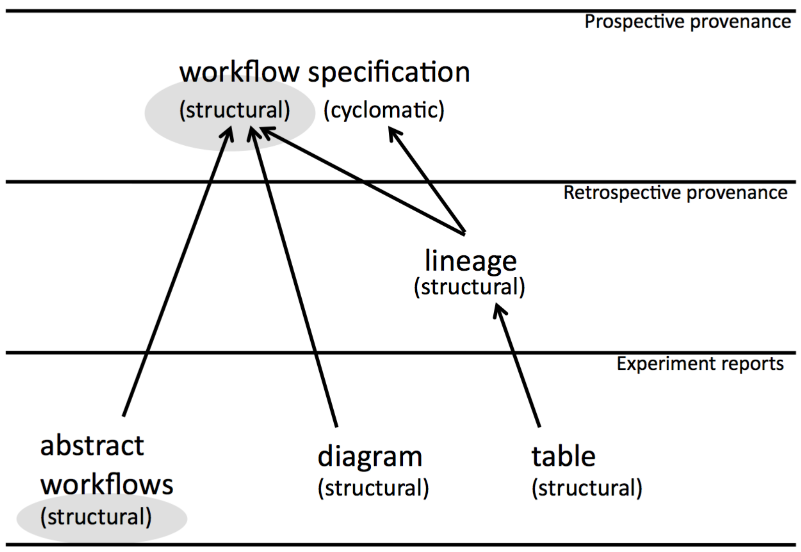


Figure 4.2: Layers of Provenance Artefacts and Dependencies among their Complexity

or there could be multiple lineage relations among fine-grained data artefacts (as in collection driven iterations of Taverna).

Finally the structural complexity of **execution provenance** can be a determinant of experiment reports especially for the reports on data. Recall from Chapter 2 that there are already tools [BKSRS12] for mapping of provenance DAGs into spreadsheets. In these approaches distinct entity and activity types becomes columns of the table and each path in a provenance trace becomes a row.

For such mapping structurally complex lineage would result in complex tables, more specifically deep lineage will correspond to several columns in the table and multiple lineage paths will lead to several rows. **Our dissertation research focuses on the structural complexity of workflow descriptions, henceforth in abstraction we aim to reduce this complexity.** Specific measures of this complexity we use are presented in the Evaluation section.

### 4.2.2 Strategies for Abstraction

Broadly, there can be two strategies for abstraction:

1. Preempting complexity by encoding abstractions into the design of the computational instrument used for data processing. As our motif analysis showed, this strategy is adopted by scientists when designing workflows. Scientists encode abstractions into workflow design, and later exploit these abstractions in

accessing workflow results or in reporting the method. A distinctive characteristic of encoded abstractions is that they are design-bound and static, in other words they are encoded once and used several times over all invocations of the same workflow. In Section 4.3 that follows we discuss the different motives that abstractions serve.

2. Devise abstractions post-hoc, either over the workflow designs or over execution provenance traces. All of the state of the art research on (semi)automated provenance abstraction, which we review in Section 4.5 adopts this approach. Unlike encoded pre-hoc abstractions, post hoc abstractions are not design-bound they therefore can be shaped dynamically using an abstraction policy.

In Section 4.3 that follows we briefly review the state of the art in the first strategy. Following that, in Section 4.4 we provide a blueprint for computation assisted abstraction. We use this blueprint as a guide to explain our approach in Section 4.7 and also to compare it to related work in Section 4.5.

## 4.3 Encoding Abstraction in Workflow Design

When building workflows, scientists utilise workflow design constructs towards **simplifying the analytical process** and **designating data points of significance** in the workflow. These two abstractions correspond to the $Composite\ Workflow$ and $Promoted$ motifs identified earlier Chapter 3. We revisit these motifs to understand the motivation of scientists in using these mechanisms.

### 4.3.1 Using Sub-Workflows

Managing the complexity of a design artefact through introduction of hierarchies/layers is a well-known technique in various fields, such as software engineering [GB84], business process modelling [RM08]. Similarly in the field of scientific workflows layering is encouraged as a best-practice [HWB$^+$12] for designing workflows.

The sub-workflow construct allows pushing down a group of selected activities to a lower layer of design, therefore simplifying the upper layer. Workflows that exploit web-based resources, such as the HTTD example contain several adapter activities and heavily use sub-workflow mechanism. In a layered workflow design the top-most level provides a coarse view of implementation, which more closely resembles the scientific method followed in the analysis. Figure 4.1 illustrates how the scientist has created four sub-workflows so as to encapsulate the four significant data analysis tasks. The motives for using sub-workflows is various:

- *Functional Modularity:* In Chapter 3 we introduced Component Workflows, where scientists modularise the smallest, meaningful, re-usable unit of functionality as sub-workflows. The four sub-workflows in Figure 4.1 are examples of such design, where scientific tasks are grouped with the boilerplate tasks needed to facilitate their execution, such as input preparation and output extraction. Multiple layers of nesting, is also a common design pattern to create modules of composite functionality. Component workflows are of re-use potential and they are often shared in workflow repositories for incorporation into other studies.

- *Experimental Documentation:* Sub-workflows may be used to impose a temporal "experimental phase" perspective on to the study; a design practice commonly used for documentation purposes. In such cases the overall experimental workflow is comprised of a chain of Phase Sub-Workflows. In the ENM workflow given earlier in Chapter 2 there were sub-workflows named such as Run_Cross_Validation, Run_Projection etc. These category of sub-workflows often build on component type sub-workflows. Another notable characteristic is that "phase" sub-workflows may contain multiple parallel threads of data processing. Sub-workflows created for phase documentation are highly context/-experiment dependent, consequently, such workflows are rarely shared with the purpose of re-use in other studies.

- *Implementation Concerns:* Realities of implementation may also direct scientists to group selected tasks into sub-workflows. More specifically, 'Sub-workflow interoperability" [AIG12] [PWH+11] refers to the ability of a workflow system to delegate the execution of a sub-workflow to the execution environment of another workflow system. Consequently scientists may group tasks that they wish to be executed on a particular external run-time into a sub-workflow. Another example is workflow systems that lack constructs looping a body of activities, like the Taverna system. Here sub-workflows are used as a means to enable the iterated execution of a collection of activities, bundled into a sub-workflow in the main workflow.

- *Aesthetic Simplification:* Sub-workflows may also be used solely for aesthetic simplification. In these cases, the sub-workflow does not correspond to a modular functional unit, an experiment phase, or a sub-part that needs to be run on a different execution environment.

### 4.3.2 Promoting Intermediary Values

In practice when a workflow is modularised using sub-workflows, some activities and their input/output ports are pushed back (hidden). This way the design at the top level becomes simpler and contains less number of activities, ports, and when executed results in a more compact data lineage. Scientists may also abstract workflows by lifting up certain intermediary ports, ones they deem significant (report-worthy). The design practice enabling this selection, as we identified in Chapter 3 is the promotion of intermediary ports to become workflow output ports. We refer to this practice as *lineage bookmarks*, others have referred to such behaviour as "trace links" [CBCG$^+$14]. In the HTT workflow in Figure 4.1 the outputs named "Matched_Concept_Id", "Query_Concept_Id" and "Summed_Similarity_Score" are examples of such bookmarks. Bookmarking provides a quick and easy way to collect report-worthy data of interest from the experiment run, without having to rely on provenance collection and querying capability.

In short, encoding abstraction into workflow design is a non-trivial design process with several drivers. There exist significant research and tool support in aiding scientists to discover and integrate resources from respective registries [BTN$^+$10] [MKRI15], and assist their composition in workflows [WKM$^+$10]. On the other hand there is little support in assisting scientists on **organising or refactoring a workflow design**. Consequently pre hoc abstraction is an established, yet predominantly manual process.

## 4.4 Blueprint for Automated Abstraction

The state of the art in computation-assisted provenance abstraction is comprised of a number of recent research systems. A common characteristic among all is that they're designed for abstracting provenance post- hoc i.e. post-collection; and, for most, information privacy is the main driver. A recent paper by Cheney and Perera [CP14] provides an excellent survey of security-motivated provenance abstraction. The authors provide a meta-categorisation identifying the features of systems in terms of 1)

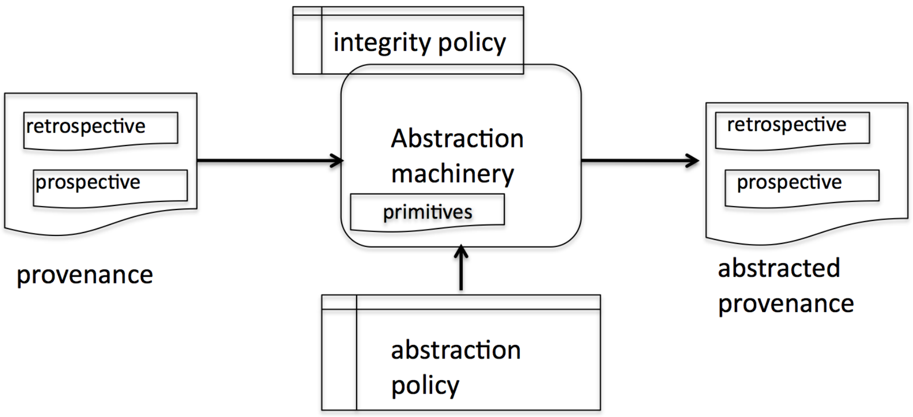


Figure 4.3: The Abstraction Blueprint

functional aspects, such as the level of granularity with which a provenance graph can be sanitised, the notion of integrity of sanitised graphs and 2) non-functional aspects, such as the (non)existence of an underlying formalism and whether conflict-checking or traceability of the abstraction process is possible. In this chapter we expand on the functional aspects identified by Cheney and Perera, and provide a blue-print for provenance abstraction machinery. We discuss and compare related work in Section 4.5 using this template and later present our system accordingly in Section 4.8. Figure 4.3 presents the blueprint as a high-level process. At the core of the process exists an *Abstraction Machinery*, which takes as input *provenance* and an *abstraction policy* and produces *abstracted provenance*. The machinery may commit to a set of *integrity policies*, which may be fixed/built-in or configurable.

Provenance information outlines a network of objects with various relations among them, such as ancestry or causality. As such, the most popular representation for provenance is the graph-based representation. In our blueprint the input to the abstraction process and its output are graph structured provenance. Informally; the process of abstraction is an act of creating a reduced provenance graph $G'$ out of the given provenance graph $G$. Depending on the approach taken for abstraction it may be possible to define a formal relationship between the $G$ and $G'$. One example is the *subgraph* relation, in cases where the abstraction is comprised of elimination of elements (vertices and edges) from G. Another example is the *quotient graph* relation. Here the output graph represents a *view* over the input graph, and it is obtained by the partitioning the input graph into distinct subsets and inducing a coarser output graph using these partitions [CP14].

### 4.4.1 Abstraction Policy

Abstraction is driven by the abstraction policy. This designates what needs to be hidden, or abstracted out, from the provenance graph, in secure provenance literature this is referred to as the *obfuscation policy* [CP14]. In this thesis we use the term *Abstraction Policy* to avoid any connotations with the word “obfuscation”. In addition to hiding, the abstraction policy may also specify what needs to be retained/kept during abstraction, this is known as the *disclosure policy* [CP14].

The elements in the graph that are to be subjected to abstraction are often identified by a graph **query**, this allows for selecting nodes of interest/disinterest with the possibility of identifying structural organisation patterns among nodes. Nodes may be identified directly by pinpointing them with unique node **identifiers** or indirectly by referring to **attributes** associated with graph elements (e.g. data artefacts nodes generated by a certain type of agent). Policies with direct reference to nodes can be considered input-dependent policy, whereas those based solely on attributes can be considered input-independent. Depending on the approach taken, the abstraction policy may be thick or thin. A thick policy would not only identify which elements should be acted upon but it would also specify how they should be acted upon, i.e. how the provenance graph should be manipulated. The manipulation operations are captured by abstraction **primitives**. In addition to specification of manipulations, in a thick policy the policy designer may be expected to specify the order with which manipulations should take place.

### 4.4.2 Integrity Policies

Provenance abstraction machinery may commit to a set of integrity policies. These are characteristics that help us understand the informativeness of the abstraction result, its validity and well-formedness. We have compiled a set of policies compiled from provenance abstraction literature [CP14][DZL11]. In the following sections we discuss these characteristics informally, the formal characterisation of each policy for the particular formalism chosen to represent our approach is given in Appendix B.

#### 4.4.2.1 Dataflow Preservation (Completeness)

Cheney and Perera define an abstraction to be *path preserving* if for any existing path among two nodes in the original graph there exists a path among corresponding nodes in the abstracted graph (where a path denotes a direct or indirect dataflow dependency or lineage). Biton et al [BDKR09] identifies the same characteristic as provenance view *completeness*. As we shall see in our literature review later, compromising dataflow completeness only occurs in abstraction approaches motivated by security. In such cases lineage relations are deliberately redacted out from provenance graphs. In the context of abstracting for simplicity, existing automated approaches are dataflow preserving. The layered workflow designs created by users are also dataflow preserving abstractions.

#### 4.4.2.2 Dataflow Reflection (Soundness)

Biton et.al. [BDKR09] state that an abstracted workflow (a workflow view) is *sound* if it includes only those dataflow dependencies in the original workflow. In other words a *sound* abstraction does not introduce false dataflow dependencies. Most typically in the context of abstractions of the quotient kind i.e. inducing a coarser provenance graph by grouping nodes, it is possible that false dependencies are introduced, and the resulting provenance is unsound. We depict an example in Figure 4.4. Here there are two activities $a$, and $b$. In the original graph all data nodes except the ones pointed with arrows has lineage relations among them, inferred from traversing data usage and generation relations. The two highlighted nodes however are not reachable from each other, hence they do not have a lineage relation among each other. When a view over this graph is generated, by grouping two activity nodes, and a shared data node among them, the resulting view is unsound. The composite activity node now creates a reachability between the prior mutually unreachable data artefacts. Note that unsound views are not necessarily an undesired result. The sub-workflow abstractions created by scientists during workflow design may create non-sound dataflow dependencies among input/output ports of analytical steps.

#### 4.4.2.3 Validity

Validity corresponds to the recognisability of the abstraction result with respect to standard provenance models/vocabularies such as PROV[BDG$^+$12] or OPM[MCF$^+$11]. Models bring restrictions on the kinds of nodes that can occur in a provenance graph and their allowed relationships.

In abstractions based on free-style grouping of nodes, a result may contain a *false-structure* (as termed by Dey et al [DZL11]), or a *type violation* (as termed by Missier et al [MBG$^+$14] ). Here a group of heterogeneously typed nodes are replaced with a single node, without adjusting the incoming and outgoing relations of the group

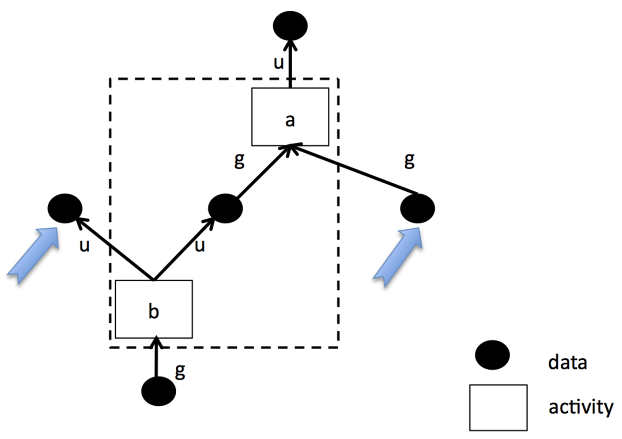


Figure 4.4: An unsound view created by grouping nodes in a retrospective provenance trace.

members, where, for instance, a replacement activity may appear to be *used by* another activity. Validity result abstractions is often a desired characteristic as it allows both provenance and provenance abstractions to be processable with the same provenance tools and libraries.

We now move onto well-formedness characteristics, these corresponds to characteristics of abstracted graphs, where - in a certain context - their existence (or absence) may be desired, but the contrary case would not invalidate the provenance graph. In our context, we identify *acyclicity* and *bipartiteness* as relevant wellformedness characteristics.

#### 4.4.2.4 Acyclicity

Cycles can occur in provenance in the following layers:

- **Cyclic definitions in prospective provenance:** Most workflow systems, including Taverna, Galaxy, Wings, Pipeline Pilot, KNIME generate workflows with directed acyclic dataflow dependencies. The Kepler system stands out as it allows cyclic dataflow dependencies, more specifically, these feedback loops, used for iterating groups of activities over incrementally supplied/refined data. A workflow description containing a feedback loop is given in left-hand side of Figure 4.5. Upon execution, a feedback loop unfolds into an acyclic account of execution including repeated invocations of activities within the loop. This is illustrated in right-hand side of Figure 4.5.

- **Cyclic definitions in retrospective provenance:** From a provenance modelling perspective cycles are not banned. It is possible, for instance, with a vocabulary like PROV, to give account of a long running activity *p* (given in left hand side of Figure 4.7) that creates an interim data artefact, which is used by another process *q*, which generates another artefact in turn used by *p*. While modelling such an account of a process is possible, it is not possible to encode such a process as a workflow description. The setting we just described would require a workflow activity to initiate execution on partial input availability. While workflow systems support input streams from individual ports, they expect data expected at all of the input ports of activity to be available (to have started streaming) so that the execution can start. In short while cycles at this level are theoretically permitted, they are a very rare sight in actual provenance traces.

- **Cycles created during provenance abstraction (retrospective/prospective):** It is very well possible that cycles are introduced to provenance during abstraction with node groupings. A cycle occurs when a group of nodes is not a Convex Hull: indicating that there exist path, that originates from a node in the group, visits a node outside the group, and, then arrives back at a node in the group [DZL11]. In our literature review of existing systems we observed that the large majority of them avoid cycles. This avoidance lies in the inherent complexity in querying, traversing cyclic provenance. Also when reporting data, as with the ISA-Tab mappings of graphs to spreadsheet, only acyclic provenance graphs are supported. An exception is the ZOOM system [BCBDH08], which allows cyclic views over workflow descriptions to be generated. A graphical depiction of this is given in Figure 4.7. We speculate that allowing cycles may be intended for making abstraction approaches more aggressive. On the other hand as we discussed, as the execution scheme in workflow systems expects data in all input ports to be available to some degree, such abstractions would not have a correspondence in real-life workflows.

#### 4.4.2.5 Bipartiteness

Often, there is a high-level semantics that is involved in the (first-hand) creation of provenance e.g. workflow languages or file system operations. Source system semantics imply certain characteristics or patterns in the provenance graph, for instance, provenance graphs originating from workflow executions are bipartite [DZL12]. In

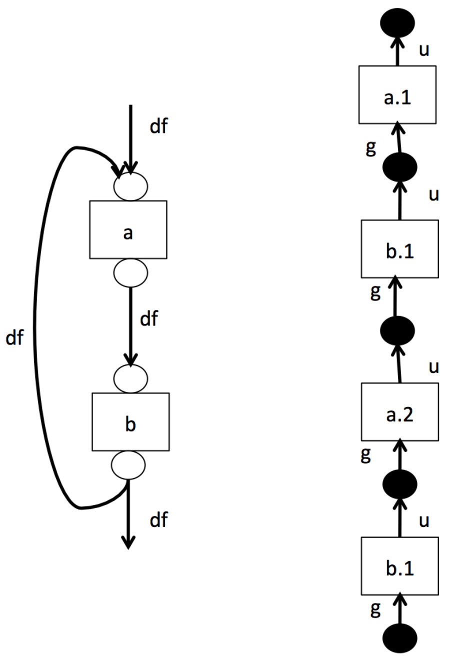


Figure 4.5: Left: A feedback loop specification in prospective provenance. Right: Its unfolding in retrospective provenance.

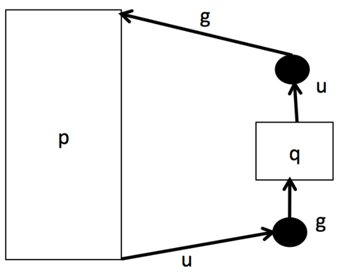


Figure 4.6: Cyclic lineage in retrospective provenance caused by process supporting partial input availability at start.

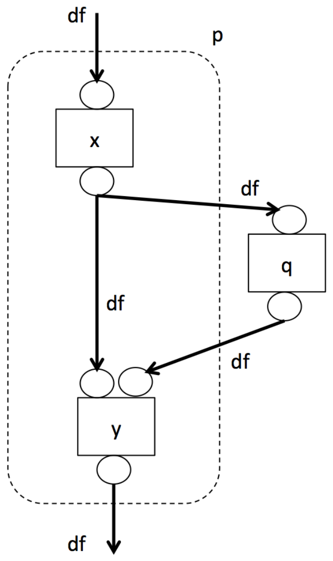


Figure 4.7: An activity grouping in workflow description that could cause a cyclic view over retrospective provenance as in Figure 4.6.

these graphs edges link nodes from disjoint sets of activity and data nodes. By traversing a bipartite provenance graph one receives an account for the creation of a result with the $\mathit{data} \xRightarrow{\text{generatedBy}} \mathit{activity} \xRightarrow{\text{used}} \mathit{data}$ pattern. This way of exposing provenance is needed in several scenarios. When provenance is used for debugging, one needs to understand whether an erroneous input truly affects its descendent data by inspecting the process that link those data. Presenting provenance in a bipartite account, is also a recommended practice in reporting scientific data. The ISA-Tab specification, puts protocols at the heart of its conceptual model, and state that ISA-Tab compliant spreadsheet should present results by stating the protocol that generated that result and its input, and configurations.

On the other hand bipartite accounts are not the only way employed for representing and consuming provenance. The opaque lineage relations among data are often used for basic traceability. For instance, while ISA-Tab recommends a presentation that is inherently bipartite when we look at the use of ISA-Tab compliant metadata published in data repositories [SZE$^+$14] [nat15] we observe that the protocol component can be missing, and results of an investigation may simply be presented with respect to their lineage to input configurations of the overall experiment.

## 4.5 Related Work

We now survey the state of the art with dimensions outlined in the abstraction blueprint. More specifically we seeks answers to the following in each system:

- what is the objective of abstraction, *security* or *simplicity*?
- does the system operate over *retrospective* or *prospective* provenance graphs?
- what comprises the abstraction policy? Does it refer to elements of graph generally by mentioning attributes (*attr*), individually by giving identifiers *id*? Does it involve the specification graph patterns (*qry*), to select elements to be abstracted? Does it require elements of an abstraction policy to be ordered (*order*)? Does the abstraction policy state how abstraction should occur (*prim*) or is it determined internally by the system?
- what kinds of primitives are supported for abstracting the provenance graph?
- which of the integrity policies (namely (Dataflow) Completeness, (Dataflow) Soundness, Acyclicity, Bipartiteness and Validity) are supported by the system's abstraction primitives?

We review seven systems comparatively, namely ZOOM [BCBDH08], ProPub [DZL11], ProvAbs [MBG$^+$14], TACLP [DCMB15], SecProv [CSL$^+$08], Provenance Redaction [CKKT11] and Surrogates [BCS$^+$11]. For the reader to follow Table 4.1 provides the comparative view of characteristics of each system.

**ZOOM** [BCBDH08] is designed for scaling down results of lineage queries over provenance via creating *dynamic user views* over workflow descriptions. An abstraction policy is simply a set of ids for activities that a user deems important/relevant in the workflow. This policy is then used to produce an abstracted workflow by algorithmically partitioning activities into groups, where each group contains at most one important activity. The dynamic nature is the ability to generate different views over a workflow description based on different input policies. This capability is an alternative to the static (hard-wired) sub-workflow abstractions that scientists employ during workflow design. The need for such dynamic views over workflow descriptions is yet to be documented with user studies or real-world use-cases. In the case of experiment reporting, for instance, we have not observed the use of changing user views, instead the static workflow abstraction encoded into the design has been used for reporting all executions of that workflow. In addition to the user-specified significant activities, ZOOM uses structural cues in the workflow as indicators of significance: overall **workflow inputs, outputs** are treated as significant items. ZOOM treats all significant items as **breakpoints** in provenance, and approaches grouping in a manner that preserves soundness of dataflow dependencies among breakpoints. The effect of this integrity policy is that a new partition is initiated whenever a fresh workflow input joins the data processing stream, similarly a partition is finalised whenever either a dataflow link outgoing to a workflow output (i.e. the lineage bookmarking pattern) or a new significant activity is encountered.

**ProvAbs** [MBG$^+$14] provides a framework for abstracting PROV compliant provenance graphs that uses grouping as the prime abstraction construct. The target use-case for ProvAbs is confidentiality protection of provenance. In this approach an abstraction policy is a set of rules which decorates nodes of a provenance graph (confidentiality) sensitivity values. The annotated graph is then abstracted with respect to end-consumer clearance level. During abstraction the system hides away those nodes, whose sensitivity is above the designated clearance level, by using node grouping. Two specialised grouping operators are introduced, one for replacing a group with an entity, another with an activity. The grouping constructs of ProvAbs are semi-autonomous so as to guarantee integrity aspects such as validity and acyclicity. Acyclicity is achieved by enlarging a group to include the convex hull of lineage among its members, whereas validity is ensured by expanding a group until all boundary nodes are of the same type (activity or entity). Due to the autonomy of grouping, nodes not intended to be abstracted away may indeed be abstracted when grouping. ProvAbs measures the proportion of unintended groupings as a combined utility metric to provide as feedback to the abstraction policy designer.

**TACLP** [DCMB15] approach has been developed by the same team that has developed ProvAbs. TACLP stands for Transformation-Oriented Access Control Language for Provenance, and it is an extension of an earlier XML based language [NXB$^+$09] for the specification of access-control policies (e.g. permit/deny ) over provenance graphs. The policy identifies provenance nodes through semantic attributes and how they should be abstracted through transformation operations. Transformations are based on the partitioning of nodes of an OPM compliant graph, for which access is restricted based on the evaluation of an input policy. The focus of this approach is in obtaining 1) node partitions that preserve the soundness of causal relations and 2) node partitions that are minimal in number for a given input graph. These two goals are formalised with the notions of *Causality Preserving Partition* and *Optimal Causality Preserving Partition*. Both notions are predicated on the existence of a particular node in a partition, that can be characterised as the defining node of that partition. Each node in a partition has an associated external Cause/Effect set, which is the set of nodes out of the partition and that are linked with the designated node through causal relations. The defining node of a partition is one whose Cause/Effect set is the superset of Cause/Effect sets of all other nodes in the partition. The existence of such a node not only makes the partitions sound, it also prevents the introduction of cyclic causal relations. TACLP approach provides two graph manipulations: the *Removal* of a partition, or, the *Replacement* of a partition with an abstract node. Authors extend the OPM specification to define 1) a higher level abstract node, from which all other OPM node types (e.g. *Agent*, *Process*) descend and 2) a general *was caused by* relation that can link any type of provenance node. Resulting abstractions are valid against this extended model.

| | **Objective** | **Defined On** | **Abstraction Policy** | **Primitive** | **Integrity Policy** | | | | | **Measuring Effect of Abstraction** |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | **Val** | **Snd** | **Cmp** | **Acyc** | **Bip** | |
| ProvAbs | Security | Retrospective | $\langle qry, attr, id \rangle, order$ | Node Grouping | Y | N | Y | Y | Y | Retained node % |
| TACLP | Security | Retrospective | $\langle attr, order, prim \rangle$ | Remove | Y[a] | Y | Y | Y | N | Minimum number of partitions. |
| | | | | Replace | Y[a] | Y | Y | Y | N | |
| Surrogates | Security | Any graph | $attr$ | Node Surrogating w. associated path surrogation | Y | Y | N | Y | N | % of Retained paths<br>Lost node utility |
| Redaction | Security | Retrospective | $\langle qry, attr, id, prim \rangle$ | Edge Contract | N | N | N | Y | N | % of Retained triples |
| | | | | Vertex Contract | N | N | N | Y | N | |
| | | | | Path Contract | N | N | N | Y | N | |
| | | | | Node Relabel | Y | Y | Y | Y | Y | |
| SecProv | Security | Prospective | $attr, id$ | Filter Lineage & wrt Policy | Y | N | N | Y | Y | - |
| ZOOM | Simplification | Prospective | $id$ | Compose Activities | Y | Y[b] | Y | N | Y | Reduction in lineage qry results |
| ProPub | Both | Retrospective | $\langle id, prim \rangle, order$ | Anonymize Node | Y | Y | Y | Y | Y | - |
| | | | | Hide Node | Y | Y | N | Y | Y | |
| | | | | Hide Dependency | Y | Y | N | Y | Y | |
| | | | | Node Grouping | N | N | Y | N | N | |
| Workflow Summaries | Simplification | Prospective | $\langle attr, order, prim \rangle$ | Eliminate Activity | Y[c] | Y | Y | Y | N | % Reduction in Wf Structural Elements |
| | | | | Compose Activities | Y | N | Y | Y | Y | |
| | | | | Collapse Activity | Y | N | Y | Y | Y | |

[a] TACLP abstractions are valid with respect to a model that is an extension of OPM.
[b] ZOOM abstractions preserve soundness among selected nodes in the workflow.
[c] Workflow Summaries abstractions are valid with respect to a model that is an extension of Wfdesc.

Table 4.1: Comparative Table of Provenance Abstraction Research

**ProPub** [DZL11] is a framework abstracting over OPM-compliant provenance graphs, where both abstraction and integrity policies are taken as inputs to abstraction. ProPub provides a large range of primitives: node grouping, elimination, anonymization, i.e. sanitisation of attributes, and distinctively, a retain primitive for keeping designated nodes during abstraction. In addition, ProPub enumerates a set of integrity policies similar to those identified in Section 4.4.2. The user is expected to program the ProPub machinery by providing an ordered set of abstractions and also an ordered set of preferred integrity policies. The ability to specify both abstraction and integrity policies brings flexibility but also brings the possibility of conflicts. During abstraction ProPub greedily performs the following, checks whether all desired integrity policies can be satisfied against all designated graph manipulations, in cases of conflict, the system attempts at local repairs (such as expanding groups), if a repair is not possible the system compromises/drops an abstraction request (or an integrity policy), and continues until non-conflicting solution is found. A use-case for ProPub in a real-world setting/domain is not provided.

**Secprov** [CCL$^+$10] demonstrates how role-based access control can be applied to nested workflows, when answering provenance queries. The part of the provenance to be abstracted out from a query result is specified through 1) an access-control policy in the form of triples associating eligibility and roles with workflow elements (activities, ports and dataflow links) and 2) a boolean policy designating whether nested/subworkflow details should be visible or not. These policies esentially act as a filter, an ineligible element is either anonymised or eliminated; the choice of action is made in a manner that preserves bipartiteness of provenance. As elements can be eliminated from a trace, original dataflow dependencies are not preserved in this approach, whereas as grouping is not used as an abstraction primitive, lineage soundness is preserved.

**Provenance Redaction** [CKKT11] is an approach that uses graph rewriting for redacting OPM compliant provenance assertions in the form of RDF statements. Here graph re-write rules identify sensitive parts (subgraphs) in the provenance and replace those with opaque (censored) nodes. The authors state that they provide a "high-level" policy specification for encoding redaction rules, yet, we observe that elements of this policy language straightforwardly map to generic components of graph rewrite systems, such as rule, left-hand-side, right-hand-side and gluing instructions. So essentially, a graph rewrite system is at the disposal of provenance redactor. A number of most characteristic redaction rules have been narratively described as edge contraction (replacing two connected nodes), vertex contraction (grouping two disconnected nodes), path contraction (replacing a path with a single edge) and node anonymisation (relabelling). Actual rules are omitted from the paper. The authors acknowledge that it is possible to encode integrity guarantees into rules, yet they fall short of demonstrating how this can be done. For the characteristic rules described, integrity aspects such as avoiding cycles or redundant edges are hinted at but not illustrated or proven. An important aspect, which is the ordering (programming) rules is achieved via dynamic (but costly) (re-)ordering phase prior to each rule application. Order is determined by rule impact on the provenance graph, where impact is the number of statements a rule redacts. As part of evaluation two strategies are explored, first one that priorities the high impact rules, and the second one that priorities low impact rules. Because high-impact rules are more likely to disable other rules from matching, overall abstraction process becomes quicker in the first strategy.

The **Surrogates** [BCS$^+$11] approach promotes the protection of sensitive (confidential) parts of graphs via surrogation (anonymisation) as opposed to a naive elimination. Here the abstraction policy is specified as annotation over the graph nodes and edges. Nodes bear information on consumer privileges necessary for their visibility. Edges have markers on both sides (source/target) designating whether they should be surrogated deleted or kept action should be taken, in case the (target/source) node is subject to some abstraction (elimination or surrogation). Two utility metrics related are defined: one in terms of the percentage of paths preserved, and the other as the loss of information content (say attributes) associated with nodes that are surrogated. The authors outline an algorithmic approach that exploits the annotations for surrogation of nodes and edges, which they claim is maximally informative (with respect to their second metric).

We make the following observations on the state of the art:

- **Level of Abstraction** ZOOM and SecProv systems commit to a particular provenance source, i.e. workflow executions. One advantage of this commitment is they allow specification of abstraction policies on prospective provenance, i.e. workflow descriptions, which are much more intuitive and simple than their retrospective counter parts i.e. execution traces. Other systems like ProPub and ProvAbs have the ability to operate over (any) retrospective provenance graphs regardless of the source it comes from. While this makes the resulting system more generic, it reduces its usability as it expects policy designers to encode node selection and graph manipulation at a lower level of abstraction.

- **Policy Re-usability** ProPub, ZOOM and SecProv systems necessitate the specification of abstraction policies by referring to specific nodes in the provenance graph (via node ids). Such input dependent policies are clearly not re-usable, or applicable to different input provenance. A particular disadvantage of approaches like ProPub, is that their policies are both thick and input-dependent, necessitating significant effort in policy (re)design.

- **Level of User-Control** Control over abstraction can be an essential requirement in certain contexts, such as experiment reporting. As scientists are accountable for the workflow designs and results they publish, they would like to be in control of which parts of their workflow are retained which are hidden during abstraction. Furthermore recall that both process abstraction and data-wise abstractions are needed. Against such settings the output of machinery with certain abstraction autonomy - autonomy stemming from various reasons, such as to keep the policy simple, as in ZOOM; or to auto-fix integrity violations, as in ProvAbs- may be of limited use. We provide an illustrative example Figure 4.8, where shaded areas are the manual sub-workflow based abstractions created by the scientist and the areas denoted with thick dashed lines are the groups created by the ZOOM system (The names for ZOOM groups are direct outputs of this tool). We can see that the boundaries of groupings are at a discord between the manual abstraction and the ZOOM abstraction. This workflow lacks the data-bookmarking design practice, consequently ZOOM's autonomous grouping relies only only positions of significant activities to determine group boundaries. Moreover as all workflow inputs are deemed significant by ZOOM the abstraction generates groups (those named NR-* in Figure4.8) that are devoid of a scientifically significant activity, yet they exist for the sake of preserving soundness among breakpoints (some of which can be insignificant configuration-type inputs). This example shows that semi-autonomous abstractions such as that of ZOOM's may not be usable as-is for scientific reporting. Against this setting there can be two possible remedies:

  - we can either perform abstraction under complete user control: pre hoc as in the current manual practice, or post-hoc through a system like ProPub, which gives increased control to the user
  - another possibility is to make use of abstractions like that of ZOOM not as ultimate end-products but as **suggestions** to the workflow designer while

she is encoding abstraction into design.

- **Cross-Purpose Use** The end-purpose for which the abstraction machinery is to be used is the critical factor shaping the machinery's integrity policies. Consider redaction, where sensitive node(s)/statements are replaced with opaque nodes that do not even bear type information. This example shows how a strong requirement for **information privacy** may compromise even the most basic integrity aspects such as the validity of result graph. In ZOOM , which is designed for **debugging**, there is a built-in policy for guaranteeing a truthful (sound) account of dependencies among significant provenance elements. An observation we make here is the difficulty in cross-purpose use of abstraction machinery. The previous bullet point demonstrated that ZOOM's abstractions can be ill-fit for reporting-use, whereas it is clear that an abstraction approach which does not even preserve validity would be hard to use in any context other than privacy. Among all systems reviewed ProPub stands out as a generic foundational approach, upon which abstraction machinery shaped for particular use can be built.

Notably excluded from our survey are work on workflow/provenance analytics [ESLF09] [Mor15]. These works were excluded as their focus is not on proposing provenance graph manipulation techniques. Instead these approaches focus on identifying information sources (e.g. textual information, structural patterns) in a provenance graph and identifying how that information can be exploited to generate analytical characterisations (e.g. most frequent process branch, frequently co-occurring activities). In [ESLF09] authors describe techniques to generate workflow snippets in response to text-based workflow search requests in repositories. Whereas in [Mor15] techniques are given for efficiently identifying and quantifying patterns in provenance.

## 4.6 Algebraic Graph Re-Writing

In this section we introduce basics in algebraic graph transformation, which we use as a formal notation to depict workflow summarisation primitives. The majority of this section is adopted from the textbook material available in [Roz97, Ch. 3] The origins of algebraic graph re-writing lie in the generalisation of Chomsky Grammars from strings to graphs. The idea is to generalise *string concatenation*, which is the main enabler of grammar productions, to a *gluing* construct for graphs. The "algebraic"-ness of the

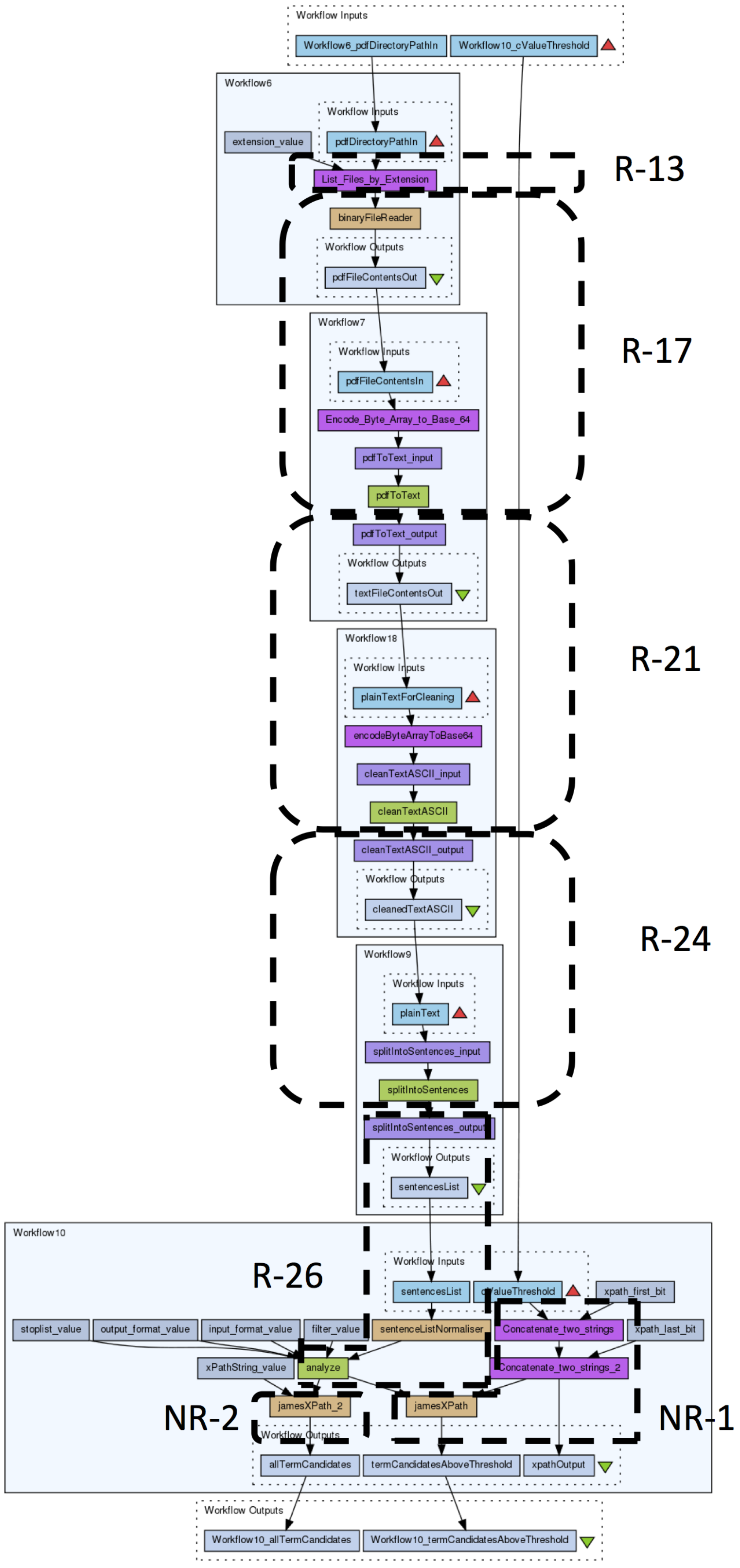


Figure 4.8: The abstraction created by scientist in terms of sub-workflows versus the abstraction generated by the ZOOM system.

approach comes from its use of categorical notions to define graph transformation rules over graphs. In this formalisation directed graphs are seen as special kinds of algebras.

**Definition 4.1.** *(Graph) A graph represented with the six-tuple* $G = (V, E, src : E \rightarrow V, trgt : E \rightarrow V, l_E, l_V)$ *where* $V$*, and* $E$ *are vertices and edge sets with two unary operations that map edges to their source and target vertices, and labelling functions for vertices and edges.*

At the heart of all graph transformations systems lies a graph transformation rule, called a production.

**Definition 4.2.** *(Graph Production)* $p : L \rightsquigarrow R$*, where graphs L and R are called the left- and right-hand-side respectively. An application of a rule is called a direct derivation, denoted with* $G \overset{p,m}{\Longrightarrow} H$*, states that production p is applied to G leading to the derived graph* $H$*. During the derivation H is obtained by replacing an occurrence of L in G with R. An occurrence is formally represented with a match* $m : L \rightarrow G$*, which is a graph homomorphism. A homomorphism is a label and adjacency preserving mapping m from vertices of a source graph, L, to a target graph, G, such that if an edge connects two vertices in* $v1$ *and* $v2$ *in L then there exists a similar labeled edge connecting mapped vertices* $m(v1)$ *and* $m(v2)$ *in G. Note that vertices mapping is also expected to be label preserving.*

Direct derivations are represented with a category-theoretic construct, called the *pushout* depicted below:

$$\begin{tikzcd}
L \arrow[r, "p"] \arrow[d, "m"'] & R \arrow[d, "m^*"] \\
G \arrow[r, "p^*"] & H
\end{tikzcd}$$

Elements in graph $G$ that do not match the left hand side $L$ of $p$ need to be preserved during the application of $p$. This is addressed with the co-production $p^* : G \rightarrow H$ that links the given graph $G$ to the derived graph $H$. Consequently, for each occurrence (match) $m$ there will be an occurrence (co-match) $m^*$ that is a homomorphism from right-hand side $R$ to the resultant graph $H$. Note that in algebraic graph transformation the *pushout* construct has a number of variations and a number of formally established characteristics such as the commutativity of its constituent morphisms where $p^* \circ m = m^* \circ p$. While these are important in the formal study of graph grammars in the context of this dissertation they are not of interest, hence we do not elaborate on such characteristics. The key intuition is that the pushout provides us with a model of computation for graph productions [Roz97, Ch. 3] .

We will illustrate this computation with an example. Figure 4.9 displays a sample production $p1 : L1 \rightsquigarrow R1$, which, informally, is a rule for replacing an indirect connection among particular vertices with a direct connection. As with all productions $p1$ defines a partial correspondence between the elements of its left- and right-hand sides determining, during $p1$'s application, which vertices are to be kept, which are to be deleted and which vertices are to be added (this correspondence can be defined by identities of elements in the graph). When a match $m1$ is found, in order to obtain $H1$:

- we need to delete every element from $G1$, which is mapped-to by an element of $L1$, and that element does not have a correspondent in $R1$. Consequently the vertice $a1$ and the edges that connect it to $d1$ and $d2$ are deleted.
- we need to add to $G1$ the fresh element in $R1$, i.e. those that do not have a correspondent in $L1$
- we need to preserve all remaining elements of $G1$, these could informally be described as un-referred and explicitly-retained elements. Un-referred ones are those elements in $G1$ that do not have an element in $L1$ that map to them. The explicitly-retained ones are those elements that do have an $L1$ element mapping to them where that $L1$ element has a correspondent in $R1$.

The set-theoretic representation of the above computation is $H1 = G1 \setminus (L1 \setminus R1) \cup (R1 \setminus L1)$.

The advantages of using the algebraic approach to denote graph manipulations are several: first of all it provides us with an intuitive, visual yet formal representation, secondly it provides several formal characterisations such as *type graphs*, *functional behaviour* and *negative application conditions*, which we use to describe our abstraction machinery, and facilitates the proofs of integrity policies adopted in our approach. There exist several generic graph transformation tooling [TB94] [Ren03] [GBG$^+$06] based on the algebraic formalism. In this thesis we use the GROOVE graph transformation system [Ren03] to showcase our primitives.

### 4.6.1 Types and Attributes

In order to customise the above outlined formalism to our domain of workflows, we make use of Type Graphs [Hec13] and Attributes [Roz97, Ch. 3].

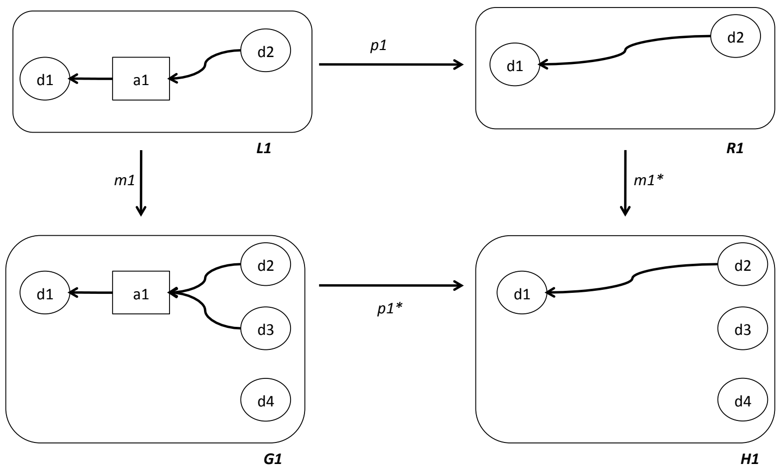


Figure 4.9: Sample production *p1*

**Definition 4.3.** *(Typed Graph) A type graph $TG$ is a fixed graph that contains nodes and edges that are representative of allowable kinds of nodes and edges for a set of graphs [Men00]. Note that definition wise a type graph is not different than a graph as per definition 4.1. A graph G is said to be typed by a type graph $TG$ if there exist a homomorphism $g : G \rightarrow TG$.*

The type graph is a meta graph which allows us to put restrictions on the kinds of graphs allowed. Which in our case is workflow description graphs input to and output from our abstraction machinery. As we shall see later we adopt attributed graphs to represent characteristics of elements in a workflow description. Attribute types are represented with corresponding *attribute nodes* in a graph $G$, where in a graph type morphism $g : G \rightarrow TG$ these nodes are mapped to nodes in $TG$ with labels corresponding to*abstract data types* such as Integer or String or Boolean.

### 4.6.2 Programming Productions

Graph transformations systems are built using graph productions (such as the one in Figure 4.9). The simplest way to build a system is by defining it as a set of productions $M_u = \{p_1, p_2, p_3..., p_n\}$. $M_u$ is unordered in the sense that during its execution, the system would pick up the next direct derivation randomly from a set of matching productions [Men05]. Another way to define a graph transformation system is to introduce order to derivations through a control specification. An ordered graph transformation system, is then a defined set of productions and a control specification defining a partial or complete order over productions $M_o = (\{p_1, p_2, p_3 ..., p_n\}, \preceq)$[BFG96]. A third mechanism in organising rules is nesting [Kup07], where a productions is comprised of sub-productions, the application of which can be quantified universally and existentially.

Depending on the interrelations of the productions, such as causal dependencies, conflicts, a graph transformation system may show functional (deterministic), or non-deterministic behaviour. Functional behaviour implies that a transformation system behaves like a function i.e. that for each input graph there exist a unique output graph. Whereas non-deterministic behaviour means there can be multiple possible outputs depending on the sequence of derivations taken. There are two main strategies to guarantee functional behaviour, one can either design rules in a conflict a causality free manner, or one can use control constructs for ordering rules.

Different behaviours may be suitable for different use cases of graph transformation. For instance in [HKT02] graph productions are used for transformations of models among languages, such as UML to(from) XMI or BPMN to BPEL. In this case functional behaviour is crucial for an unambiguous mapping. Whereas in [HHRV15] productions are used to create and explore a space of multiple (candidate) designs. In our context, i.e. the abstraction of scientific workflows to assist in reporting of their results, we argue that both behaviours may find applicability. We further elaborate on this in Section 4.8.3.

### 4.6.3 Negative Application Conditions

Recall our example production $p1$ in Figure 4.9. The application of the rule leaves a previously connected vertice $d3$, disconnected as a result. In a more mindful design of this rule one could eliminate activity $a1$ only in cases it is solely connected to $d1$ and $d2$ and not to $d3$. These additional patterns in the left hand-side of a rule are restrictions called negative application conditions (NAC).

**Definition 4.4.** *(NAC) A negative application condition (NAC) is a morphism $n : L \rightarrow N$. Where for a match $m : L \rightarrow G$ We say that a match $m$ satisfies a NAC $n$ if there is no morphism $n' : N \rightarrow G$ such that $n' \circ n = m$. A rule is applicable to a graph $G$ if there is a match satisfying all NACs.*

Intuitively a negative application condition represents a graph pattern, which restricts the applicability of a graph production. The extended version of production $p1$

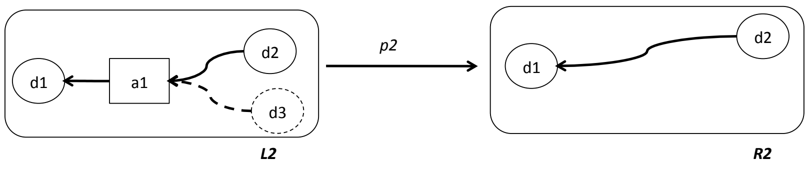


Figure 4.10: Sample negative application condition in *p2*

including a NAC pattern is given in Figure4.10 as $p2$. The NAC pattern is depicted with dotted lines shown jointly with the left hand side $L2$ of the production.

### 4.6.4 Workflow Description Graphs

**Definition 4.5.** *(Workflow Description Graph) A scientific workflow is a directed, acyclic, attributed, typed graph W comprised of a vertice and edge set $W = \langle V,E \rangle$. There are four types of vertices $V = (V_{ACTIVITY} \cup V_{INP} \cup V_{OUTP} \cup V_{ATTR})$, these are respectively: the workflow operations , the input and output ports and data vertices representing values of motif attributes associated with ports and operations. Nodes are connected using four types of edges $E = (E_{inputOf} \cup E_{hasOut} \cup E_{df} \cup E_{iddf})$: linking input ports to the operations they belong, linking operations to their output ports, and linking ports to ports denoting either the direct dataflow among them or indirect influence relation among them. Note that an executable workflow description, which is the input to the abstraction process only contains direct dataflow relations. We use the influence relation in our abstracted workflow descriptions.*

The type graph $T_W$ we use for representing workflows and a sample instance graph representing a fragment of the HTT workflow are given in Figures 4.11 and 4.12 respectively. Both diagrams are screenshots from the GROOVE system.

We will now move onto description of our primitives for abstracting workflow graphs. We specify each primitive formally as a graph production that operates on graphs of type $T_W$.

## 4.7 Productions for Workflow Abstraction

Our presentation of primitives follows a common structure, where provide:

- An intuitive overview of the effect of the primitive,

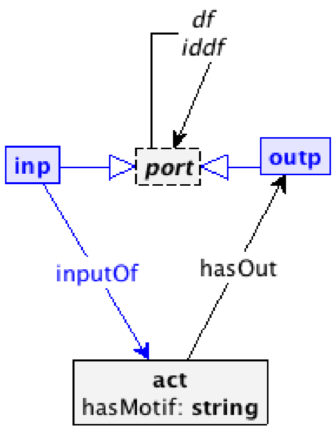


Figure 4.11: Type Graph describing allowed node and edge types in the Left- and Right-Hand sides of our Workflow Abstraction rules

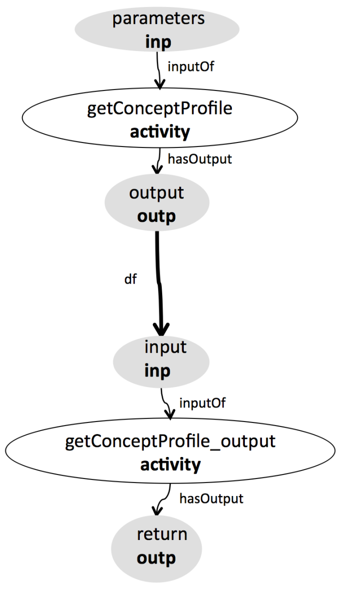


Figure 4.12: Graph based representation of a fragment of the Huntington's Disease Workflow

- An example application of the primitive with a before and after view of a small workflow fragment,
- The formal specification of the primitive as a graph production and discussion of its integrity features

We shall note that these formal depiction of primitives have the following assumptions and restriction:

- our primitives operate over flat workflows, sub-workflows and its impact of abstraction primitives is left for future work .
- the iteration constructs of workflows have not been modelled, similarly left for future work.
- in our productions we assume workflows that have been cleared of intermediary bookmarking patterns, where an output of an activity is forwarded to a workflow output port in addition to another activity's input port.
- we do not specify the motif related behaviour of primitives in our formal notation. We provide information on how motifs drive abstraction in the description of the procedural implementation of abstraction machinery (in Section 4.8).

### 4.7.1 Elimination

**Overview** Elimination primitive removes a designated activity, and its input and output ports from the workflow description graph. When an activity is removed the incoming and outgoing dataflow links (df) would become dangling. In elimination we remove these dangling links and replace them by indirect dataflow links ($iddf$). The indirect dataflow link is a placeholder for some "eliminated" data processing occurring between two ports. If an activity has $l_m$ incoming and $l_n$ outgoing links we introduce $l_m \times l_n$ number of indirect dataflow links upon its elimination.

**Example** A particular application of the elimination rule is given diagrammatically in Figure 4.13. The left hand side displays the Host graph to which the elimination production is applied, and right hand side show the result graph after activity $Z$ is eliminated.

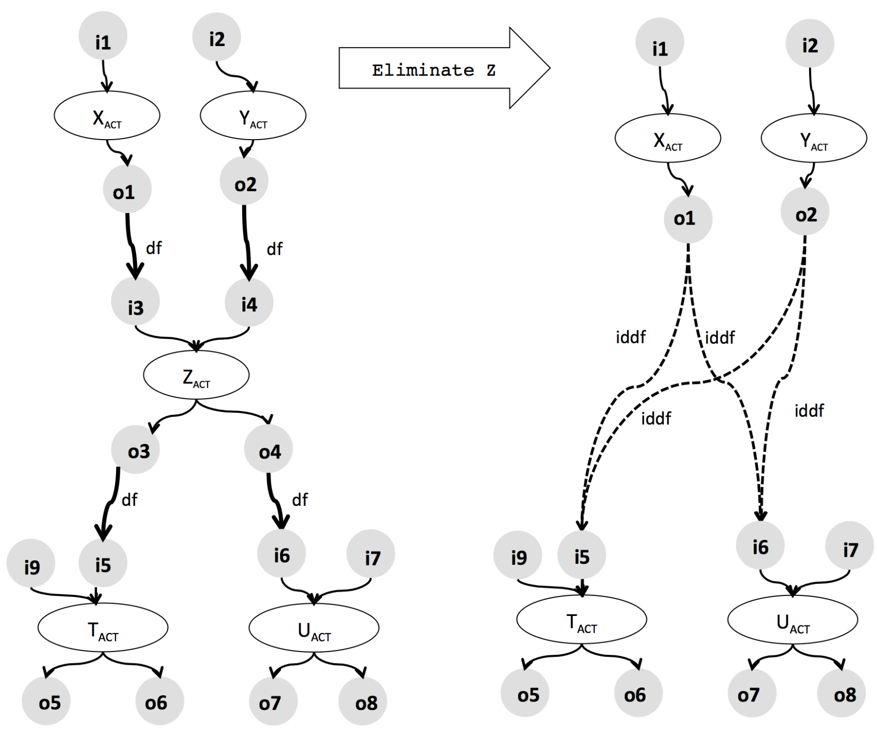


Figure 4.13: Before-After view of a sample application of Eliminate primitive.

**Production** Before presenting the productions we shall note the conventions adopted by the GROOVE system in presenting graph productions. In GROOVE both the LHS and the RHS of the rule are shown in a single graph. The following *line* visualisation conventions are adopted to differentiate between matches and morphisms:

- Medium solid (color black): can occur on both LHS and RHS, elements that are matched and preserved by a production.
- Medium solid dashed (color blue) : can occur on LHS, elements that are matched and eliminated by the production.
- Thick solid (color green): can occur on RHS, elements that are created by the production.
- Thick dashed (color red): these correspond to negative application conditions (NACs).
- Thin dashed (color red): these correspond to quantifications of sub-productions with universal or existential quantification.

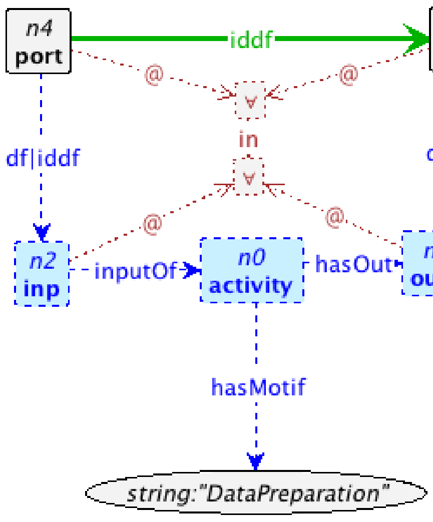


Figure 4.14: The encoding of the Eliminate production in GROOVE graph grammar system.

**Definition 4.6.** (Eliminate) We define the production $eliminate : L \to R$ as one, which matches activities with a designated motif attribute and eliminates them from the host graph. The GROOVE screenshot given Figure 4.14 depicts the $eliminate$ production. In the GROOVE system elements in a rule may be associated with a quantifier. The *in* links among quantifiers allow the rule to be of a nested structure. The elements that are not explicitly linked to a quantifier is considered to be linked to the (unseen) top-level existential quantifier. A semi-formal reading of the rule in Figure 4.14 can be done based on quantifier hierarchy as follows:

(∃) Exists match of node of type activity that is linked to a datatype node with value "DataPreparation", this matched activity node is to be removed.

(∀) For all input port and output port combination for the (prior matched ) activity, matched port nodes are to be removed

(∀) For all link combinations of $df$ (or $iddf$) link incoming to the input port and $df$ (or $iddf$) link outgoing from the output port of the (prior matched) port combination, a new $iddf$ link shall be added from the source of the incoming link to the target of the outgoing link.

As per the definitions given in Section 4.4.2 for integrity policies, the eliminate primitive preserves Validity, Soundness, Completeness and Acyclicity but it does not preserve Bipartiteness. Basic proof argumentations for these are given in Appendix B.

### 4.7.2 Collapsing

Composition, or the grouping together of activities is a well-known abstraction approach for workflow descriptions. The sub-workflows created manually by users, or the Views in the earlier described ZOOM system [BCBDH08] are all based on composites. Our Collapse primitive is a derivative of composition. Collapsing an activity is a way to abstract that activity from a workflow graph by composing it with its immediate upstream of downstream activiti(es) with which it shares a dataflow link. Composites, in the ZOOM or the sub-workflow approaches are $n-ary$, i.e. they are groups of multiple activities. Our collapse primitives define *binary* groupings, but when applied iteratively they have the ultimate effect of creating $n-ary$ groups. We provide Collapse in two forms. Collapse- Up and Collapse-Down, described in the following sub-sections.

#### 4.7.2.1 Collapse-Up

**Overview** The Collapse up primitive abstracts away a designated activity from the graph by composing it with an upstream activity. Note that an activity can have multiple upstream activities, as per the definition of collapse up, we compose the activity with only one designated upstream activity. As a result the collapse up primitive is a binary composition primitive.

**Example** A particular application of the collapse up rule is given diagrammatically in Figure 4.15, where activity $Z$ is collapsed up onto activity $Y$. Here activities $Z$ and $Y$ and their associated elements i.e. port nodes, incoming and outgoing links, are deleted from the graph to replaced with $Y'$ and associated elements. A number of dataflow links and ports associated with $Z$ and $Y$, which we call interface elements are carried forward to $Y'$ by clone nodes. (These are named after their originals with the addition of the prime (') suffix.). The interface elements are the following:

- inputs ports of $Y$: $i2$.
- outputs ports of $Z$: $o4$, $o5$.
- input ports of $Z$ that have an incoming link from some port not belonging to $Y$:no such ports in our example.
- output ports of $Y$ that have an outgoing link to some port not belong in to $Z$: $o3$.

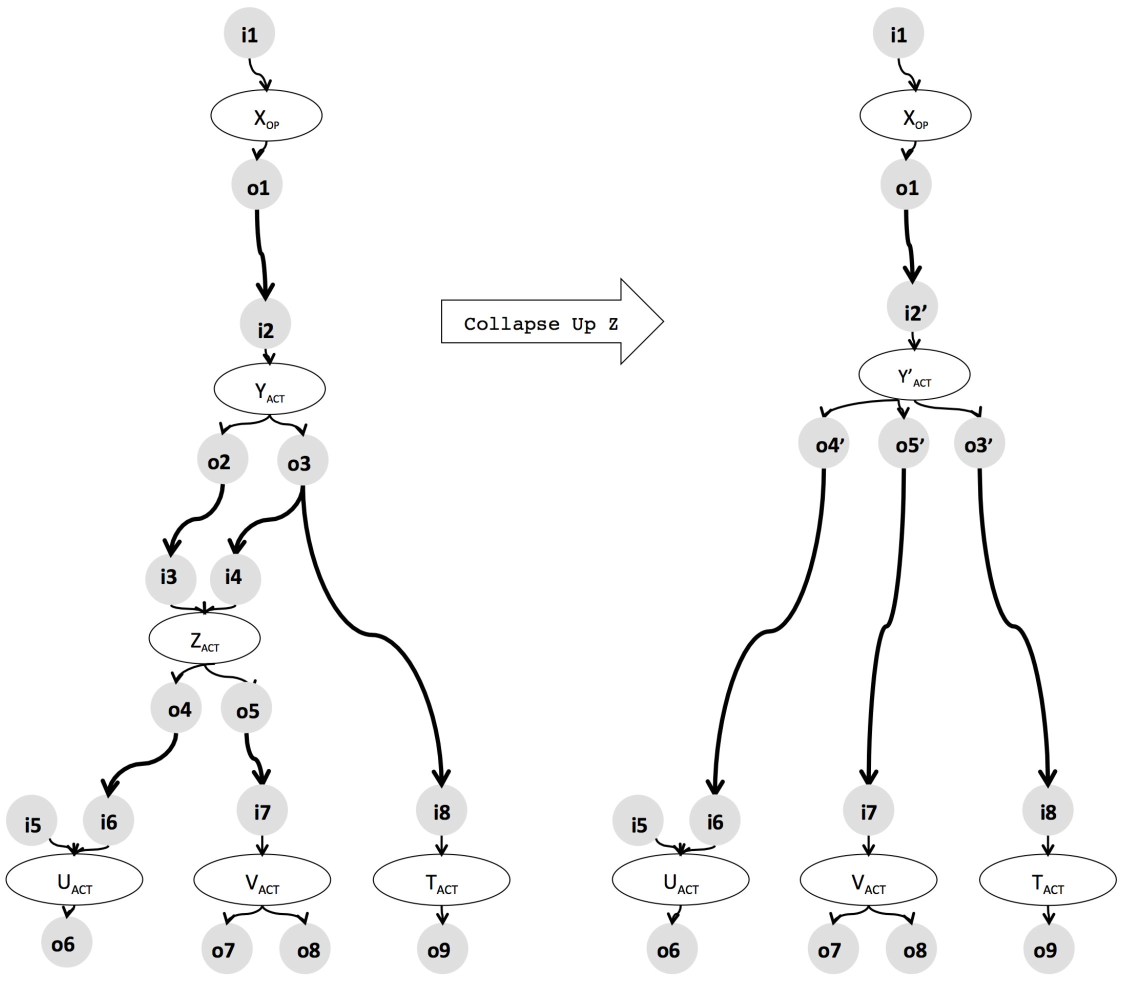


Figure 4.15: Before-After view of a sample application of the Collapse-Up primitive.

**Definition 4.7.** (Collapse-Up) We define the production$collapse_up : L \to R$ as one, which matches an activity with a designated motif attribute and composes it with its upstream processor. The GROOVE screenshot for this production is given in Figure 4.16. When discussing integrity policies we identified acyclicity as a desirable property. On the other hand, as we discussed in related work [DZL11] that node grouping is known to result in cycles. In the collapse up primitive we use Negative Application Conditions (NACs) to prevent the application of rule to cases where activity composition would result in a cycle. The detailed discussion is provided in Appendix B. The reading for the production given in Figure 4.16 is as follows:

($\exists$) Exists a match for a pair of activities (upstream and downstream), with 1) a direct dataflow dependency between them, and 2) a link from the downstream activity to a datatype node with value "DataPreparation", and 3) there exists no NAC match for the activity pair i.e. no indirect dataflow path (passing via a third activity) between the matched pair, then activities in the matched pair are to be deleted, and a new activity (composite) is to be added.

($\forall$) For all output ports of upstream activity in the (prior matched ) pair, and for all input ports of the downstream activity in the (prior matched) pair these ports are to be deleted.

($\forall$) For all input ports of the upstream activity with an incoming dataflow link from some source port, the input port is to be be deleted and a new input port should be added to the composite activity with an incoming link originating from the same source port.

($\forall$) For all output ports of the downstream activity with an outgoing dataflow link to some target port, the output port is to be be deleted and a new output port should be added to the composite activity with an outgoing link from the arriving at the same target port.

($\forall$) For all output ports of the upstream activity with an outgoing dataflow link to some target port that is no the input port of the downstream activity, a new output port should be created and added to the composite activity with an outgoing link arriving at the same target port.

($\forall$) For all input ports of the downstream activity with an incoming dataflow link from some source port that is not the output port of the upstream activity, a new input port should be created and added to the composite activity with an incoming link originating from the sam source port.

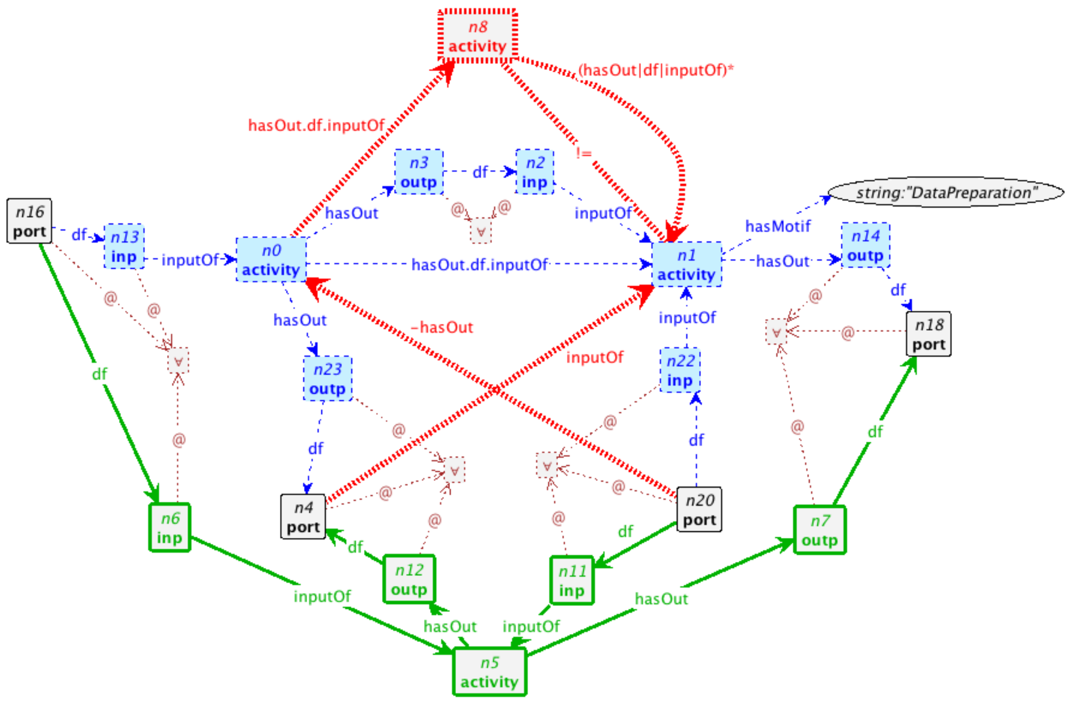


Figure 4.16: The encoding of the Collapse-Up production in GROOVE graph grammar system

The Collapse-Up primitive preserves Validity, Completeness, Acyclicty, and Bipartiteness but it does not preserve Soundness. Simple proof argumentations are given in Appendix B.

#### 4.7.2.2 Collapse-Down

**Overview** Collapse down primitive abstracts away a designated activity from the graph by composing it with its downstream activities. Note that an activity can have multiple downstream activities. Unlike the collapse-up primitive in collapse down we create multiple composites. This is akin to the imaginary case where in the workflow description we do not have the activity to be collapsed, instead we have its copies (as many as the number of the original to-be-collapsed activity's successors in the original graph). Each such copy activity has dataflow links exclusively with one successor activity. The collapse-down primitive creates multiple composites by grouping each copy with its successor.

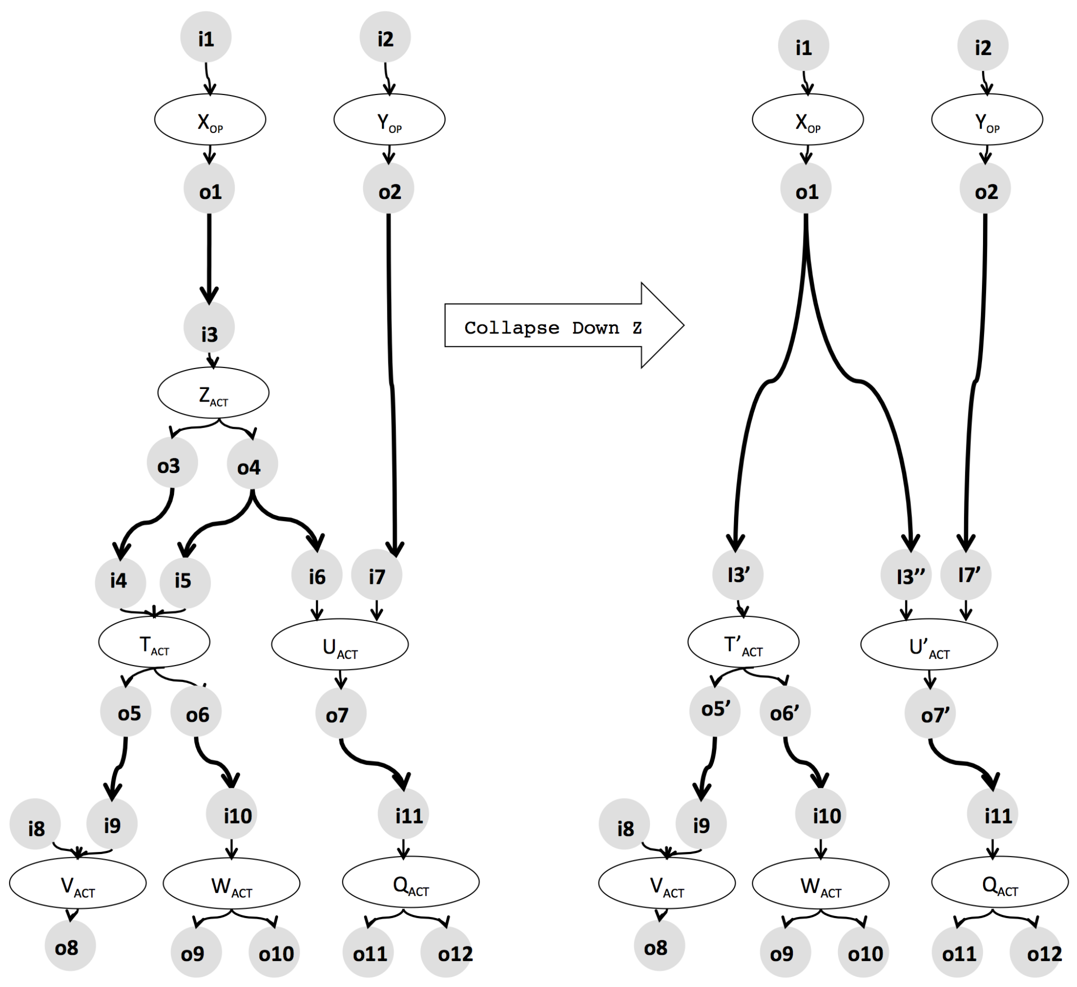


Figure 4.17: Before-After view of a sample application of the Collapse-Down primitive.

**Example** A particular application of the collapse down rule is given diagrammatically in Figure 4.17, where activity $Z$ is collapsed up onto activities $T$ and $U$. As can be seen from the figure, the primitive has created two composites: of $Z$ and $T$ as $T'$, and of $Z$ and $U$ as $U'$. The composition rule has differences when compared to the collapse-up primitive.

## Production

**Definition 4.8.** (Collapse-Down) We define the production$collapse_down : L \rightarrow R$ as one, which matches an activity with a designated motif attribute and composes it exclusively with all of its downstream processors. The GROOVE screenshot for this production is given in Figure 4.16. The reading for the production given in Figure 4.18

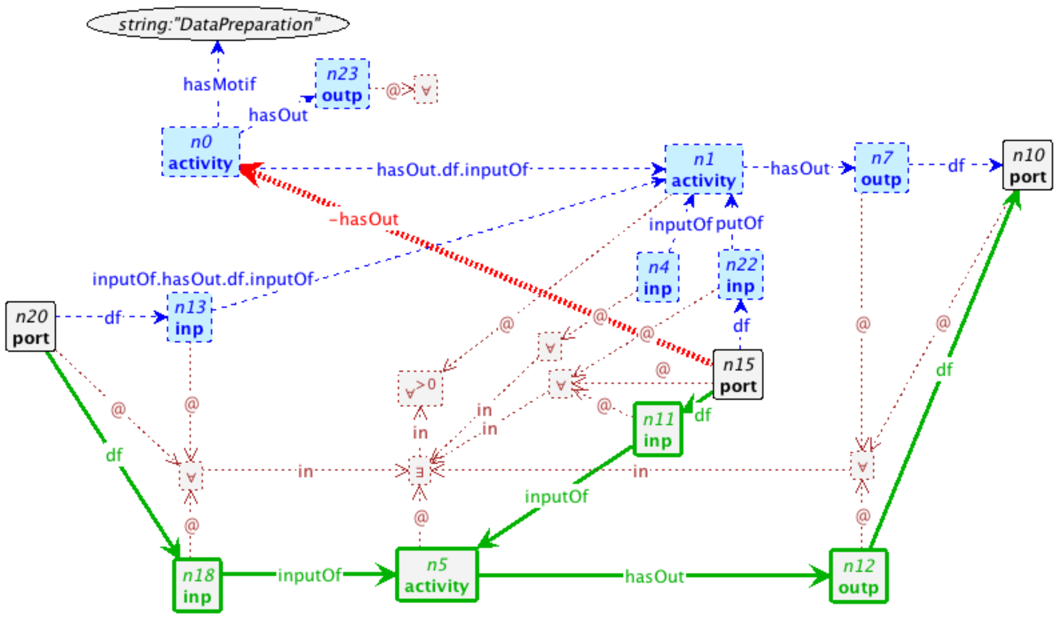


Figure 4.18: The encoding of the Collapse-down production in GROOVE graph grammar system

is as follows:

(∃) Exists a match for a to-be-collapsed activity that has a link to a datatype node with value "DataPreparation", and 2) a direct dataflow dependency to at least one follow-on successor activity, then the to-be-collapsed activity is to be deleted, the input port and the output ports of this activity are to be deleted.

(∀) For all input ports and outport nodes of the to-be-collapsed activity these port nodes are to be deleted.

($\forall^{>0}$) For all matching successor activity nodes these nodes are to be deleted.

(∃) For each such matched successor activity node a new composite activity is to be added.

(∀) For all input ports of the matched successor node these nodes are to be deleted.

(∀) For all input ports of the matched successor node with an incoming datalink from an external node (a node that is not the output of the to-be-collapsed) activity a new input port should be added to the composite activity.

(∀) For all output ports of the matched successor node with an outgoing dataflow link to some target port, a new output port should be added to the composite activity with a dataflow link to the same target port.

($\forall$) For all input ports of the to-be-collapsed activity with an incoming dataflow link from some source port, a new input port should be added to the composite activity with a dataflow link from the same source port.

Similar to Collapse Up, the Collapse-Down primitive preserves Validity, Completeness, Acyclicity, and Bipartiteness but it does not preserve Soundness. Simple proof argumentations are given in Appendix B.

To this end we have formally presented three primitives to abstract workflow description graphs. We will now describe our approach, named Workflow Summaries. In the following section we discuss the general characteristics of our approach, the inner workings of our abstraction machinery and relevant implementation details.

## 4.8 The Workflow Summaries Approach

### 4.8.1 General Characteristics

In this Section we outline the characteristics of our Workflow Summaries using the abstraction blueprint given in Section 4.4. The reader can follow our approach's characteristics and its comparison to related work in Table 4.1.

Our abstraction machinery operates over prospective provenance specifications, in other words **workflow descriptions** [BCG$^+$12]. This decision comes from observation[2] on what scientists primarily perceive as provenance. Scientists are fairly fluent in devising workflow descriptions [HWB$^+$12], and they have high familiarity with the conceptualisation (vocabulary) that workflow systems provide to represent their analytical processes. Meanwhile they perceive retrospective provenance, i.e. execution traces, often as a specialised form of technical metadata, that is at a lower-level of the scientific tooling. When compared to their familiarity to workflow descriptions scientists have very limited familiarity with retrospective provenance models. Note that the workflow abstractions generated by our productions are not intended to be executable workflows, but instead intended to be abstracted depictions of the analytical method or intended to be used as views over retrospective provenance traces.

[2]Based on interactions with Astronomers and Bioinformaticians participating in the EU Wf4Ever project.

Recalling the pre hoc vs post hoc strategies outlined in Section 4.2, we argue that our abstractions may find use for both cases. For the pre hoc case abstractions could be used as modularisation suggestions to scientists who develop workflows. For the post hoc case abstractions can be used as filters over lineage traces that are to be reported.

Our abstraction policies are in the form of an ordered list of $< attribute, primitive >$ pairs. Each such pair in the policy designates that **activities with the designated** $attribute$ **should be abstracted away from the workflow graph, using the mechanism of the designated** $primitive$.

- For the attributes we use the activity functional motifs described earlier in Chapter 3. Note that our abstraction machinery is not tied to motifs, and if desired so, other activity characterisations can be used. Note that using activity characteristics in the abstraction policy, means the policy is general, it is not tied to individual workflows, and it can be reused across workflows that bear the activity attributes (e.g. motif annotations).

- For the primitives we use the three declaratively specified graph transformation productions Eliminate, Collapse Up and Down. Primitives allow abstraction policies to be specified at a high level, while behind the scenes the system cascades this to lower-level graph manipulations while obeying certain integrity guarantees.

The *integrity policy* of our abstraction machinery is drawn from the primitives it supports. In our three primitives Completeness, Acyclicity and Validity aspects are un-compromised, whereas Bipartiteness (in Eliminate) and Soundness (in Collapse Up) can be compromised. In the scope of this dissertation we do not make a detailed assessment of which integrity policy may suit a particular provenance application scenario. However based on our observations on 1) existing experiment reports and 2) the manual abstractions encoded in workflow descriptions (discussed inline in Section 4.4.2), we hope our approach provides an appropriate selection of integrity guarantees in the context of experiment reporting.

### 4.8.2 Abstraction Procedure

In Section 4.6 we represented our abstraction primitives using declarative specifications, which allowed us to more intuitively discuss their integrity guarantees (Appendix

B). This exercise also made us aware of an important notion, called confluence, that characterises the behaviour of a graph transformation system in terms of its output. Confluence refers to a system's deterministic behaviour: meaning, that given the same input abstraction rules and same input graph, the system produces the same output graph.

A graph transformation system is considered not confluent when transformations (matched productions) are in conflict [TB94] with one another, henceforth the order of application affects the ultimate output. The most common case of conflicts is the "delete-use" conflict, where the RHS of one rule deletes the nodes matched by the LHS of another rule. Our declarative specification showed us that our three productions (due to their reductive nature) cause delete-use conflicts. A very simple example is shown in Figure 4.19. Let us assume that an abstraction machinery is configured with two rules that state activities with attribute (motif) $y$ can be abstracted either by collapsing up or down. When these rules are fed into a declarative graph transformation system with a simple input (start) graph as given in left of Figure 4.19, there will be two matches $m1$ and $m2$. The matches are in conflict as the application of one disables the application of the other. So if we choose to apply $m1$, then $m2$ is disabled and vice verse. And depending on choice the result abstractions would be different.

This conflicting nature of productions is interesting as it brings the possibility of alternative abstractions. Systems like GROOVE allow for the simulation of all possible application orders of rule matches (state-space exploration). Though the size of a state-space is exponential in the number of matches, it presents an opportunity to create and explore alternative workflow abstractions. We have left this study for future work, discussed in conclusions.

While we use the declarative specification for presenting primitives, in our implementation we realised them procedurally as a set of custom Java API methods. Each method contains an implementation of the graph production readings given in Section 4.7. This choice in implementation is an engineering decision to enable easier interoperation of our abstraction machinery with real-world models of provenance and associated provenance management and querying infrastructures. We will now describe the main procedure that coordinates the execution of individual productions with respect to a given abstraction policy. Following the procedure description we will revisit our discussion on confluence. Our abstraction machinery $M$ operates a brute-force procedure given in Algorithm 1.

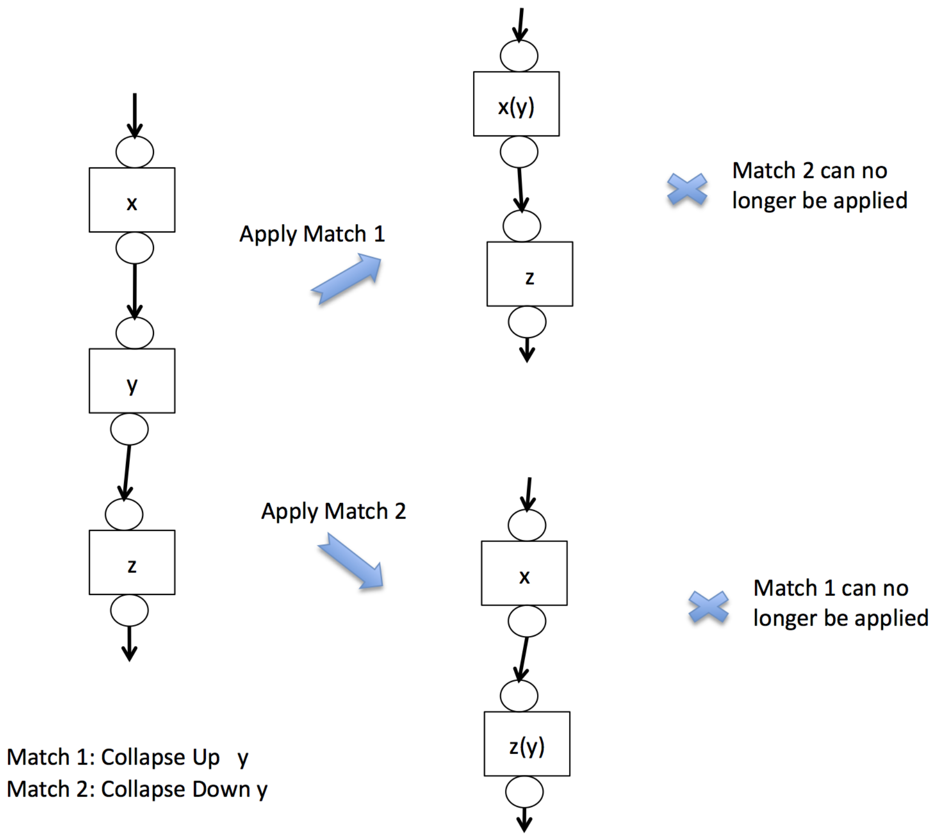


Figure 4.19: Two conflicting rule matches. And results of different application order.

- The *MainLoop* of $M$ receives as input 1) an abstraction policy and 2) a workflow graph to be abstracted. Within this procedure we iterate over the list of rule pairs ($< motif, primitive >$) in the abstraction policy. Per pair we call the *ExhaustRule* procedure to apply all matches of the rule to the workflow graph. We reiterate over the rule list, until there are no further changes made to the graph by the application of any rule in the list.

- The *ExhaustRule* procedure receives as input 1) the workflow graph being abstracted 2) the motif of a rule pair 3) the primitive of a rule pair. Similar to the main loop it adopts a brute-force approach. Recall that our abstraction policy is specified in terms of motif attributes, and does not refer to individual activities in the workflow graph. This procedure first obtains a materialised list of activities that are a potential match for the given rule by calling the $GetActivitiesWithMotif$ function. For each *activity* in this list we call the $MatchAndApplyProduction$ function, which performs the following: 1) obtains the true match for the rule, by checking for the existence of the LHS pattern for the designated graph primitive (as outlined in Section 4.7) 2) applies the match if exists and 3) returns true or false depending on whether a match has been applied. We continue iterating over the activities with matching motifs until we apply one match for that motif. Once a rule application (hence manipulation of the $workflowGraph$) occurs, then we refresh the list of activities with the designated motif and iterate over it to try to find another match. If we encounter the case where no application occurs for the list of activities, then it means this rule (motif-primitive pair) has been exhausted for the time being.

Due to its procedural implementation, our abstraction machinery, $M$, shows deterministic behaviour. To argue for this we need to establish that whenever inputted the same abstraction policy and same workflow graph our procedure creates the same unique order of $matchAndApplyProduction(wfGraph, primitive, activity)$ calls. As per the iterative nature of the procedure the factors that determine procedure call order are the order of (rule and activity) lists that are iterated on. We assume that the rule list in the abstraction policy is ordered. To ensure that the list of activities that match a given motif, i.e. the result of function $getActivitiesWithMotif(wfGraph, motif)$ function is the same for the same state of the $wfGraph$, we use ordering construct in the RDF Queries that underpin this function.

Finally we discuss how we handle motif information during abstraction. The behaviour of our primitives in this regard is as follows:

**Algorithm 1** Brute-Force procedures for applying an input abstraction policy to a workflow graph.

```
procedure SUMMARISER MAIN LOOP($abstractionPolicy, wfGraph$)
    do
        $changesMade \leftarrow false$
        for Each pair R in abstractionPolicy do
            $changesMadePerRule \leftarrow$ EXHAUSTRULE($wfGraph, R.motif, R.primitive$)
            if changesMadePerRule then
                $changesMade \leftarrow true$
            end if
        end for
    while $changesMade$
end procedure


procedure EXHAUSTRULE($wfGraph, motif, primitive$)
    $ruleApplied \leftarrow false$
    $ruleExhausted \leftarrow false$
    do
        $Activities \leftarrow$ GETACTIVITIESWITHMOTIF($wfGraph, motif$)
        $aMatchApplied \leftarrow false$
        for Each $activity$ in $Activities$ do
            $applied \leftarrow$ MATCHANDAPPLYPRODUCTION($wfGraph, primitive, activity$)
            if applied then
                $aMatchApplied \leftarrow true$
                $ruleApplied \leftarrow true$
                break
            end if
        end for
        $ruleExhausted \leftarrow \neg aMatchApplied$
    while $\neg ruleExhausted$
    return $ruleApplied$
end procedure
```

- Elimination simply removes activities from the graph, so motif handling is not required.
- For the collapse case new activities, that replace a binary group of activities, are introduced. Here the new activity inherits the motifs of both activities it replaces.

Our primitives are designed to abstract away designated activities in the workflow graph. The abstraction policy is therefore a specification of uninteresting activities. The others, i.e. those do not have the designated motifs, or do not have any motif information at all, are considered landmarks (significant steps). The abstraction procedure is therefore designed to abstract away uninteresting activities but keeps landmarks intact. As per the behaviour identified above, an insignificant activity may be collapsed onto a landmark activity, and the resulting activity would have the motifs of both. To prevent abstraction machinery from sweeping away this group, which involves a landmark, but also bears the motif of an uninteresting activity, we note this group as a landmark.

### 4.8.3 Implementation

We have used the Wfdesc [BZG$^+$15] ontology for representing workflows in an abstract technology independent manner. For testing we used the $Scufl$ to $Wfdesc$ converter utility [3] to convert Taverna workflows into Wfdesc representations. The outputs of abstraction are also specified in Wfdesc model. One of our primitives (Eliminate) introduces a modelling construct not directly available in Wfdesc, which is the indirect dataflow $iddf$. We have added this modelling element by extending the Wfdesc ontology. We have used the Jena library for storing Wfdesc RDFs. For querying we have used SPARQL [PS08] queries and for graph manipulation we have used the Jena API. The abstraction policies have been represented with a simple XML configuration file, example policy files used in our evaluation are given in Appendix C. The sources for the abstraction framework can be found in the code repository at [Alp15b].

## 4.9 Evaluation

We assess Workflow Summaries from two main perspectives:

[3]Developed by the Wf4Ever project [SRHB14].

1. We showcase and compare the capabilities of the Elimination and Collapse primitives in reducing structural complexity of workflow description graphs, as well as their capability in reducing the structural complexity of retrospective provenance views (in case the abstracted workflow used as a view over workflow execution traces).

2. We asses whether our approach of building abstraction policies on activity characteristics, where we capitalise on the frequently recited " adapters/shims are uninteresting" assumption, can be sustained against real-world abstractions of users.

In our evaluations we use two different workflow cohorts, and three different abstraction policies. We will now describe these before discussing individual assessments.

### 4.9.1 Workflow Cohort

We have used two different workflow cohorts:

- **Set 1:** These are a group of 27 Taverna-2 workflows. In terms of their number of activities these workflows are medium sized (An average of 14 activities per workflow ), and they are flat (contain no sub-workflows). All workflows in this set have narrative summaries in their landing page in the myExperiment workflow repository.

- **Set 2:** These are a group of 78 Taverna-2 workflows. This set corresponds to workflows slightly larger than those in Set-1 (Average 17 activities per workflow). Distinctively, these workflows have sub-workflows, and they are 1-level deep; meaning that the sub-workflows they contain do not contain any further sub-workflows. To obtain this set we have inquired the myExperiment repository for workflows that are 1-level deep and contains significant amount of abstraction, which we translated to the following filters: 1) at the top level (level-0) these workflows embody more than one sub-workflow 2) the number of activities found in sub-workflows (i.e. activities at level-1) are greater than or equal to the number of activities at level- 0.

All 105 workflows have been manually annotated with activity functional motifs using the Taverna' systems textual workflow annotation feature and a custom representation, which have programmatically converted into well-formed RDF based annotations as the ones exemplified in Section 3.7. Workflows in test set and all supplementary material for this chapter can be downloaded from the myExperiment pack [Alp15a].

### 4.9.2 Abstraction Policies

Abstraction policies are used to define behaviour for abstraction machinery. Policies are comprised of a motif component and a primitive component. With regards to the primitive, in the scope of this evaluation we have used policies based either entirely on Elimination or entirely on Collapse. We have not explored the effects and outcomes of using the combination of primitives (left for future work). With regards to the motif, we have targeted shim or adapter steps in workflows, which corresponds to those that have the *Data Movement*and *Data Preparation* motif (or sub-motifs) with respect to our motif ontology. We believe this policy is reasonable to be explored as a default policy as the insignificance of adapter steps have been underlined several times in literature [BCBDH08] [DZL11]. Moreover for the workflow Set-1, when we inspected the textual descriptions of 27 workflows, all descriptions referred to *DataRetrieval*, *Data Analysis* or *Data Visualisation* capabilities within workflows. Meanwhile the adapter functions were all transparent in the descriptions. Only in 4 out of 27 workflow descriptions were there one specific reference to adapter steps in workflows.

We used the following two policies for evaluation (respective configuration files are given in Appendix C):

- **Abstract-All-Adapters-By-Elimination**: This policy states that all activities with the
  *Data Preparation* or *Data Movement* motif, should be eliminated.

- **Abstract-All-Adapters-By-Collapse:** This policy states that all activities with the
  *Data Preparation* or *Data Movement* motif, should be collapsed either up or down. We establish this by providing rules first for collapsing up followed by rules for collapse down.

Additionally we use a third policy, named **Abstract-Selected-Adapters-By-Collapse**, to showcase that we can replicate the user's abstraction behaviour for certain classes of workflows. This policy is intended to abstract less aggressively, more mindfully based not only the functional characteristics of adapters but also the characteristics of their inputs and outputs. Here we encode a rule-of-thumb in the policy regarding the reporting-friendliness of data:

- Data that is stripped of resource specific padding is preferable. Consider as example the result of an analysis in raw web service XML response form, versus a form stripped of tags and padding. To implement this preference the policy states that $Augmentation$ steps should be collapsed down (input preferred), and $Extraction$ steps should be collapsed up (output preferred).
- Data with reduced cardinality is preferable. More specifically a singleton item instead of a collection is preferred, due collection being more structural complex. To implement this preference the policy states that $Flattening$ steps should be collapsed up (output preferred), and $Splitting$ steps should be collapsed down (input preferred).

## 4.9.3 Understanding Reductive Effect of Primitives

When presenting the Workflow Summaries approach we anticipated that workflow abstractions can have two possible uses: 1) to stand-in as simplified depictions of the analytical method or 2) to be used as views over execution provenance traces. To assess the reductive effect of primitives we measure the change in the following structural features:

- At the **workflow description** level we measure the decrease in the number of **Steps** and **Dataflow Links** ($df$ or $iddf$) among steps.
- When an abstracted workflow is used as a view over the **execution provenance**, an important feature is the reduction in the data derivation account(s) obtained by traversing this view. As an indicator of structural simplification we measure the reduction in the number of distinct **Data Artefacts** and distinct **Activities** that one would encounter when traversing the execution provenance using abstracted workflows. Workflows often have a particular dataflow path that represents the main branch of data processing, main path may be supported by several other auxiliary paths. In this assessment we considered the path in workflows with the largest number of significant activities as the main path. So, therefore we look at the reduction of elements in the data derivation account for the main path.

The percentage reduction in structural elements for workflows of Set-1 using the Collapse-All and Eliminate-All policies is given in Figure 4.20. A sample workflow from this set and the result of application of Eliminate-All and Collapse-All is given in Figures 4.22, 4.23, and 4.21 respectively. We can make the following observations on these results:

- The amount of abstraction that will occur is dependent on the occurrence of motifs that have been targeted in the abstraction policy. As our test cohort is comprised of Taverna workflows. It is unsurprising that reduction obtained in the number of steps (63 in Collapse, 68 in Eliminate in Figure 4.20) is close the rate of occurrence of adapter motifs in Taverna workflows.

- Elimination can abstract away steps at a slightly higher level than Collapse (68 vs 63). Cycle prevention prohibits certain collapse abstractions, the *adapterGroup*s in Figure 4.21 have remained separate groups due to cycle prevention, whereas in Eliminate no such preventions needed.

- The dataflow links are reduced at a percentage lower than that of steps (38 vs 63 for Collapse, 36 vs 68 for Eliminate). This is simply due to links that exclusively connect steps in workflows, account for only a part of all links. There are also those that link workflow inputs to steps and those remain intact during abstraction. Elimination can result in an increase in the number of dataflow links in cases where the product of incoming and outgoing links of an activity is greater than their sum (Recall from Section 4.7.1). Such a case would occur for activities that have multiple unlinks fanned-in and multiple out-links fanned out. We observed such an increase in a few abstractions in our results. However the small difference between the average link reduction among two strategies (i.e. 38 for Collapse, 36 for Eliminate ) show that activities with the fan-in & fan-out pattern are a rare sight in our test cohort.

- To understand what abstraction implies in terms of execution provenance we look at the number of processes and data on the main path (the account of derivation from output through to contributing input). The reduction in the number of steps in workflow description finds reflection as reduction of processes on derivation path (63% to 62% in Collapse, 68% to 68% in Eliminate in Figure 4.20). When we look at the reduction of data artefacts, we see that in Collapse process reduction implies data reduction (55%), but for Elimination, data reduction

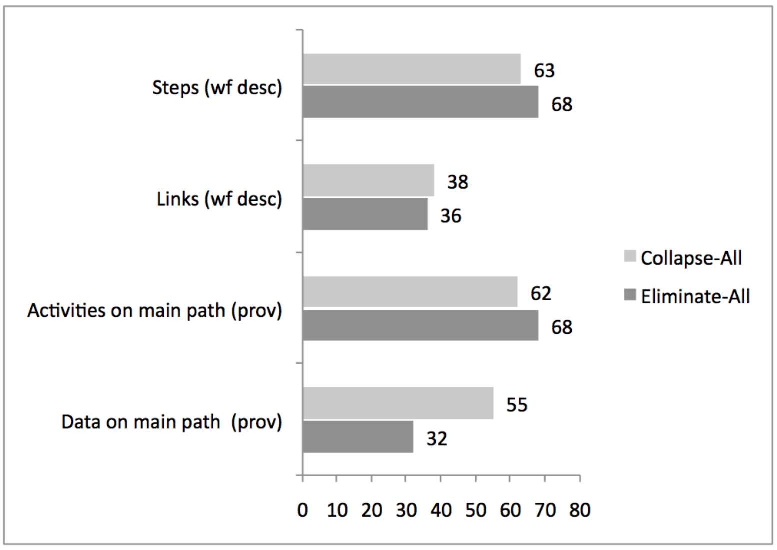


Figure 4.20: The structural reduction in provenance obtained by using the Eliminate All and Collapse All policies.

is rather limited (32%). This is due to the *iddf* data links. Inspect the abstraction created by Eliminate-All in Figure 4.23. Here the reduced (abstracted) account of derivation involves entirely *iddf* links. Each end of an *iddf* link corresponds to a distinct data artefact, because each end represents data input to and output from some abstracted computation. Therefore, the derivation path for *Found_genes_by_xref* involves 3 *iddf* links and 6 data artefacts. Whereas in the Collapse-All abstraction (Figure 4.21), there are *df*, direct dataflow links among activities. Each such link manifests as a single data artefact in execution provenance. Therefore, even though the derivation path is process-wise longer in Collapse-All abstraction it is more compact data wise. It contains 4 *df* links and therefore derivation contains 4 data artefacts.

### 4.9.4 Matching Against User Abstractions

We now assess how much our abstractions agree with the abstractions manually created by users (during workflow design). We use Set-2 (nested workflows) for this assessment. We take the user abstractions as ground truth: this involves the ports that the user has chosen to retain (keep visible at top level) in an abstraction, which corresponds to

1. output ports of activities at the level-0 (i.e. ports of activities that have been kept at top level, not pushed back to sub-workflows),

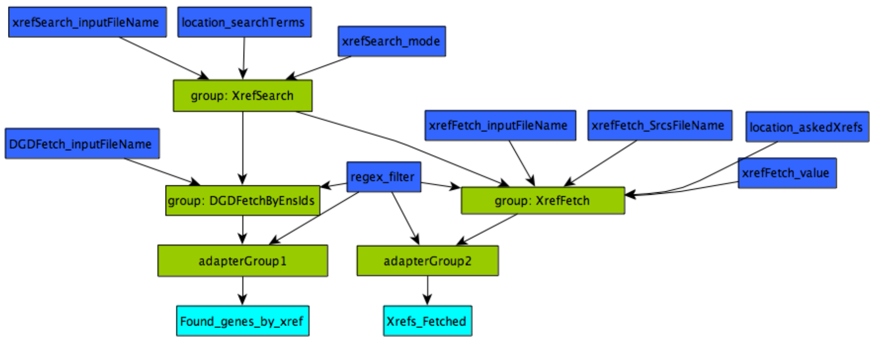


Figure 4.21: Abstraction generated by Collapse-All policy

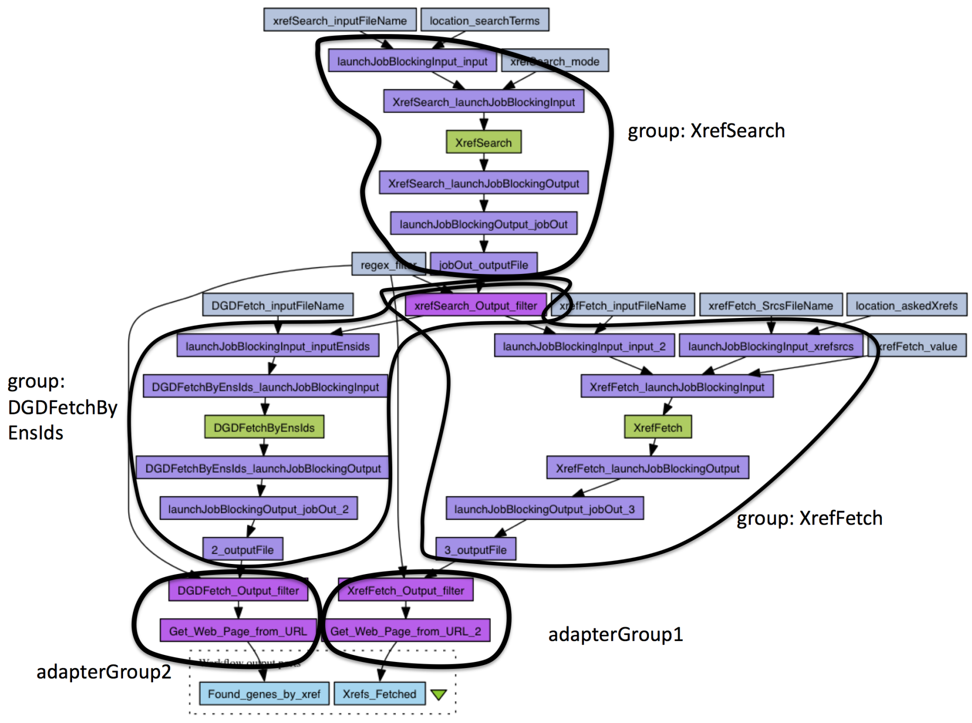


Figure 4.22: A sample workflow from Set-1 with abstractions of Collapse-All policy superimposed.

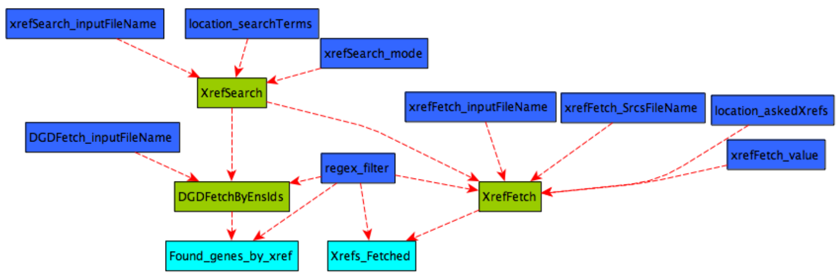


Figure 4.23: Abstraction generated by Eliminate-All policy

2. ports of activities in the sub-workflows that have been promoted from level-1 to level-0.

Against this ground-truth, the average F1 scores of abstraction policies for correctly retaining the ports that the user has retained is 0.36 for Eliminate-All, 0.60 for Collapse-All, and 0.70 for Collapse-Selected (given in Figures 4.26, 4.24 and 4.25 respectively). We can make the following observations on these results:

- Given that abstraction is a binary categorisation task determining whether an activity (and its ports) should be abstracted or retained, even a random categorisation would yield half (0.50) a correct answer. The low F1 score of Eliminate-All primitive shows us that, for a large number of workflows the way this abstraction primitive works represents an anti-pattern in reporting. More specifically that the (immediate) outputs of significant activities are not reported. A significant activity and its report worthy output may be separated by multiple activities in a workflow. Consider as an example the HTTD workflow (Figure 4.1). While the four web service calls are the activities with the significant *Data Analysis* or *Data Retrieval* motifs, in neither of these calls the output data is visible at the top level of workflow design. Instead these activities are grouped with *(Output) Extraction* type adapter that strip results from their service specific XML packaging. Therefore it is the output of these activities that gets reported.

- The Collapse primitive is based on the grouping of tasks, so unlike Eliminate-All, Collapse-All abstractions do not represent an anti pattern for most workflows. However abstractions in this category barely exceeds the expected minimum level of agreement, stands at 0.60. This shows us that partitioning the workflow description indiscriminately around significant steps does not yield the abstractions that scientists would make.

- As an alternative to Collapsing or Eliminating indiscriminately, in the Collapse-Selected policy we have taken influence from the scientists' abstraction behaviour we observed in workflows from Life Science domain (which account for 50% of workflows in Set-2). Here the policy targets specific adapter steps, and abstracts them away depending on the reporting-friendliness of their input and output. We observe that abstractions with this policy has significant overlap with user abstractions (see Figure 4.25).

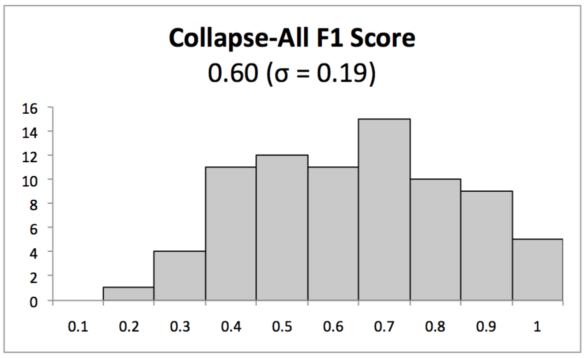


Figure 4.24: Mean F1 Score of Collapse-All Abstractions against User Abstractions, and its frequency distribution

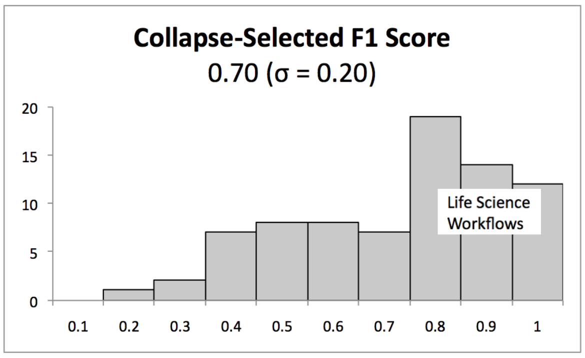


Figure 4.25: Mean F1 Score of Collapse-Selected Abstractions against User Abstractions, and its frequency distribution

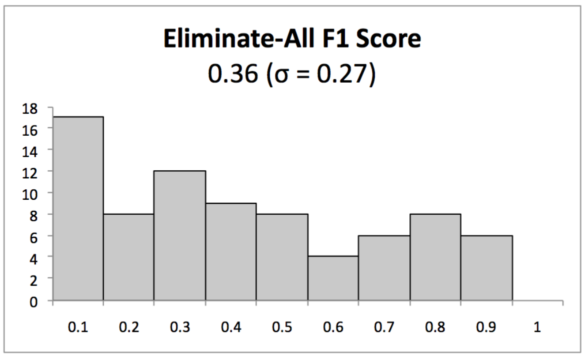


Figure 4.26: Mean F1 Score of Eliminate-All Abstractions against User Abstractions, and its frequency distribution

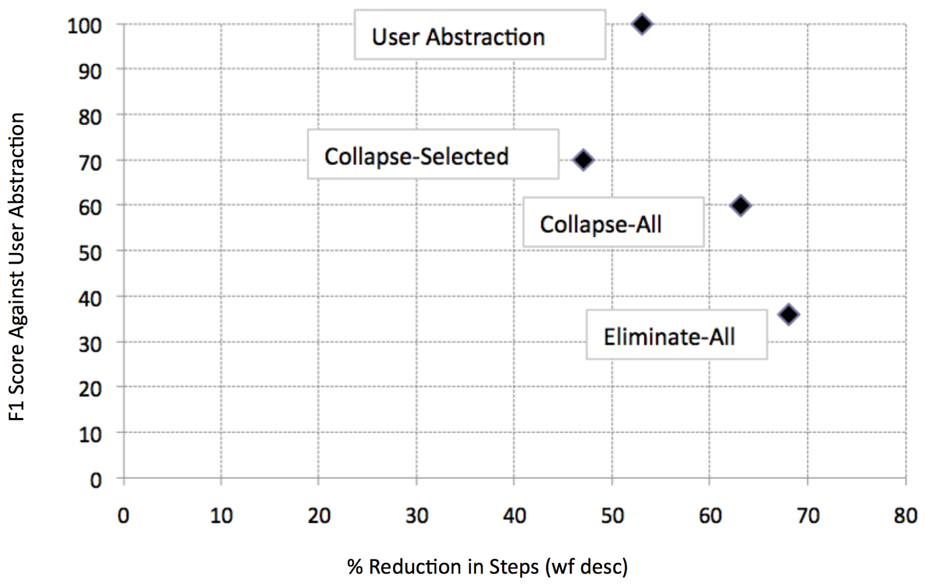


Figure 4.27: F1 Score versus Percentage Reduction in Number of Steps in Workflows, for all policies.

## 4.9.5 Discussion

Based on our observations from the evaluation we will now have a critical discussion on whether and how our abstraction approach may find use. Our primitives cater for two different means of abstraction.

**First is by retaining significant activities (and their input/output ports), while eliminating others.** An important finding from our evaluation was that in real-word workflows a report-worthy activity may not always be co-located with the reporting-friendly version of its input/output data. Because our Elimination approach operates under the such an assumption, while it can create workflow abstractions, these abstractions poorly overlaps with user abstractions. In fact early work on traversing or querying provenance traces have often operated with similar assumptions [MPB10] [ABML09], allowing as part of a view/filter definition, to denote either data or activity points of interest, but not separated activity and data elements in relation to one another. The ZOOM approach reviewed earlier is also affected by this reality of workflows, as it puts the boundary to a group right when it encounters a significant activity. The mismatch between our assumption in designing the Eliminate primitive and the realities of workflows reduces its applicability for reporting data, where abstraction is used as a view over execution provenance.

Despite restrictions, there can be room for using elimination based abstractions when sharing workflows in workflow repositories. Recall from Chapter 3 that a large number of workflows shared in repositories are flat, and these have limited narrative descriptions, within which only references to significant activities are made. Abstraction can find use as a post hoc simplification mechanism to generate simplified depictions of the experimental process, snippets [ESLF09], to go along with narrative descriptions.

**Second mechanism of abstraction is by grouping.** Our observation over existing workflows showed that grouping is a versatile mechanism as it not only simplifies the process by creating clusters around significant activities, but also acts as the contextual glue that links a significant activity with the reporting-friendly version(s) of its input and output data. In our approach we achieve the creation of groups with the collapse primitive. Note however that our policies do not specify top-down the groups and the activities that should go into groups. Instead we build groups bottom-up by giving a certain cohesiveness to adapter activities by associating with them a preference to be part of an upstream or downstream group. Larger groups then become a byproduct of repeated application of cohesiveness-based binary groupings. Our evaluation showed that using the collapse primitive we can abstract workflows, and abstraction can overlap significantly with user abstractions for certain cases, where the grouping decisions can be translated into an adapter cohesiveness based policy (the Life Science domain). There are obviously certain limitations with this approach

- Our means of grouping indirectly by sweeping adapters onto nearby landmark (significant) activities does not cater for several cases where groups can be based on the contextual relatedness of two (or more activities). An example is the $Stateful\ Invocation$ motif, where multiple adapter, or non-adapter activities may take place to serve one high-level function. Another example is groups comprised entirely of adapter steps, representing an adapter component. Our simplistic way of achieving groups would not cater for the creation of such groupings.

- While we demonstrated that simple grouping rules-of-thumb were encodable for Life Science workflows, we need to check whether encodings for other domains are possible. Our experience with workflows show that grouping preferences differ starkly among domains. While in web-service based life science workflows the datasets wrapped in XML padding are rarely reported, in the Astronomy domain the XML wrapping for the data is a standardised domain format called the VOTable format [OWD$^+$04], and this form of the data is preferred for other forms occurring within the same workflow.

Despite these limitations, we see use for collapse based abstractions as sub-workflow suggestions to scientists during workflow design.

In designing the Workflow Summaries approach we paid particular attention to having thin and mobile abstraction policies. We strive for a thin policy, not demanding too much input from user, because the effort that goes into designing a thick policy may as well be used to encode those abstractions into workflow designs. Similarly designing a fixed (immobile) policy, that refers to specific activities in workflow cannot be reused across multiple workflows. The cost of having thin policies is

- having imperfect abstractions, or those that do not overlap fully with what users would have devised. We saw that abstractions can (in cases) be improved by specialising the abstraction rules in the policy to denote how specialised adapter activities ($(Input)$ $Augmentation$, $(Output)$ $Extraction$) should be acted upon.

- the requirement to have attributes of activities, in our case motif annotations, to be able to refer to those attributes in the policy. For our evaluation we annotated the workflow cohort manually using the motif ontology. Overall effort amounted to 40 person-hours, corresponds to 0.5 person-hour annotation effort per workflow. Recall from the ENM workflow that the effort that goes into the design of workflows can be significant, when compared to design, the cost of annotation is minor. It is also possible to make annotations over description of activities in registries or module libraries and then propagate those annotations to activities' occurrences in workflows.

In our default policies and evaluation datasets we refer to Motifs to identify activities to be abstracted. Meanwhile our abstraction framework has no particular dependency on the Motif vocabulary. In case activities have characteristics (attributes) that can be referred to these can be used in the abstraction policies.

## 4.10 Chapter Conclusion

In this chapter we described our approach to tackle provenance complexity. We outlined the complexity problem in the context of experiment reporting, and identified the various categories of artefacts for which complexity can be defined. Among these categories we have chosen to tackle structural complexity of workflow descriptions through a graph transformation based approach. We provided a comparative review of the state of the art on computationally-assisted abstraction of provenance.

In our approach we view workflows as graphs, and abstraction as a graph transformation activity. Therefore we first provided formal background necessary to represent workflows as graphs and the workflow abstractions as algebraic graph productions. Using the formalism introduced, we described three primitives (productions) for abstracting workflows. We showed how primitives, as building blocks, can be used in conjunction with activity functional motifs to encode abstraction policies. We assessed the effectiveness of our approach through experimental evaluation using real-world workflows from the Taverna system. The future work will be discussed in Chapter 8.

# Chapter 5

# Provenance Driven Data Selection

## 5.1 Chapter Introduction

When outlining the Provenance Gap in Chapter 2 we observed that in order for workflow provenance to support reporting:

- the factorial-design of a computational scientific analysis gets reflected in workflow descriptions (prospective provenance) and execution provenance (retrospective provenance),
- both prospective and retrospective provenance is exploited to support result access, comparison [CFS$^+$06], and presentation [ABML09][BDKR09].

In Chapter 2 we identified *complexity*, *genericity* and *mixed-granularity* as characteristics that hamper provenance exploitation. We tackled complexity with work presented in earlier Chapters 3 and 4. Where we identified the cause and extent of complexity in Chapter 3 and provided abstraction techniques for workflow descriptions, which are commonly used as views when presenting provenance.

In this chapter we take the initial step in tackling *genericity* and *mixed-granularity* by identifying the how these characteristics hamper provenance exploitation for result access. More specifically we provide a concrete case of result access through querying of provenance, which we earlier identified as Provenance Driven Data Selection. Using this case we identify concrete problems in *mixed-granularity* and *genericity*, for which solutions techniques are provided in follow-on Chapters, 6 and 7 respectively.

We start in Section 5.2 by briefly recalling the Provenance Gap. In Section 5.3.3 we provide a case study using a real-world workflow, where we explore the utility of standard-provenance to answer provenance driven data selection queries. The case study shows us that workflows are of limited use for reporting results, in part because traces do not comprise domain-specific annotations needed for explaining results, and the granularity of provenance may not match up to the granularity of experimental parameters, which are important reporting anchors. The architectural overview of our solutions to these problems is presented in Section 5.4.

## 5.2 Motivation

When reporting their analyses scientists are expected to make available the datasets used and produced and provide information on the **context** and **source** of data.

- **Reporting origin** Emerging reporting guidelines state that data shall be deposited to respective repositories, and data reports (metadata tables describing data) shall have citations in them pointing to those repositories [nat15]. Moreover guidelines also state that if any data product from an investigation is a derivative of, or built using prior shared data then citations to those original datasets should also be provided [Dat11].

- **Reporting context** For the datasets produced in an investigation the data's context, i.e. the conditions under which it has been generated needs to be provided [fSDSC02]. This requirement involves identifying experimental parameters, configurations, input datasets, how parameters have been combined to configure the experiment [TFS$^+$08].

In our review of existing experiment reports in Chapter 2 we saw that these two high level requirements break-down into lower level requirements on how analyses are to be represented (factorial design), how analysis traces are to be used (provenance driven data selection), and the expected nature of descriptions of data (domain-specific information). We briefly recall challenges that workflow provenance poses in each category:

**Factorial Design** Scientific workflow systems provide constructs, with which factorial aspects of experiments can be encoded within a workflow. Our comparative review in Section 2.8.2 showed that workflow systems like Taverna provide high level of support in this regards, with the ability to 1) model data and parameters as collections not only in workflow descriptions, but also consistently so in workflow execution provenance 2) iteratively apply analysis over collections, 3) to configure iterations over multiple collections, thereby enabling the exploration of the data/parameter space created by multiple collections of parameters. The high-level of support and flexibility brings certain challenges. Systems like Taverna give users full control over factorial design, meanwhile it has no restrictions in place to check whether such a design is encoded end-to-end in a workflow. **Scientists may cause breaks in factorial design when they configure the workflow in a way that would cause an analysis activity consume at once multiple data generated with multiple points in the input parameter space.** In such cases it is no longer possible to differentiate result data products based on the parameters they descend from. A broken factorial design would manifest itself with the data-wise $n - by - m$ pattern in the execution provenance traces. Factorial designs may be compromised either.

- intentionally, when external analytical resources are coarsely integrated into workflows. Scientists may submit data from multiple parameters to an external resource all at once to reduce resource access cost.
- inadvertently, by errors in design.

A method to guarantee health of factorial designs could be being restrictive, an approach we observed in the Vistrails workflow system [CFS$^+$06]. Vistrails supports a restricted form of factorial design, by 1) outlining the parameter space in terms of a cross product of a pre-determined set of parameter lists, and 2) re-running the entire workflow with parameters from this space. This way Vistrails guarantees that all outputs generated from a particular run of a workflow are traceable to a pre-determined parameter combination.

**Provenance-Driven Data Selection (PDDS)** PDDS is used to select data subsets of interest for reporting. Given that constructs are in place for factorial design, and that analyses within the workflow are repeated per input/configuration then one can perform PDDS using workflow execution provenance. Data lineage is the key enabler of PDDS as it allows provenance consumers to trace experimental configurations/parameters through to corresponding results. PDDS implies the following access pattern over provenance graphs:

1. seek input parameter/data nodes of interest,
2. traverse lineage to reach outputs derived from selected inputs.

When illustrating workflow models with Wfdesc [BZG+15], and provenance models with PROV [BDG+12], we mentioned their **generic** nature. These models present a world view comprised of *processes* and their *ports* at the workflow description level; and similarly talk about *activities* and *data* at the execution provenance level. Given a workflow execution trace captured by a generic provenance model, Step 1 of PDDS needs to **differentiate between opaque data nodes in the graph**. To do this one requires attributes to predicate on, to define nodes of interest. The common way to refer to nodes is predicating on generic attributes such as names (e.g. names associated with port nodes in a workflow description or roles that qualify a data node's involvement with an activity in execution provenance) [ZSM+11] [SKS+08] [KDG+08]. Another way to pinpoint data nodes is to refer to data values. As discussed in Section 2.8.2, such value-based node identification is only possible when data values are stored jointly with provenance information, a data management pattern not adopted by all workflow systems. The success of Step 2 of PDDS depends on the health of the encoding of factorial designs, ensuring discrete traceability from parameters to respective results.

**Domain-specific information** In Chapter 2 we observed that domain specific information is a quintessential part of an experiment report. We observed that scientists supplied such information either using structured annotations (e.g. ontologies, vocabularies as in ISA-Tab example) or using unstructured annotations (e.g. column and cell descriptions in the ENM example). On the workflow provenance side, explicit domain specific information is by default absent. Workflow provenance models are generic and aim to capture the computational process that has led to the derivation of an artefact. These models pose no obligation on documenting what the domain specific characteristics of the activities and the data are. Therefore the common mechanism of obtaining these description is by manual annotation. Also note that domain specific information is typically available implicitly in (above described) generic attributes like names and roles, or found in data values.

In the following section we will illustrate a PDDS scenario with a real-world workflow and sample queries. Our aim in giving this example is to

- provide a glimpse into capabilities of Taverna system for factorial design
- illustrate a case of broken factorial design and show how this affects the precision of answers to PDDS queries

- motivate the need for domain-specific descriptors for data, which in their absence are substituted by generic attributes, data values, and structural patterns in provenance.

## 5.3 Case Study

### 5.3.1 Factorial Design in Taverna

Figure 5.1 illustrates a simplified version of a workflow from Astronomy [Exp12] that takes as input a set of galaxy names (*list_cig_name*), and outputs extinction/reddening calculations per galaxy (*data_internal_extinction*). The workflow starts by retrieving data, including coordinates, for each galaxy through a service based lookup from the Sesame astronomical database (Step-1- *SesameXML*). Coordinates are used to query the Visier Database to retrieve further data regarding galaxies (Step-2- *VII_237*). Galaxy morphology information is extracted from the Visier results, which is input together with coordinates into a local tool that computes galaxy extinction values (Step-3-*calculate_internal_extinction*). The scientifically significant activities in this workflow are the data retrievals and the local extinction calculation. The remaining activities are adapters, which perform (*Output*) *Extraction*, *Format Transformation* and *Data Moving*. In Figure 5.2 we provide an illustration of the execution of this workflow with two input galaxies.

Following from the two figures, we can summarise the factorial design constructs provided by Taverna as follows:

- Factorial design depends on the ability to define a parameter space for the experiment. In our example the input list of galaxy names defines such a space. The intent is to repeat the analysis per input galaxy. Our example also illustrates the case where the parameter space gets expanded during the execution of the workflow. From Figure 5.1 we see that the output of the coordinate retrieval operation (Step-1) is used as a parameter for the follow-on activities, the retrieval of galaxy information (Step-2) and the calculation of Galaxy extinction values (Step-3). The coordinate retrieval operation (Step-1) expands the parameter space comprised of galaxy names, into a space of galaxy coordinates.

- Taverna implements parameter spaces by supporting structured types, more specifically nested collections for data. In the workflow description (Figure 5.1) the

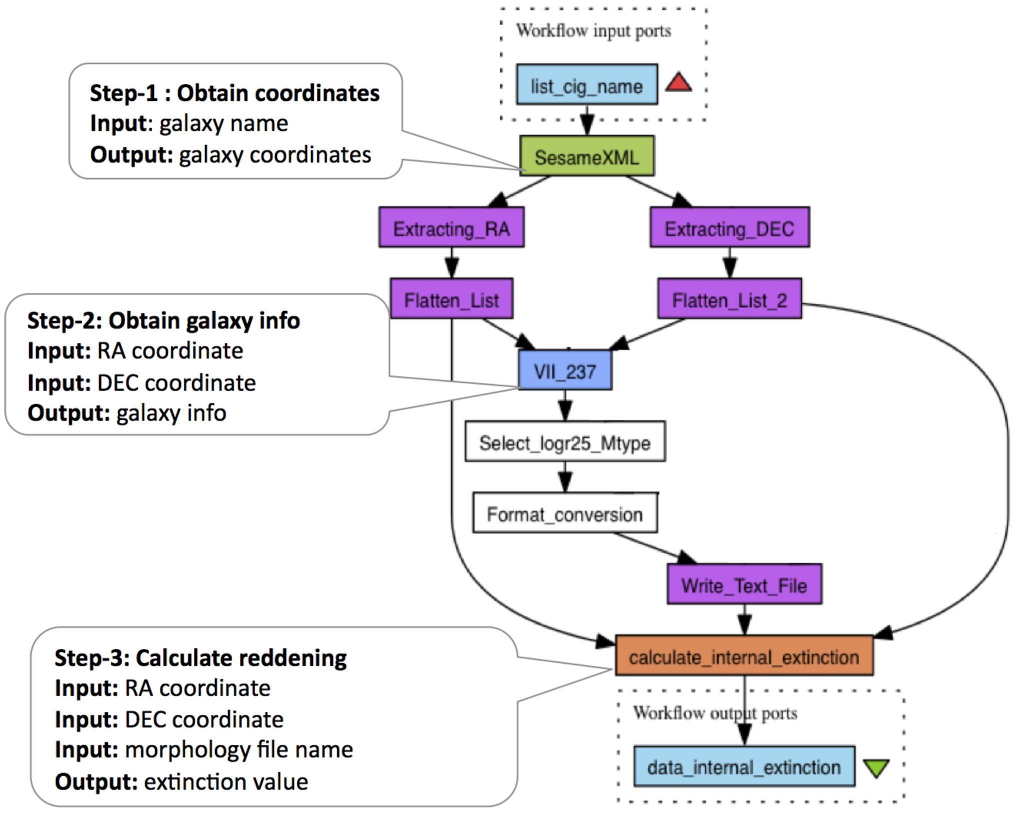


Figure 5.1: Sample workflow from Astronomy developed by the Wf4Ever project.

workflow input *list_cig_name* is defined to be a *List* of depth 1. Note that this fine grained modelling is consistently reflected to workflow execution traces, meaning that for a particular execution of the workflow the actual input list of galaxy names will be a *Collection* type entity, and it will have member entities representing each input galaxy name (see input list comprised of strings $M31$ and $M33$ in Figure 5.2).

- Taverna allows the exploration of the parameter/data space (identified by input lists) through its configurable iteration capability. Iteration (or repeated application of analyses) is achieved by exposing an analysis operation with input that is of bigger cardinality than its expected input. The *SesameXML* activity in Figure 5.2 represent the simplest case of iteration. By definition *SesameXML* processor accepts and returns singleton items. But in the execution illustrated it is exposed to an input collection with two galaxy names. This is called *Iterated Composition* in Taverna, and as per its execution semantics the analysis activity will be iteratively applied to each item in the input list resulting in an output list with two galaxy information strings. A more complex case of iteration is in case when an activity accepts input from multiple input ports, then, how iteration will occur over these lists can be configured via *Dot product* and *Cross product* configurations. This is illustrated with the *VII_237* processor, which by definition accepts two singleton inputs (coordinates) and returns a singleton output (galaxy info) (Figure 5.1). The iteration configuration for this processor states that in case it encounters lists at its ports rather than singletons it should find its input pairs by creating *dot* product of input lists, consequently in our example run (Figure 5.2) we see that this activity consumes same indexed items from each input list.

Let us now look at how factorial design is broken by inspecting Figure 5.2. The scientist intends to repeat the analysis for each input galaxy, and in turn for each coordinate associated with that galaxy. So, by adopting iteration, the workflow is able to obtain galaxy coordinates in an XML form (*SesameXML* step), and then extract galaxy coordinates per XML file (*Extracting_RA* and *Extracting_DEC* steps). Up until this point we are able to trace every data artefact back to the corresponding galaxy name parameter. At the *Flatten_List* step however, factorial design breaks. At this step all coordinates for all galaxies are bundled into a single list via a single invocation of the *Flatten_List* step. By inspecting the follow-on structure of execution we understand that the workflow designer has inadvertently created this design mishap. Because we observe the follow on Visier DB lookup step (*VII_237*) is iterated over the bundled up list of coordinates. So if the designer iterated the Visier DB lookup over the nested lists instead of the flattened list, we would be able to tell which output(s) of Visier lookup correspond to which input galaxy name.

Note that the occurrence of broken factorial design in execution traces is not just observable in Taverna provenance. As identified in our survey of workflow systems in Chapter 2, Kepler COMAD and Wings workflow systems provide constructs for factorial design (i.e. modelling of parameter/data collections and iterated analyses), meanwhile, they provide no restrictions on how those constructs are put together. As a similar patterns may occur in traces from these systems.

### 5.3.2 Provenance Queries for Data Reporting

Provenance Driven Data Selection (PDDS) corresponds to exploitation of provenance for the exploration of result space of a workflow-based analyses. The "seek node and traverse lineage" pattern of PDDS identified in Section 5.2 uses provenance as a scaffold to reach results of interest. Provenance applications exemplified in Chapter 2 such as the result comparison feature of Vistrails [CFS+06], or the Provenance Browser of Kepler [ABL10] all rely on provenance query scenarios involving the pattern of PDDS.

In order to obtain a set queries representative of the PDDS pattern we consulted the provenance literature. The work of Zhao et al [ZSM+11] and the Provenance Challenge [MLA+08] stood out as systematic query collections.

- Zhao et al [ZSM+11] enriches workflow provenance by 1) semantic annotations regarding domain-types of data and activities and 2) external community-accumulated information on scientific subjects (e.g. a gene, a galaxy, a species) for which data analysis has occurred. Authors provide seven queries; Four of them are focused on reaching out to external information available as Linked Open Data (LOD) for the life science community. The remainder three are focused on querying workflow traces; more specifically they exploit lineage to reach results that are obtained for certain scientific subjects and or the origin of data/parameters that have been supplied from external repositories through a *Data Retrieval* step. We have made an adaptation of these latter group of queries when building our case study queries.

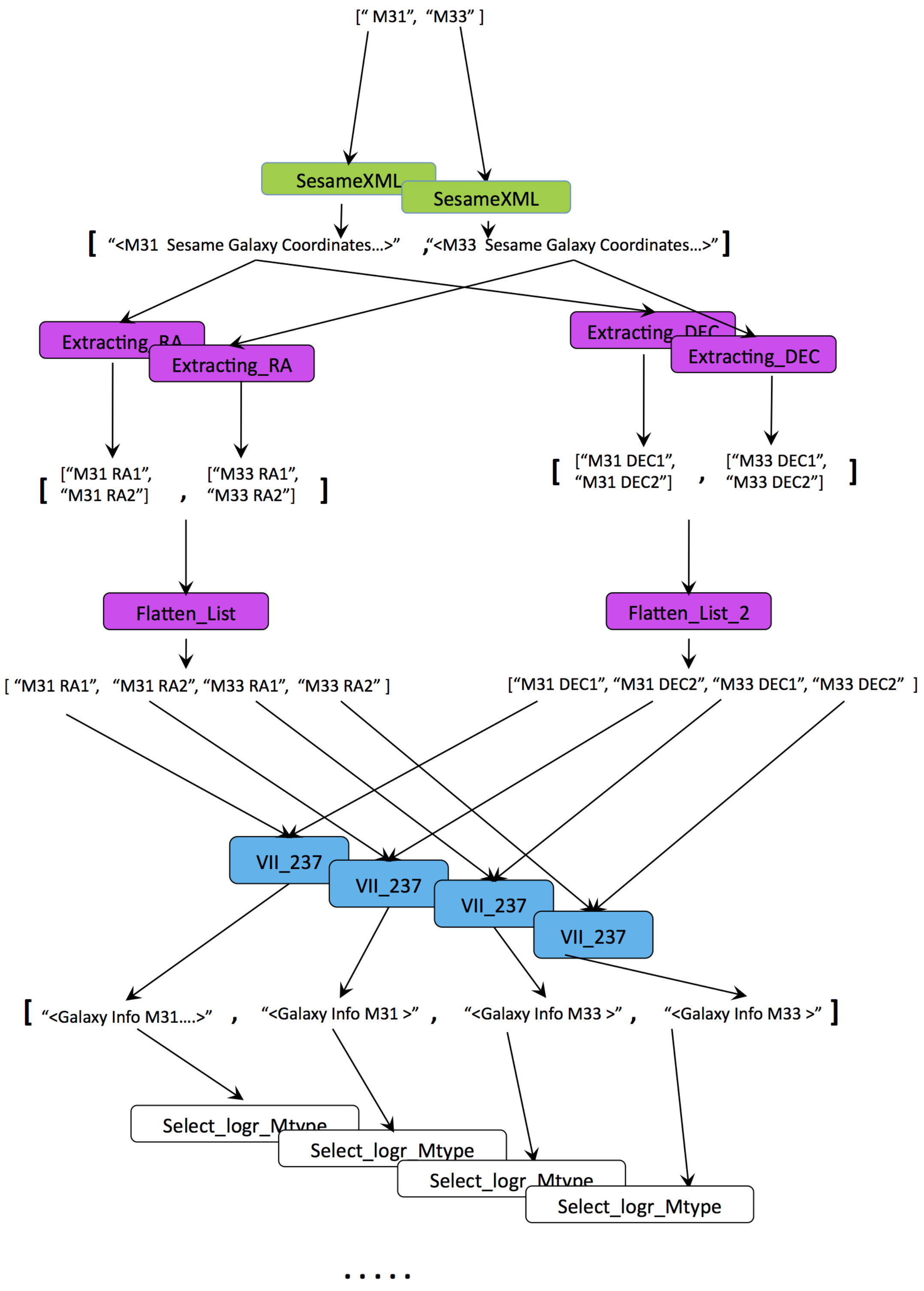

[" M31", "M33" ]
SesameXML
SesameXML
[ "<M31 Sesame Galaxy Coordinates...>" ,"<M33 Sesame Galaxy Coordinates...>" ]
Extracting_RA
Extracting_RA
Extracting_DEC
Extracting_DEC
[ ["M31 RA1", "M31 RA2"] , ["M33 RA1", "M33 RA2"] ]
[ ["M31 DEC1", "M31 DEC2"] , ["M33 DEC1", "M33 DEC2"] ]
Flatten_List
Flatten_List_2
[ "M31 RA1", "M31 RA2", "M33 RA1", "M33 RA2" ]
["M31 DEC1", "M31 DEC2", "M33 DEC1", "M33 DEC2" ]
VII_237
VII_237
VII_237
VII_237
[ "<Galaxy Info M31....>" , "<Galaxy Info M31 >" , "<Galaxy Info M33 >" , "<Galaxy Info M33 >" ]
Select_logr_Mtype
Select_logr_Mtype
Select_logr_Mtype
Select_logr_Mtype
.....

Figure 5.2: A fragment of execution illustrated for the Astronomy workflow.

- Provenance Challenge [MLA+08] is a joint effort of computer scientists to establish a case that can be used to understand the capabilities of different provenance systems. The case involves a Medical Imaging workflow and nine queries. Three of those are focused on process and seeks process structure and details from provenance. Two queries focus on joint querying of user-added annotations and provenance, another one focuses on workflow-evolution and involves provenance graph comparison. Finally three of them focus on data. Similar to Zhao's queries these focus on selecting results based on the parameters they are derived from. In devising our queries we have made adaptation from these queries (particularly Query #6) of provenance challenge.

We illustrate PDDS with three queries. Q1, Q2 are adapted from [ZSM+11], and Q3 is an adaptation of from of challenge Query #6 [MLA+08].

| |
|---|
| Q.1 Which of the results are coordinates obtained from the Sesame database, from which source catalogs are the coordinates obtained. |
| Q.2 Select all results that have been obtained with inputs belonging to the Andromeda Galaxy. |
| Q.3 Select extinction calculation results that are obtained with inputs belonging to the Andromeda Galaxy, where the morphology parameter setting was 0.45. |

Table 5.1: Sample Queries for Provenance Driven Data Selection

We will now analyse if and how these queries can be implemented with PDDS and their effectivity. To measure effectivity we use **Contextual-Precision**, which we define as:

$$\frac{\#\, of\ \textbf{Contextually-Accurate}\ results}{Total\ \#\ of results}$$

We define **Contextual Accuracy** as the results actually belonging to the scope implied by the query (e.g. for Q1 the results that actually contain data that is retrieved from the Sesame database, or for Q2 the results that do indeed belong to galaxy M31).

We refer to our realisations of queries with the "-P" suffix denoting that they have been implemented over PROV-compliant traces obtained from the Taverna system.

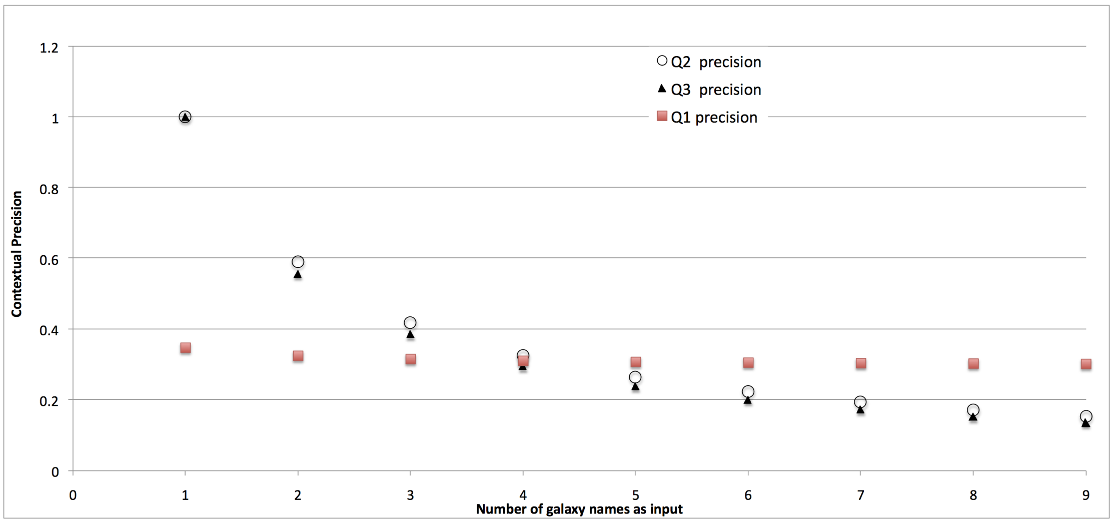


Figure 5.3: Contextual-Precision of provenance driven data selection queries.

These are generic traces capturing activity invocations, data artefacts and the usage and generation relations among data and activities (as exemplified in Section 2.6.3). Used traces do not contain any explicit domain specific characteristic of data or activities.

**Q1-P** This query is in two parts; first part seeks data by its origin second part seeks detailed information on the origin. We can realise this query partially:

- for the first part the intent is to obtain data that represents **a result from the Sesame database or its local copies generated by adapter steps**. Here we use lineage as a pseudo (replacement) mechanism to denote data origin. We seek data artefacts, whose derivation path includes the activity named *SesameXML*, which we know accesses the Sesame database. Roughly one third of the results whose derivation path includes a Sesame DB lookup actually contain data that is retrieved from Sesame (See Q1 precision in Figure 5.3). Remaining two thirds of results are those that are **computed through analyses by using the data obtained from Sesame**. As workflow activities are observed as black-boxes by provenance collection framework we have opaque lineage that tells us there is some influence relation among data artefacts but falls short of differentiating between:

  - a derivation relation based on value-copying.
  - a derivation relation based on any other analytical computation

  Designating origin via path based linkage to an element identified in workflow design is weakly precise, yet it is robust to increases in the workflow inputs. Increasing the number of galaxies in a workflow run does not alter the fact that only one thirds of outputs are copies of data from the Sesame Database.

- the source catalog information sought in the second part of query is available within some of the data values, i.e. in the XML output of the *SesameXML* activities, there is a field that specifies the catalog (the relevant subpart in Sesame DB) that the result comes from. However when these results are stripped of their XML padding the catalog information is no longer associated with retrieved coordinates (outputs of the $Extract\_RA$ and $Extract\_DEC$ steps). Therefore this part of the query cannot be implemented.

**Q2-P** Given that the workflow designer has attempted to parameterise the workflow with a list of galaxy names, this query seeks to find all (intermediary and final) results that descend from one parameter, i.e. $M31$, Andromeda Galaxy. We realise this query by seeking the input data node with value $M31$ and then traversing the execution trace to find all downstream data products that are derived from it. Figure 5.3 shows the contextual precision for this query. In response to this query, for data generated post $Flatten\_List$, we would get results obtained for all galaxies rather than just $M31$, as all results are linked to $M31$ via a single invocation of $Flatten\_List$. Due to broken factorial design provenance (lineage) becomes a weakly precise index for selecting data when the workflow is ran with increasing number of parameters.

**Q3-P** This query seeks results of a particular activity, extinction calculation. This activity takes 3 input parameters: the two parts of a coordinate and a path to the file containing the morphology information for the galaxy. This query can be realised only partially. The predicate stating that morphology parameter should be 0.45 requires access to morphology parameter's values. On the other hand, the extinction calculation activity accepts as input a configuration parameter file name, rather than the actual parameter value. This value is available in outputs of upstream adapter steps, or in the output of extinction calculation. However within these data values, the morphology information is present in tab delimited texts where multiple numeric values are present. This prohibits to build a mechanism to systematically predicate over a fragment of those texts. Therefore we implement this query partially, without the morphology value predicate, only seeking extinction values computed for a particular galaxy's coordinates. As extinction calculation activity accepts inputs, which are no longer accurately traceable to input galaxy names(post $Flatten\_List$), the resulting query precision is as equally bad as Q2-P (see Figure 5.3).

### 5.3.3 Outcomes of Case Study

Our illustrative case for PDDS highlights the following issues:

**Correct implementation of factorial design is key for PDDS being possible.** The way scientists design Taverna workflows determines whether factorial design will be successfully reflected in execution provenance. Queries that seek results descending from a particular input parameter require discrete reachability between designated parameters and results that depend on those parameters. When factorial design is broken, this renders provenance minimally useful for certain types of queries in PDDS. A crucial requirement, then, is the ability to anticipate breaks in factorial design by inspecting workflow descriptions and prevent them (before they occur). Results from such a check can be provided as feedback to workflow designers in order for them to understand whether they will later be able to use provenance for data selection. For instance, for the astronomy workflow the feedback would be "The results of this workflow are not discretely traceable to the **galaxy name input parameter**. Discreteness is broken at the $Flatten\_List$ step". Such feedback could be useful in re-factoring workflows by users (where possible).

**Domain specific information is needed for PDDS.** We realised PDDS queries with the assumption of them operating on generic provenance. In order to find nodes of interest over such a provenance graph, we are forced to put selective criteria on **data values** (Q2-P and Q3-P), or in **attributes** like name (Q1-P). This approach proved to have the following disadvantages:

- Most workflow systems including Taverna, adopt **a separated storage scheme** for data and provenance. As a result it is not possible to seamlessly implement PDDS queries. In fact for queries Q2-P and Q3-P, as a precursor, we identified which data node in the provenance graph corresponds to the $M31$galaxy by first scanning through the data values stored in the file system.
- The selective criteria in our queries contains predicates over domain-specific information implicitly available in data values, and attributes. Our implementations of these queries can therefore be considered **adhoc**, as there are no structure or vocabulary restrictions on query criteria targets. The informativeness of a name for a workflow port or activity is at the disposal of workflow designer. Names can be freely given and the same activity (e.g. Sesame database lookup)

can have different names across workflows. Similarly in Q3-P we were unable to implement morphology criteria part of query as the data values were not self-describing and structured enough to allow a systematic implementation of this criteria.

An alternative to using generic provenance is to first annotate provenance with domain specific information and then place query predicates over those annotations. The Taverna system, and related tooling [HDZ$^+$14], provides means to annotate workflows. Using this annotation feature we can identify characteristics of workflow elements. For instance using semantics annotations and an ontology from the respective domain we can state that a processor is an instance of a Sesame Database lookup. These design-bound characteristics can be propagated to workflow provenance, each invocation of that activity would then be considered an object of the same semantic type. While such an approach would help systematise Q1-P, it would not help with Q2-P or Q3-P. In these queries, we are interested in dynamic characteristics of data nodes that get determined at runtime. For instance, each input data node containing a galaxy name is an object of the same semantic type, yet each such node contains information about a different galaxy. In order to unearth these dynamic characteristics we would have to annotate elements in the provenance trace. Unlike annotating workflow descriptions [MSZ$^+$10] [WKM$^+$10], annotating workflow provenance quickly ceases to be a manageably sized task when workflows are parameterised with input parameters. For our example, for a single galaxy, a total of 17 final results are generated. The number of outputs increases linearly with the number of input parameters. For a list of 6 galaxy names supplied as input, we get 20+ extinction result , the number is 100+ when considering intermediary results as well. This illustrates how workflows as automation tools proliferate data generation and makes apparent that any approach for annotating provenance would require automation support.

**Qualified lineage is needed for PDDS.** One of our queries (Q1-P) was seeking data based on its origin. In our implementation we represented this with a query where we sought nodes that have some lineage relation to designated origin node. Our answers to this query was partly correct. This is because of we were using opaque lineage relations, a typical result of black-box workflow provenance. These relations are inform us that there is some influence relation among data artefacts, but do not capture the specific nature of relations. On the other hand query requires lineage in a stronger (more specific) sense, it seeks those data artefacts that descend from an origin by means of a particular derivation, i.e. value-copying.

In the following section we introduce the approach we take in addressing requirements derived from the case study.

## 5.4 Our Approach

We aim to improve the conditions that hamper PDDS by providing:

1. A mechanism to detect breaks in factorial design. The technique we provide is specific to Taverna workflows. We capitalise on two aspects 1) that Taverna has well-defined execution behaviour [MPB10], 2) that there are no discrepancies in the level of granularity of modelling processes and data within a Taverna workflow description and its execution provenance. These two allow us to devise rules that can anticipate the structure of workflow provenance by analysing the workflow description. We encode these rules in a deductive database environment, and use them to determine whether multiple lineage traces originating from parameters will be joined up at any point in the workflow's execution.

2. The *Label*Flow framework for creating explicit domain-specific metadata, which is tailored to provide hooks for data selection, thereby replacing ad hoc means ( selecting nodes by referring to implicit information.) We adopt a semi-automated approach which as input expects functions that encapsulate the ability to make explicit the domain specific information embedded in data values. We adopt simple attribute-value pairs, we call Labels to capture explicit domain specific information. We provide a generic set of Labelling Operators that perform the middle-man duty of transferring the execution trace of an activity to its associated labelling function, and then transferring labels obtained from the labelling function back on to provenance. Our operators are generic in the sense that they are oblivious to the content of Labels they carry around, and they can operate over PROV compliant standard workflow provenance traces. Our approach maximises the coverage of labels by propagating them where possible. We take inspiration from "Where- Provenance" in databases where lineage based on value-copying brings the possibility of propagating attributes of a data to its copied descendants [BCTV04]. In particular, we exploit the $Motif$ characterisation of workflow activities (described earlier in Chapter 3) to determine lineage relations that are based on value-copying and for such cases we perform label

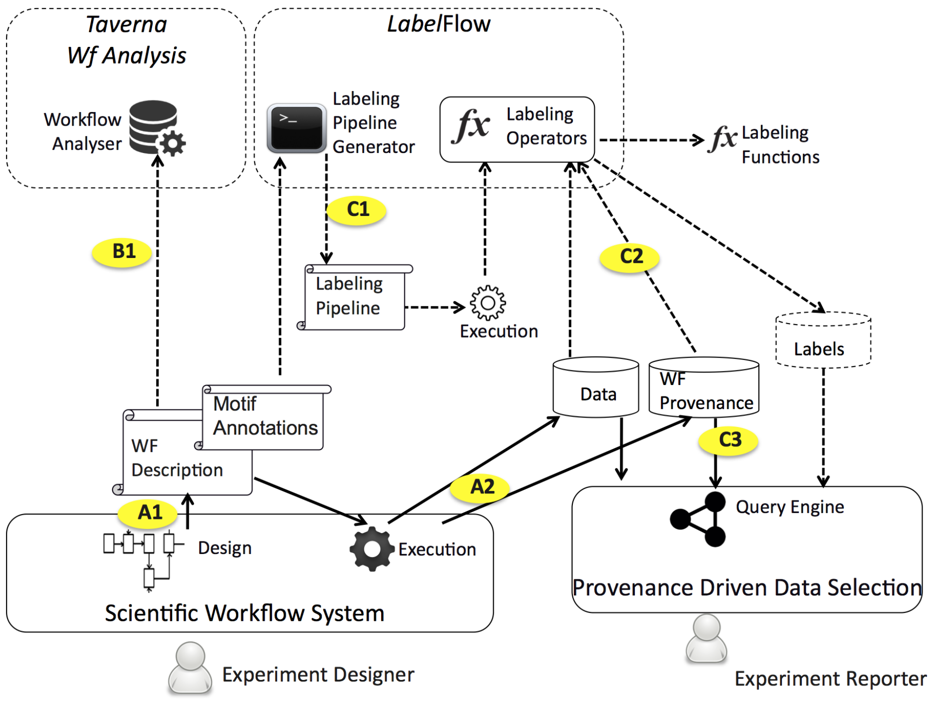


Figure 5.4: Labelling System Architecture.

propagation. We exploit the structure of the scientific workflow to induce simple processes that coordinate the execution of labelling operators associated with activities in the scientific workflow.

A common characteristic of both of the above solutions is that they are non-intrusive to the existing practices of workflow development and existing architecture of provenance capture and access. Figure 5.4 provides the overall architecture of our approach. The first part of our solution is workflow analysis, where we process, through rules, Taverna workflow descriptions to compute the anticipated provenance structure that will come out of its execution, and the health of factorial design (Step B1 in Figure 5.4). The second part is labelling with *Label*Flow. We undertake labelling as an offline process, where we do not interfere with the established process of scientific workflow design (Step A1) and execution (Step A2). Workflow runs result in the generation of data artefacts and generic workflow provenance. These two make up our primary sources of information for obtaining labels.

We bundle all label generation/propagation capability for a scientific workflow into a corresponding **Labelling Pipeline**. Using 1) workflow descriptions that contain labelling specifications over analytical steps (processors) and a 2) repository of labelling functions we generate a labelling pipeline for a given scientific workflow (Step C1).

This pipeline can in-turn be used to decorate all execution traces coming out of that workflow with labels (Step C2). Once labels are generated they can be used in conjunction with generic workflow provenance to support PDDS (Step C3).

## 5.5 Chapter Conclusion

In this chapter we presented a case study on provenance driven data selection (PDDS). This case highlighted two particular problems in using workflow provenance for reporting data:

- first was the negative impact of broken factorial design (in Taverna workflows) to PDDS queries. We illustrated that broken factorial design corresponds to the case where not all activities in the workflow are repeated for all the points in the input parameter space of the workflow. We showed that the provenance traces from these workflows are a poor index to be used to select result datasets, because lineage traces do not discretely link workflow parameters to resulting data, instead they get joined up.

- second was the need to refer to domain specific characteristics of parameters/data to be able to refer to nodes of interest in a provenance graph within a PDDS query. We illustrated that in the absence of domain specific characteristics as it is the case with standard workflow provenance, we would have to place selective criteria on data values, which:
  - for certain workflow systems, including Taverna, are stored separately from provenance, therefore not co-queryable with provenance.
  - expresses characteristics implicitly, therefore is an adhoc mechanism to define data selection criteria.

In response to these problems we introduced two solutions:

- A **workflow static analysis** approach that aims to detect breaks in factorial design.

- A framework to facilitate the systematic and fine-grained **annotation of workflow provenance** traces.

These two solutions are described in the follow on chapters, Chapter 6 presents our workflow analysis approach, and Chapter 7 presents provenance annotation.

# Chapter 6

# Workflow Analysis

## 6.1 Chapter Introduction

In this chapter we describe our static analysis technique aimed at detecting broken factorial design in Taverna workflows. Earlier we informally defined Broken Factorial Design as the existence of an activity record in provenance, where the activity consumes input descending from multiple parameters. As a prerequisite to formally represent this pattern we start in Section 6.2 by introducing how collections drive iteration in Taverna:

- In Section 6.2.1 we illustrate the footprint of iterated activity executions in provenance, and show how well-defined behaviour of Taverna in iteration provides us with rules to anticipate data characteristics and lineage among collections of data in a prospective fashion, without executing the workflow. We outline these as depth prediction and depth mapping rules.

- In Section 6.2.2 we introduce Taverna's *Dot* and *Cross* product operators and show how they are used to build up input parameter spaces from multiple collections of inputs.

- In Section 6.2.3 we illustrate that the break in factorial design corresponds to a discontinuation of Depth Mappings.

In Section 6.3 we break down Taverna's execution behaviour into a set of known computations and we give the functional specifications for each. We also specify what the depth mappings are for each functional specification.

We use Datalog to implement our rules. In Section 6.4 we provide an overview of our solution approach and Datalog rule modules. From Section 6.4.3 through to Section 6.4.6 we outline each module and illustrate its operation over a running example. We discuss our implementation in Section 6.5.

In Section 6.6 we review related work that aims to improve provenance through analysis of workflows or source code. In Section 6.7 we discuss how the outputs of our analysis can be utilised. We conclude in Section 6.8

## 6.2 Modeling Taverna Workflows

Throughout this section we build on the formal representation provided by Missier et al [MPB10] for representing Taverna workflows and their execution behaviour. As stated earlier Taverna allows for structuring of data artefacts as nested collections. More specifically;

**Definition 6.1.** (Taverna type designators) $\mathcal{T}_{name}$ denotes the set of possible type designators for data processed in Taverna workflows. $\mathcal{T}_{name}$ is comprised of derivations with the following syntax rule:

$$\tau ::= s|L(\tau) \quad where;$$

$s$ denotes the basic *string* type and $L(\tau)$ denotes list types. $s$, $L(s)$, $L(L(s)) \in \mathcal{T}_{name}$ are example type designators. We denote nesting of lists with a superscript numbered shorthand; $L(L(s))$ is denoted as $L^2(s)$. We use $\mathcal{T}$ to denote the set of data values that conform to any type in $\mathcal{T}_{name}$.

**Definition 6.2.** (Taverna workflow) is denoted with the triple $w = \langle PRO, POR, LINK \rangle$ and the functions *in*, *out*, *src* and *snk* where;
*PRO* is the set of processor names,
*POR* is the set of port names,
*LINK* is the set of dataflow links among ports,
$in : PRO \rightarrow 2^{POR \times \mathbb{N} \times \mathcal{T}_{name}}$ and $out : PRO \rightarrow 2^{POR \times \mathcal{T}_{name}}$ are two interface functions that maps processors to their ordered inputs and to their outputs with type designators. Each port within a signature of a processor has a unique name. For a particular a input/output $port \in POR$ of a processor $proc \in PRO$ we use the combination of $proc.port$ to refer

to it.
$src : LINK \rightarrow PRO \times POR$ and $snk : LINK \rightarrow PRO \times POR$ are two functions that map links to their source and sink ports.
For the purposes of this chapter we extend the core workflow model with the following functions:
$procFun : PRO \rightarrow S$, where $S \subseteq \mathcal{T}$ denotes the set of strings, is a function that maps processors to their underlying functions. Multiple processors in a workflow may be underpinned by the same function (recall multiple list fattening processors in the case study workflow).
$lhb : PRO \rightarrow E$ is a list handling behaviour function which maps processors to their corresponding *iteration strategy expressions* obeying the following syntactic rule:

$$\varepsilon ::= (\varepsilon \otimes \varepsilon) | (\varepsilon \odot \varepsilon) | \langle portname \rangle$$

Expressions can be specified using binary *Cross* ($\otimes$) and *Dot* ($\odot$) product operators and port names. Each port appears once in the expression and expressions can be comprised of subexpressions. Iteration strategy expressions encode crucial information on how a processor is to be executed with multiple input collections.

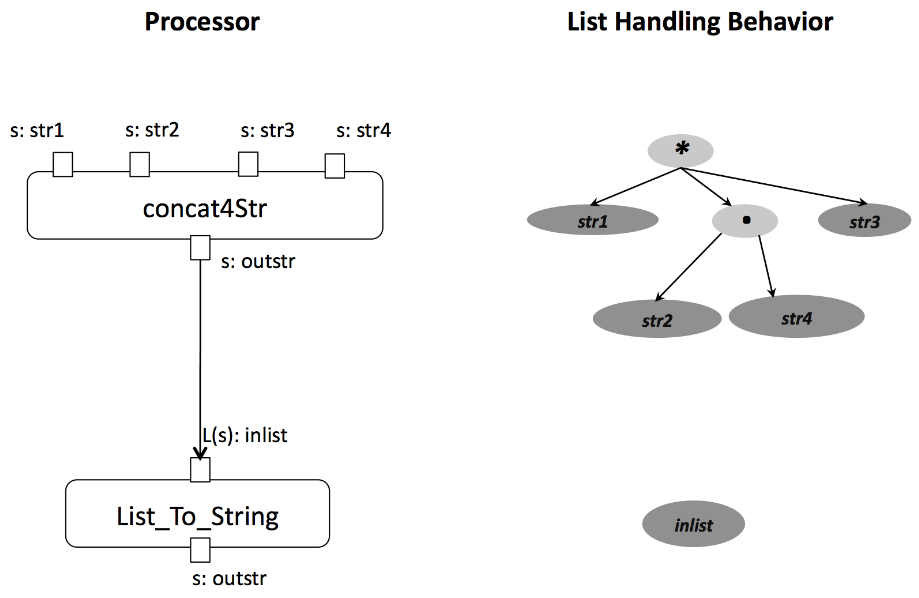


Figure 6.1: Two sample Taverna processors, their port types and list handling behaviour.

**Example 6.3.** (Workflow) In the left hand side of Figure 6.1 we depict two sample processors with their defined input and output types. The *concat4Str* processor accepts four inputs eachof type $s$, and produces one output of type $s$. The *List_To_String* processor accepts an input of type $L(s)$ and returns $s$ created by coalescing all items in the input list. The specification of the workflow fragment in Figure 6.1 as per Definition 6.2 is as follows:

$$\begin{aligned}
PRO = & \quad \{concat4Str, List\_To\_String\} \\
POR = & \quad \{str1, str2, str3, str4, outstr, inlist\} \\
LINK = & \quad \{l1\} \\
\\
in(concat4Str) = & \quad \{\langle str1, 1, s\rangle, \langle str2, 2, s\rangle, \langle str3, 3, s\rangle, \langle str4, 4, s\rangle\} \\
out(concat4Str) = & \quad \{\langle outstr, s\rangle\} \\
lhb(concat4Str) = & \quad \{(str1 \otimes (str2 \odot str4)) \otimes str3\} \\
\\
in(List\_To\_String) = & \quad \{\langle inlist, 1, L(s)\rangle\} \\
out(List\_To\_String) = & \quad \{\langle outstr, s\rangle\} \\
lhb(List\_To\_String) = & \quad (str1) \\
\\
src(l1) = & \quad \{concat4Str, outstr\} \\
snk(l1) = & \quad \{List\_To\_String, inlist\}
\end{aligned}$$

In right hand side of Figure 6.1 and throughout the rest of the chapter we denote processor's *iteration strategy expressions* with an intuitive diagrammatic view using a List Handling Behaviour (LHB) formula tree. Note that the tree-based representation in Figure 6.1 is compact in the sense that operators are not binary, they have multiple parameters. This compact form can be expanded into a binary-operator-only tree through addition of nesting in a left associative manner.

Dataflow links among processors is the mechanism of processor composition in Taverna. At the design interface Taverna allows the composition of processors with mismatching nesting-levels of input output types. At the workflow specification backend Taverna automatically infers the dataflow adjustments required for a successful execution of the workflow. These adjustments can be informally described as follows:

- Simple Composition: This is the straightforward case where the input of a processor is of expected depth expected by the processor, requiring no adjustment.
- Wrapped Composition: Represents the case when the data supplied to a processor is of a lesser depth than expected. In this case Taverna infers that a *wrapping* dataflow adjustment is necessary, and prepends as many outer lists as necessary around a processor's input port so that during execution the inputs will be of depth acceptable by the processor.

- Iterated Composition: Represents the case where input supplied is of depth greater than expected, here Taverna requires information in the LHB to determine how to consume inputs. There are two sub cases of iterated composition.
  - Single-Input Iterated Composition: This is the case where the processor has a single input. The LHB for the *List_To_String* operation in Figure 6.1 is an example. In this case Taverna is prescribed to traverse down the input list structure until the it encounters an item that is of depth acceptable by the processor.
  - Multi-Input Iterated Composition: When a processor accepts multiple collections as inputs, Taverna gives the possibility of configuring selection of input combinations from lists with *Dot* and *Cross* operators. The LHB for the $concat4Str$ processor is an example.

In iteration because the processor is exposed to inputs with depth greater than expected, the output(s) from the iterated execution of processor will also be of depth greater than the defined depth of processor's outputs. The impact of depth differences of inputs to the output depends whether iteration is a single or multi input one, and also depends on the structure of the LHB formula. When the predicted depth of a port is greater than its defined depth this creates a ripple effect for the downstream portions of the workflow. Based on Taverna execution semantics Missier et al [MPB10] have provided formulas and an algorithmic approach to make predictions on port depths for a limited case of processors, which has a restricted kind of LHB formula. We highlight differences between our work and Missier's work in the next section after discussing Taverna iteration behaviour in detail.

**Example 6.4.** (Processor Composition) In Figure 6.2 we illustrate how the processors given earlier in Figure 6.1 are composed as part of a larger workflow. The workflow has 4 input ports which are connected with dataflows to inputs of the $concat4Str$ processor. Note that three of workflow's input ports are of type $L(s)$, as a result dataflows from these ports to $concat4Str$ correspond to iterated composition, with adjustment consequences for the output of this processor. The amount of depth adjustment for the output of $concat4Str$ (the calculation of which we will show later) is 2. Hence while a single invocation of this processor results in a single string (as by its definition), in its composed occurrence in the workflow the overall (iterated) execution of this processor will result in a list of list of strings, $L^2(s)$ at output port $outstr$, designated with

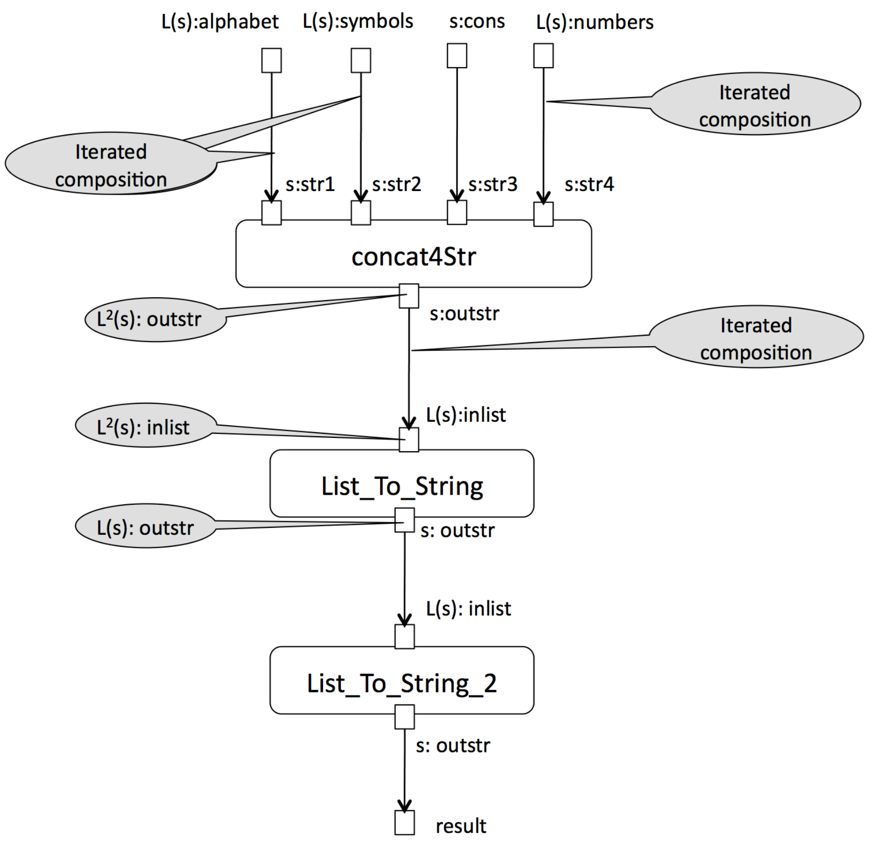


Figure 6.2: Depth adjustments and iterated compositions inferred for a workflow description.

call-out in Figure 6.2. As a ripple effect the iteration of $concat4Str$ will cause iteration of $List\_To\_String$, which will be exposed to an input of depth $L^2(s)$, greater than its defined depth $L(s)$. As a result of iteration $List\_To\_String$ will generate an output of type $L(s)$ greater than its defined type $L(s)$. Note here that as a result of iteration of $List\_To\_String$, its downstream processor $List\_To\_String\_2$ will encounter an input that matches exactly its defined input type, henceforth the very last step in the workflow, $List\_To\_String\_2$ will not be iterated and will be executed just once.

We will now present an abstract model for representing information generated through workflow analysis. We will outline iteration behaviour of Taverna workflows, and how anticipations of iteration and data generated from iterations are represented in this abstract model. We will also outline how the well-defined execution behaviour Taverna allows us to have rules that can be used to populate our abstract model.

### 6.2.1 Footprint of Iteration in Provenance

**Definition 6.5.** (Predicted Provenance Model) Analysis based predicted provenance $PP$ for a workflow $w$ is denoted is with the tuple $PP = \langle w, D, M, R \rangle$ where;
$D$ represent relations outlining iteration characteristics of processors and predictions on depths of processor input/outputs.
$M$ represent relations on predicted patterns in provenance.
$R$ denotes a set of Taverna workflow analysis rules that populate relations in $D$ and $M$ using relations in $w$, $D$ and $M$.

**Definition 6.6.** *(Depth Predictions Model)* The information model $D$ for representing depth predictions is comprised of the following relations:
$definedDepth : \langle PRO \times POR \rangle \rightarrow \mathbb{N}$ is a function that maps a processor port to the depth-wise size of the data structure that it is defined to consume/produce as per its type designator. Defined depth is a numerical characterisation of type designators of ports. Given $\langle p, t, n \rangle \in definedDepth$ we denote this with $dDep(p.t) = n$.
$predictedDepth : \langle PRO \times POR \rangle \rightarrow \mathbb{N}$ is a function that maps a processor's port to the depth-wise size of the data structure that the port is predicted to hold during workflow execution. Given $\langle p, t, n \rangle \in predictedDepth$ we denote this with $pDep(p.t) = n$.
$\Delta Depth : \langle PRO \times POR \rangle \rightarrow \mathbb{N}$ is a function that maps a processor's port to the difference (delta) between the predicted and defined depths. Given $\langle p, t, n \rangle \in \Delta Depth$ we denote this with $\Delta Dep(p.t) = n$.
$\Delta Proc : PRO \rightarrow \mathbb{N}$ is a function that designates whether a processor is predicted to iterate or not. Given $\langle p, v \rangle \in \Delta Proc$, $v = 0$ denotes that $p$ is predicted to execute once, $v > 0$ denotes that $p$ is predicted to iterate. We denote this tuple with $\Delta Proc(p) = v$.
$\Delta Link : LINK \rightarrow \mathbb{Z}$ is a function that designates the kind of composition that a dataflow link represents. Given $\langle l, v \rangle \in \Delta Link$, $v = 0$ denotes simple, $v < 0$ denotes iterated, and $v > 0$ denotes wrapped composition. We denote this tuple with $\Delta Link(l) = v$.

**Example 6.7.** Consider processor $List\_To\_String \in PRO$ the interface definition of which were given earlier in Example 6.3. Depth predictions for $List\_To\_String$ due to its composition with other processors in the workflow were illustrated in Figure 6.2. Using Definition 6.13 a subset of predictions (the computation of which we will discuss later) is denoted as follows:

$$
\begin{aligned}
dDep(List\_To\_String.inlist) &= 1 \\
pDep(List\_To\_String.inlist) &= 2 \\
\Delta Dep(List\_To\_String.inlist) &= 1
\end{aligned}
$$

We will now move on to illustrating how the exposition of Taverna's execution behaviour allows for the definition of rules to populate *PP*. We denote Taverna execution behaviour (as exemplified in Equation 6.1) using the notation initially given by Missier et al [MPB10]. Our adopted notation has the following characteristics:

- Execution behaviour is given in terms of recursive functions that are comprised of a conditional body. The conditions are comprised of 1) a base case and 2) a recursive case.
- Conditional test expressions of each case are given with the infix notation.
- Body of each case is a function call given with the prefix notation similar to most functional programming languages.
- The recursive case involves the use of *map* function, well-known from functional programming languages. For any other utility function used in the body we provide inline definitions.

Taverna achieves iteration by encapsulating processor functions within a recursive evaluator.

**Definition 6.8.** (*eval* function) $eval : \mathbb{N} \times S \times \mathcal{T} \to \mathcal{T}$ is a function that applies a designated processor function over a given input data value. The specification of *eval* is as follows:

$$
(eval_l \quad f \quad v) = \begin{cases} (map \quad (eval_l \quad f) \quad v) & |v| > l \\ (f \quad v) & |v| = l \end{cases} \tag{6.1}
$$

*eval* uses the following utility function in its conditional expressions:

**Definition 6.9.** ($|\,|$) *depthwiseSize* : $\mathcal{T} \to \mathbb{N}$ is a function that returns the depth-wise size of a data value that is of type $\mathcal{T}$. We denote this function with "$|\,|$" as a shorthand. Example applications would be, $|\,[\text{“}a\text{”},\text{“}b\text{”},\text{“}c\text{”}]\,| = 1$, $|\,[[\text{“}x\text{”},\text{“}y\text{”},],[\text{“}z\text{”},\text{“}t\text{”}]]\,| = 2$, $|\,\text{“}a\text{”}\,| = 0$.

When executing a single input processor $p \in PRO$ with input $i \in POR$ Taverna instantiates the *eval* with $l$, $f$, and $v$, where $l = definedDepth(\langle p,i \rangle)$, $f = procFun(p)$ and $v$ is the data value appearing at port $i$. *eval* traverses down the input list structure until it encounters an item that is of a depth that is acceptable by the processor, given with $l$. A sample execution of the *List_To_String* processor with *eval* would be as follows:

$$\begin{aligned}
(eval_1\ \mathit{List\_To\_String}\ [[\text{“11”},\text{“12”}],[\text{“21”},\text{“22”}],[\text{“31”},\text{“32”}]]) &= \\
(map(eval_1\ \mathit{List\_To\_String})\ [[\text{“11”},\text{“12”}],[\text{“21”},\text{“22”}],[\text{“31”},\text{“32”}]]) &= \\
[(eval_1\ \mathit{List\_To\_String}\ [\text{“11”},\text{“12”}]),(eval_1\ \mathit{List\_To\_String}\ [\text{“21”},\text{“22”}]),(eval_1\ \mathit{List\_To\_String}[\text{“31”},\text{“32”}])] &= \\
[(\mathit{List\_To\_String}\ [\text{“11”},\text{“12”}]),(\mathit{List\_To\_String}\ [\text{“21”},\text{“22”}]),(\mathit{List\_To\_String}\ [\text{“31”},\text{“32”}])] &= \\
[\text{“11} - \text{12”},\text{“21} - \text{22”},\text{“31} - \text{32”}] &=
\end{aligned}$$

During execution each invocation of the processor function (the base case of *eval* in Equation 6.1 ) fires an event, which gets caught by Taverna's provenance framework and gets logged. For the iterated *List_To_String* processor, there will be three such event logs, that record explicitly the lineage between the inputs and outputs of the processor function (depicted with solid lines in Figure 6.3). Note that Taverna does not explicitly record lineage among output lists created in response to traversing down input lists (the recursive case of eval in Equation 6.1). In Figure 6.3 we illustrate explicitly recorded lineage with solid lines and implicit lineage (only among lists) with dashed lines.

**Observation on *eval*** The exposition of how a processor is executed (Equation 6.1) and how iteration occurs allows us to anticipate the structural characteristics of data and lineage before a workflow gets executed. Because the evaluator is recursive and exploits the *map* function, this gives us a rule of thumb stating that:

a. for each enclosing list structure that we travel down (to find inputs that are of depth acceptable by a processor) a corresponding output list will be generated. As a result each extra (delta) depth for the input will become an extra (delta) depth for the output.

b. each enclosing output list (generated due to a corresponding one in input), will have the same size (number of items) as the corresponding input list.

We will now provide the formal encoding of these rules. Throughout Sections 6.2 and 6.3 we denote rules with implications in the form of "**if**... **then** ..." statements where each statement is a Horn clause (a conjunction of literals implying a single literal) [AHV95]. Each literal represents information in relations of Predicted Provenance Model (Definition 6.5). In Section 6.4 we present how these rules are implemented in Datalog.

**Definition 6.10.** (Delta Depth Calculation Rules) Given a port $r$ of processor $p$ the following apply:

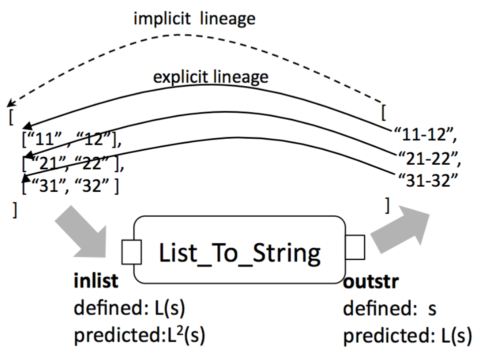


Figure 6.3: Explicit and Implicit provenance in an iterated processor execution.

**If** $dDep(p.r) = n$, $pDep(p.r) = m$ **then** $\Delta Dep(p.r) = m - n$
**If** $dDep(p.r) = n$, $\Delta Dep(p.r) = d$ **then** $pDep(p.r) = d + n$.

The delta depth of a port is the difference between its predicted and defined depths.

The delta depth calculated for the input port of a processor can be used to predict whether the processor will be iterated or not and also to predict the processor's output depth.

**Definition 6.11.** (Depth Prediction Rules - Single Input Processor) For a single input processor $p$, where $in(p) = \{\langle i, 1, t\rangle\}$ executed with *eval* the following apply:

**If** $\Delta Dep(p.i) = n$ **then** $\Delta Proc(p) = n$.

The delta depth of the single input determines whether the processor will be iterated or not.

**Definition 6.12.** (Depth Prediction Rules - Processor Outputs) Using the delta depth of a processor we can predict the depth of its outputs. For each output of processor $p$, $\langle o, t\rangle \in out(p)$ the following holds:

**Definition 6.13.** *(Depths, Depth Mappings)* The information model $M$ for representing predicted patterns in execution provenance of a workflow $w$ is comprised of the

**If** $\Delta Proc(p) = n$, $dDep(p.o) = m$ **then** $pDep(p.o) = m + n$.

A delta depth associated with a processor becomes delta depth its outputs.

following relations:

Depths $D \subseteq PRO \times POR \times \mathbb{N}^+$ is the set of possible nesting levels for lists that appear at a designated port during workflow execution. We denote a particular depth $\langle p, r, m \rangle \in D$ as $d_m^{p.r}$, where $m$ denotes a depth index. $m$ can range over values in $[1..k]$ where $pDep(p.r) = k$. Depth indices start from 1, which corresponds to the nesting level of 0. A depth with index 1 refers to the top level collection at nesting level 0, whereas a depth with index 2 refers to all collections at nesting level 1.

Depth Mappings $DM \subseteq D \times D$ is the set of predictions on the existence of discrete lineage relations among list items within collection structured data to appear at designated ports. We denote a depth mapping $\langle d_{m1}^{p1.r1}, d_{m2}^{p2.r2} \rangle \in DM$ with $d_{m1}^{p1.r1} \rightarrow d_{m2}^{p2.r2}$. If there is such a mapping we can be assured that for any execution of workflow $w$ there will be discrete lineage relations among list items at nesting levels designated by each depth.

**Example 6.14.** Intuitively, if we were to think of list structured data of type $L^n(s)$ as an n-dimensional array, a depth would correspond to a particular dimension of that array. Imagine that within workflow $w$ we have a processor $p$ with input port *in* where $pDep(p.in) = 2$. During a particular execution of that workflow data of type $L^2(s)$ (as predicted) would appear at port *in*. An example such value could be:

$$v1 = [\ [\text{“11”}, \text{“12”}], [\text{“21”}, \text{“22”}], [\text{“31”}, \text{“32”}]\ ]$$

The depth $d_1^{p.in}$ would refer to the 1st dimension of this array. $d_1^{p.in}$ is an address for lists at nesting level 0, which for $v1$ is a single list comprised of the set of items {[“11”, “12”], [“21”, “22”], [“31”, “32”]}. Meanwhile $d_2^{p.in}$ is the second dimension, an address for all lists at nesting level 1, for $v1$ these are 3 particular lists, first comprised of items {“11”, “12”}, second comprised of items {“21”, “22”} and so on.

**Definition 6.15.** (Depth Definition Rule) For a processor $p$ with input/output port $r$:

**If** $pDep(p.r) = n$, $i \in \mathbb{N}$, $1 < i \leq n$ **then** $d_i^{p.r}$.

The number of possible depths for a port is bound by the predicted depth of that port. Whenever the predicted depth of a port is calculated this can be used to define depths for this port with indices ranging from 1 up to the port’s predicted depth.

Our earlier observation on *eval* allows us to model anticipated implicit lineage relations among collections in the form of mappings among depths.

**Definition 6.16.** (Depth Mapping Rule - Single Input Processor Evaluation) For a single input processor $p$ executed with *eval*, where $in(p) = \{\langle i, 1, t1 \rangle\}$ $out(p) = \{\langle o, t2 \rangle\}$ the following apply:

**If** $\Delta Dep(p.i) = m$, $k \leq m$, $k \in \mathbb{N}^+$, $d_k^{p.i}$, $d_k^{p.o}$ **then** $d_k^{p.i} \rightarrow d_k^{p.o}$.

This rule creates mappings among same indiced depths of input and output ports from 1 up to delta depth of the input.

Intuitively this rule states that lists at depth $n$ of the processor’s output will be shaped by corresponding lists at depth $n$ of the input. In other words, if there is such a mapping between two depths, we can be assured that there are discrete lineage relations among each item within lists at corresponding depths. This assurance brought by the existence of depth mappings provides the foundation of our static analysis.

**Example 6.17.** Using rule in 6.16 a single depth mapping will be inferred for the $List\_To\_String$ processor, which is $d_1^{in} \rightarrow d_1^{out}$ (we omit processor name prefix for brevity). We know that this mapping implies similar structure so if the workflow were to be run with an input list of (three) items, where each item is itself a list (of two). Then as per rules outlined thus far we can expect the output to be a list of (three) items, where each list item will be string with depth 0 equal to the defined depth of $List\_To\_String.outstr$.

Missier et al [MPB10] have provided the pioneer insight into the impact of Taverna's iteration on provenance. This insight has allowed us to anticipate implicit lineage (among lists) with the notion of depth mappings. Meanwhile Missier et al [MPB10] have used it to anticipate explicit lineage relations, with a mechanism they call *Index Projection*. Such that, for a given prospective location of an output data item, the location of the contributing input item can be calculated. Note that, both depth mapping and *Index Projection* calculations would be independent from any particular input data size, or data value.

The ability to make predictions on provenance can have several potential uses:

- Missier et al [MPB10] have shown that *Index Projection* can be utilised in provenance query optimisation. When iteration occurs throughout a workflow with long dataflow paths one can shortcut lineage traversal by relying on location calculations.

- Mappings/Projections essentially provide formulas to compute provenance on demand, and eliminate the need to record explicit lineage relations at workflow execution time. It could be exploited for efficient provenance storage, which has been studied in the context of workflow systems [CJR08].

What sets apart our work from earlier work of Missier et al [MPB10] is as follows. Missier et al provide depth prediction and *Index Projection* formulas for multi input processors, where the iteration strategy is represented with a flat LHB formula restricted to a single *Cross* product operator. We provide functional specification of the *Dot* product as well as the *Cross* product. Furthermore, we provide declarative rules for depth mapping inference for multi-input processors with complex iteration strategies (nested LHB formula trees), which covers the entire spectrum of Taverna workflows. Finally we formalise the notion of broken factorial design anti-pattern in provenance and provide declarative rules to detect it.

To this end we have analysed processor execution with a single input, where the processor did not have an associated LHB formula. While iteration in the case of single inputs may come across as an adjustment feature, iteration in the case of multi-input processors with LHB formulas is Taverna's key mechanism for realising factorial designs, i.e. parameter explorations. We will now illustrate the role of LHB formulas in factorial design.

### 6.2.2 Factorial Designs with Multi-Input Iteration

We will now introduce *Dot* and *Cross* product operations with examples (formal specifications will be given in Section 6.3). Both operations creates tuples out of items they encounter in lists, which could themselves be tuples. Note that in a repeated application of these operations, n-ary tuples will be generated. Note however in the formal representation of Taverna workflows there is no separate type for tuples. To adhere to this restriction we assume that *Dot* and *Cross* product operations are underpinned by functions that accept and return *string* types, where the result is a *string* based representation of a tuple.

Cross product is a *cartesian product* function that adheres to the nested structure of the lists, similar to the recursive evaluator given earlier, when the cross product encounter lists it will traverse down to find items of appropriate depth. So unlike the traditional cartesian product, which creates a list out of two lists, the Cross product will create an output of type $L^2(s)$ when applied over two inputs of type $L(s)$.

$$\begin{gathered}(cross\quad [\text{“}a\text{”}, \text{“}b\text{”}, \text{“}c\text{”}]\quad [\text{“1”}, \text{“2”}]) = \\ [[\text{“}a{\times}1\text{”}, \text{“}a{\times}2\text{”}], [\text{“}b{\times}1\text{”}, \text{“}b{\times}2\text{”}]], [\text{“}c{\times}1\text{”}, \text{“}c{\times}2\text{”}]]\end{gathered}$$

Dot product is similar to the *zip* function found in many programming languages. It will pair up items at same positions in each list. Similar to Cross product the Dot product also obeys the nested structure of lists. So the Dot product of two inputs of type $L^2(s)$ will result in an output of type $L^2(s)$.

$$\begin{gathered}(dot\quad [\text{“}x\text{”}, \text{“}y'\text{”}], [\text{“}z\text{”}, \text{“}t\text{”}], [\text{“}u\text{”}, \text{“}v\text{”}]]\quad [[\text{“1”}, \text{“2”}], [\text{“3”}, \text{“4”}], [\text{“5”}, \text{“6”}]]) = \\ [[\text{“}x{\times}1\text{”}, \text{“}y{\times}2\text{”}], [\text{“}z{\times}3, \text{“}t{\times}4\text{”}], [\text{“}u{\times}5\text{”}, \text{“}v{\times}6\text{”}]]\end{gathered}$$

In the previous section we saw that the driver of iteration is the depth difference, delta, between the expected and encountered depth of input $i$ of a processor $p$. An intuitive way to think about delta depth is to think of it as a space of dimension $\Delta Depth(p.i)$ to be explored. In the case of multiple inputs, the application of the processor to input lists is preceded by step(s) that are responsible for creating an overall input space from individual spaces supplied for each input. This process of input space creation is informed by the LHB formula. We provide the intuition for parameter spaces with three examples and identify depth mappings that will be inferred by our analysis rules. In Section 6.3 we will provide a functional exposition of input space calculation and provide the rules that underpin the deduction of such mappings.

**Example 6.18.** Consider the case in Figure 6.4. $concat3Str$ is defined to be a processor that has three input ports, each accepting singleton strings (input port type $s$), and returns their concatenation. In this example each input port is instead exposed to a list of strings $L(s)$. Each such list represents a 1-dimensional space for respective input parameters. The LHB formulas are left-associative in Taverna the *Cross* product of three lists will be created by first creating the cross product of the first two lists and then crossing the result with the third. As a result of this operation a 3-dimensional input space (nested list of depth 3) comprised of input triples will get generated. These triples represent all possible combinations of inputs from respective input lists. Once the input space is prepared Taverna will explore this space by configuring the recursive evaluator given earlier in (6.1). An output per item in the input space will be generated. The mappings between the dimensions of individual inputs and the output depends on the order of inputs in the LHB formula, and the operator that combines them. For $concat3Str$ the mappings that will be inferred based on the analysis of workflow description will be : $d_1^{alphabet} \rightarrow d_1^{result}$, $d_1^{symbols} \rightarrow d_2^{result}$, $d_1^{numbers} \rightarrow d_3^{result}$.

**Example 6.19.** Note that the space to be explored per input can be of n-dimension. The is exemplified in Figure 6.5. Here we have a processor $concat2Str$, which concats two strings. One of its inputs, $surface$ is a 2-dimensional input space, whereas the other is 1-dimensional. Such a setting also results in a cube of possible input combinations to be explored. In this case the 2-dimensional input (i.e. the one with delta depth of 2) will contribute two dimensions to the output space. The mappings for $concat2Str$ will be as follows: $d_1^{surface} \rightarrow d_1^{result}$, $d_2^{surface} \rightarrow d_2^{result}$, $d_1^{numbers} \rightarrow d_3^{result}$.

**Example 6.20.** We exemplify the use of Dot product in Figure 6.6. As discussed before, this operator expects its inputs to be of same size, and returns an output that is of same size as the inputs. Here the LHB prescribes a Dot product of input spaces for $str2$ and $str4$. Here the delta for both of these inputs is 1, hence a Dot product is possible, which creates a single input space that is representative for both inputs. This space is then combined with the input space for $str1$ with a cross product. Note that the formula also prescribes how input spaces for $str3$ should contribute to the overall space. However, as the input for the ports is of expected depth, i.e. has no delta depth. There is no input space to be explored for $str3$. As a result each input quadruple in the overall input space that will be prepared with respect to the LHB formula will have the same value for str3. In other words $str3$ is not a parameter that drives the iteration of $concat4str$. The inferred depth mappings will be as follows: $d_1^{str1} \rightarrow d_1^{result}$, $d_1^{str2} \rightarrow d_2^{result}$, $d_1^{str4} \rightarrow d_2^{result}$.

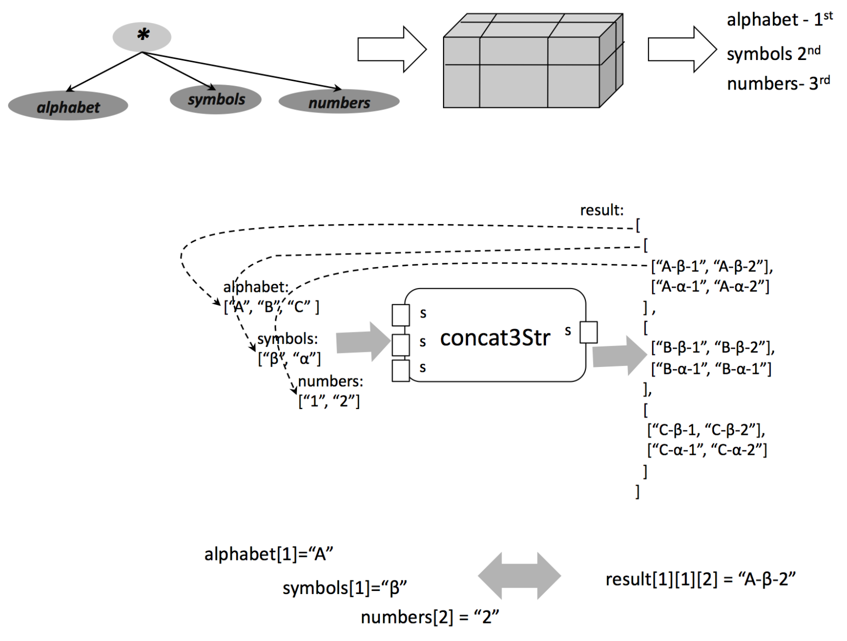


Figure 6.4: Illustration of input space resulting from Cross product of three parameters.

In all our examples in Figures 6.4, 6.5 and 6.6 the defined depth for the output of (string concatenation) processors is 0 (each invocation produces a single string). As a result the application of the processors will result in an output list structure that is of the same depth as the dimension of the overall input parameter space.

## 6.2.3 Breaks in Factorial Design

So far we outlined the elements of workflow analysis on a per processor basis. Our ultimate goal is to perform the analysis over a workflow comprised of multiple processors. As illustrated with the Astronomy workflow Chapter 5 broken factorial design (informally) corresponds to an activity record in execution provenance where the activity consumes data descending (via multiple activities) from distinct workflow input parameters. In this section we illustrate and formalise this pattern as a discontinuation of depth mappings.

**Example 6.21.** Figure 6.7 there are two processors, composed by a dataflow link. *concatStr* processor explores an input space of 2-dimension (based on the cross product of its inputs). As per rules that will be given in Section 6.3 $\Delta Proc(concatStr) = 2$. Two depth mappings will be created for this processor.

For *concatStr*: $d_1^{alphabet} \rightarrow d_1^{outStr}$, $d_1^{numbers} \rightarrow d_2^{outStr}$

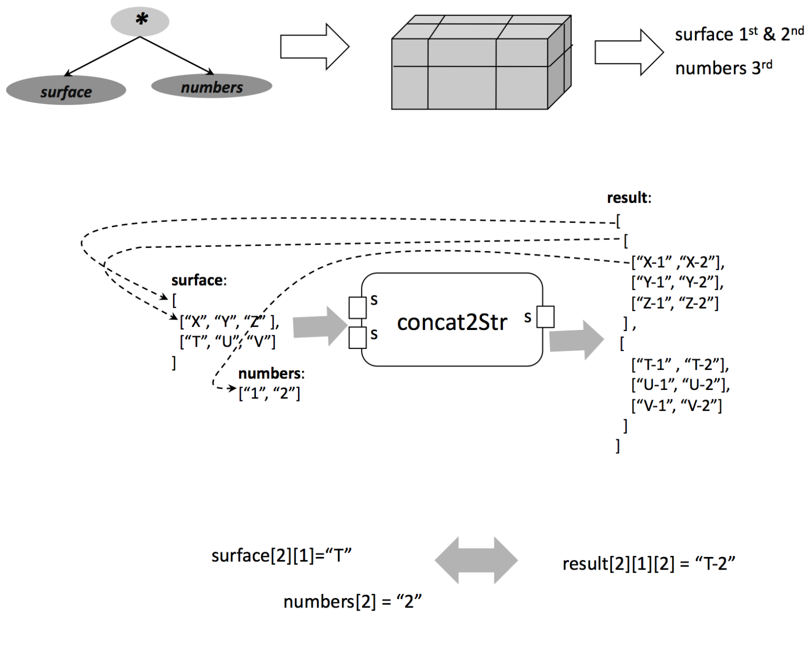


Figure 6.5: Illustration of input space using Cross product of two parameters.

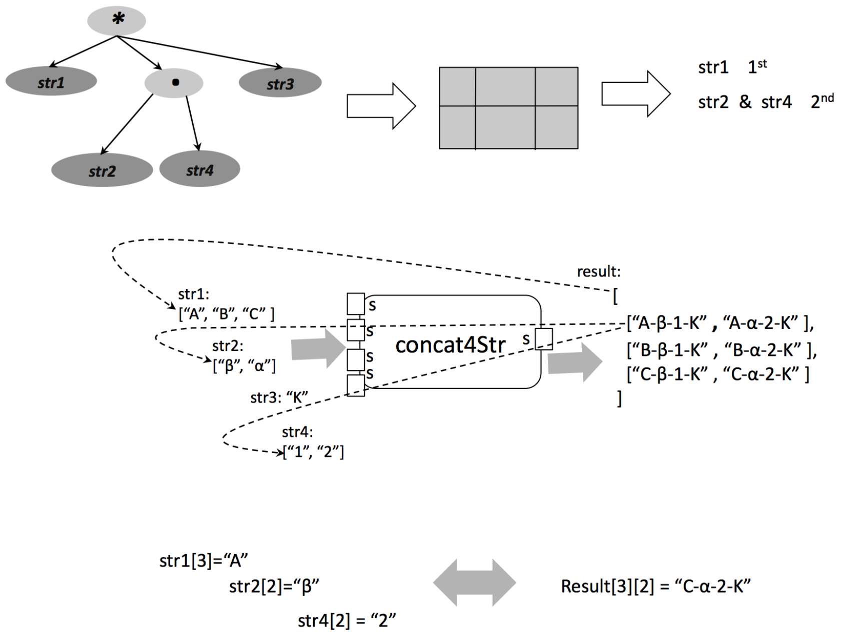


Figure 6.6: Illustration of input space using a combination of Cross and Dot over four parameters.

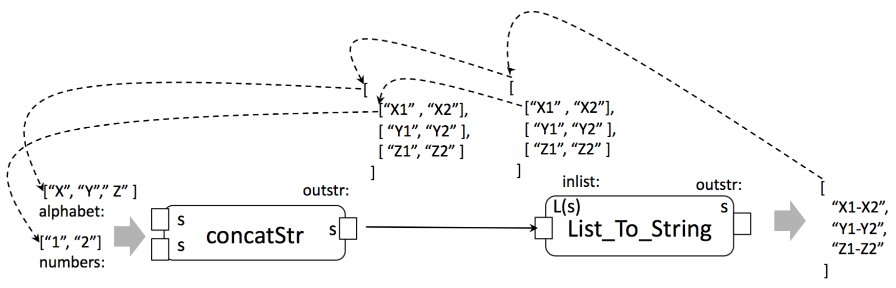


Figure 6.7: A broken factorial design illustrated with discontinued depth mapping at the *List_To_String* step.

As per rules that will be given in Section 6.3 when the data from the output of one processor is transferred to the input port of a follow on processor via a dataflow link, depth mappings will be created among the link source and sink.

For the dataflow link: $d_1^{outStr} \to d_1^{inList}$, $d_2^{outStr} \to d_2^{inList}$

Based on the predicted output depth of *concatStr* processor, *List_To_String* processor will be exposed to $L^2(s)$, greater than its defined input type $L(s)$. As a result we'll have $\Delta Dep(List\_To\_String.inlist) = 1$ and $\Delta Proc(List\_To\_String) = 1$ denoting this processor will be iterated. Note that as the delta for this processor is 1, it will traverse down only one level to find data of acceptable depth. One such item that will be consumed per execution is ["X1", "X2"]. Per execution a single string representing the flattened version of items in the input list will be generated. By doing so *List_To_String* will consume data all at once from the second dimension (depth) of the output of *concatStr*. As per rule in Definition 6.16 the following mappings will be generated:

For the *List_To_String* processor: $d_1^{inlist} \to d_1^{outStr}$

Note here that we are unable to create a mapping for $d_2^{inlist}$ as this depth is beyond the delta depth for *inlist*. In other words this depth is part of the data we consume, rather than a dimension we explore. This inability to map an input depth (which is the target of a prior mapping) to any output depth represents the break in factorial design, which can be defined as follows:

**Definition 6.22.** (Broken Factorial Design) Let $reachesDepths : D \to 2^D$ be a function that accepts an input depth and returns the set of depths that are reachable from the given depth by traversing the depth mapping relations in $w$. We state that workflow

$w$ has broken factorial design for depth $d_i^{p.t}$ if the set $\{d_j^{q.r} \in \mathit{reachesDepths}(d_i^{p.t}) | j > \Delta \mathit{Depth}(\langle q, r \rangle)\}$ is non-empty.

## 6.3 A Model of Taverna's Operation

To this end we have only exemplified how input spaces look like in several multi-input iteration scenarios, and what the depth mapping is for each example. We will now outline a high-level model for the process that Taverna follows to prepare input spaces, to apply processors onto these input spaces and to carry data among data links. After that we will provide (in functional terms) the formal specification of each step of this process. Based on these specifications we will provide depth prediction and depth mapping rules for each step. We will later show how we implement these rules with a Datalog based intentional database.

A convenient way to model the process that Taverna adopts in executing workflows is to break it down to a set of known computations. We illustrate these for the execution of *concat*4*Str* processor in Figure 6.8. There are different four types of computations involved:

- the carrying of data through a dataflow link. Depicted with *link* boxes in Figure 6.8.
- the initialisation of a processor's (multiple) inputs and identifying whether there exists an input space to be explored or a single input. Depicted with *init* boxes in Figure 6.8.
- the build up of the space of input tuples from the individual input spaces using the LHB formula as a guideline. Depicted with *dot* and *cross* boxes in Figure 6.8.
- applying the processor to each input tuple in this space. Depicted with the *eval* box in Figure 6.8.

We will now describe the behaviour of each kind of computation as function followed by corresponding depth prediction and mapping rules for that function. Taverna's execution behaviour for a dataflow link is given with the *link* function.

**Definition 6.23.** (*link* function) $\mathit{link} : \langle \mathbb{N} \times \mathcal{T} \rangle \rightarrow \mathcal{T}$ is a function accepts an input value and a target depth and returns an output data value with depth wise size equal to the target depth. *link* ensures that the nesting level of the result is equal to the target

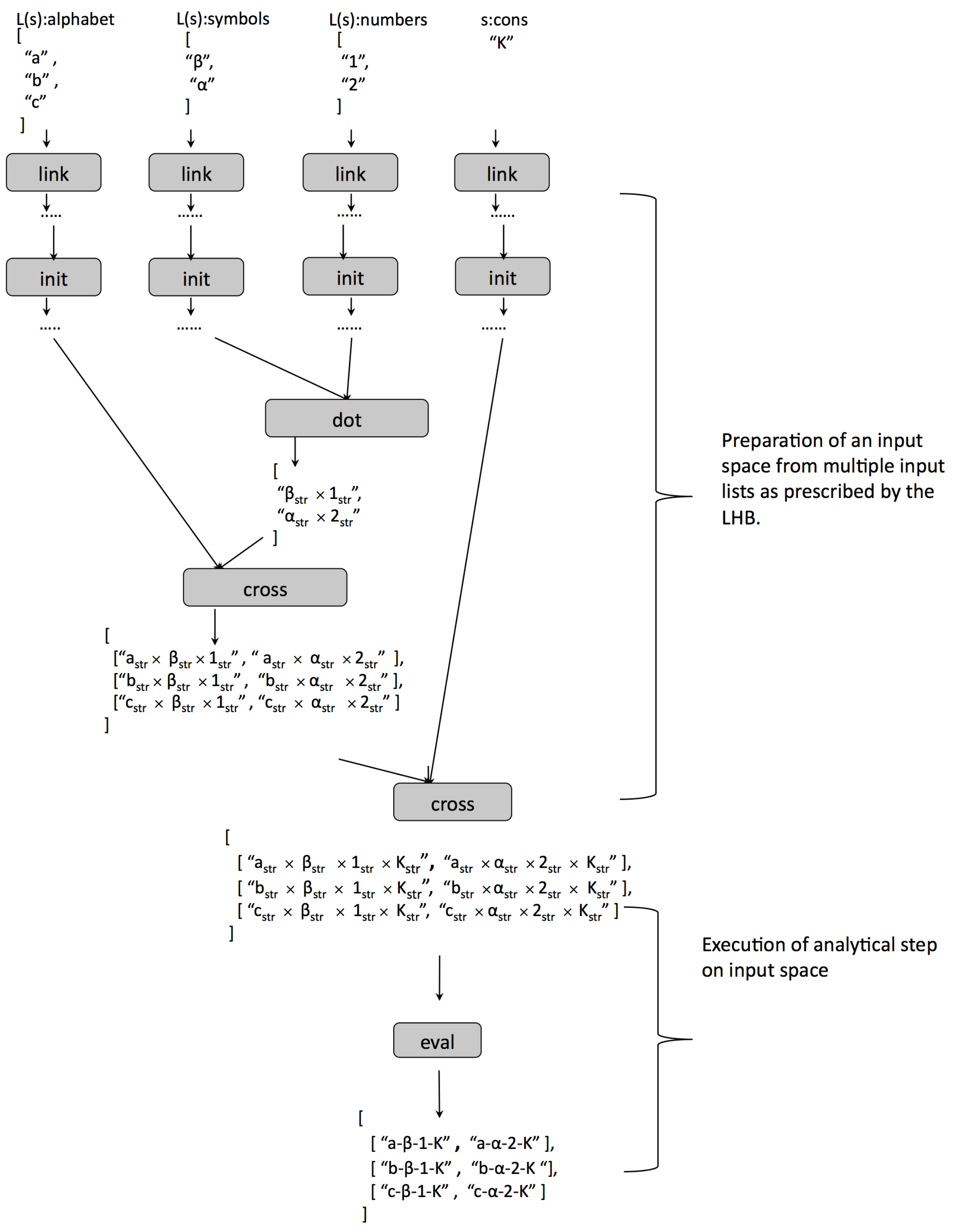

L(s):alphabet
[ "a" , "b" , "c" ]
L(s):symbols
[ "β", "α" ]
L(s):numbers
[ "1", "2" ]
s:cons
"K"
link
link
link
link
init
init
init
init
dot
[ "$β_{str} \times 1_{str}$", "$α_{str} \times 2_{str}$" ]
cross
[ ["$a_{str} \times β_{str} \times 1_{str}$", "$a_{str} \times α_{str} \times 2_{str}$" ], ["$b_{str} \times β_{str} \times 1_{str}$", "$b_{str} \times α_{str} \times 2_{str}$" ], ["$c_{str} \times β_{str} \times 1_{str}$", "$c_{str} \times α_{str} \times 2_{str}$" ] ]
cross
[ [ "$a_{str} \times β_{str} \times 1_{str} \times K_{str}$", "$a_{str} \times α_{str} \times 2_{str} \times K_{str}$" ], [ "$b_{str} \times β_{str} \times 1_{str} \times K_{str}$", "$b_{str} \times α_{str} \times 2_{str} \times K_{str}$" ], [ "$c_{str} \times β_{str} \times 1_{str} \times K_{str}$", "$c_{str} \times α_{str} \times 2_{str} \times K_{str}$" ] ]
eval
[ [ "a-β-1-K" , "a-α-2-K" ], [ "b-β-1-K" , "b-α-2-K "], [ "c-β-1-K" , "c-α-2-K" ] ]
Preparation of an input space from multiple input lists as prescribed by the LHB.
Execution of analytical step on input space

Figure 6.8: Illustration of iterated execution of *concat*4*Str* over multiple input lists with respect to its LHB definition.

depth by wrapping the input with as many enclosing lists as necessary and returns. The specification of *link* is as follows (bracket [ ] represents list constructor):

$$(link_d \quad v) = \begin{cases} (v) & \text{if } d \leq |v| \\ (link_d \quad [v]) & \text{if } d > |v| \end{cases} \tag{6.2}$$

When realising a dataflow link $l \in LINK$, where $src(l) = \langle p1, o \rangle$ and $snk(l) = \langle p2, i \rangle$, Taverna instantiates the *link* function with the data value $v$ appearing at the link source, port $o$ of processor $p1$, and the defined depth of the link's target port $d = definedDepth(\langle p2, i \rangle)$. In Figure 6.8 each four links's targets correspond to input ports of the multi-input *concat4Str* processor. The responsibility of *link* is to ensure that its result depth is no less than the target's defined depth, note that inputs with depth greater than the targets defined depth will be forwarded as is.

**Definition 6.24.** (Depth Prediction and Mapping Rules - *link*) For a dataflow link $l \in LINK$, where $src(l) = \langle p1, o \rangle$ and $snk(l) = \langle p2, i \rangle$ realised with the *link* function (as per Definition 6.23) and the following apply:

**If** $pDep(p1.o) = n$, $dDep(p2.i) = m$ **then** $\Delta Link(l) = m - n$

Difference between the link target's defined and link source's predicted depth is the delta for the link. It's value determine the type of composition for the link.

**If** $\Delta Link(l) = d$, $dDep(p2.i) = m$, $d > 0$ **then** $pDep(p2.i) = m$.
**If** $\Delta Link(l) = d$, $pDep(p1.o) = n$, $d \leq 0$ **then** $pDep(p2.i) = n$.

A positive delta represents wrapped composition and the predicted depth of target would be the same as its defined depth. A negative delta represents iterated, a zero delta represents simple composition. In these latter cases no wrapping occurs hence the predicted depth of the link's target would be the same as its source.

**If** $\Delta Link(l) = d$, $d > 0$, $d_n^{p1.o}$, $d_{n+d}^{p2.i}$ **then** $d_n^{p1.o} \rightarrow d_{n+d}^{p2.i}$.
**If** $\Delta Link(l) = d$, $d \leq 0$, $d_n^{p1.o}$, $d_n^{p2.i}$ **then** $d_n^{p1.o} \rightarrow d_n^{p2.i}$.

If the link represents wrapped composition, then each depth of the source map to depths of the target with an index shift factor equal to delta of the link. For simple or iterated composition, no wrappings are made, so all depths of source map to same indiced depths of target.

Next we outline the *init*, *dot*, *cross* product steps. As illustrated in Figure 6.8 the composition of these with the earlier introduced *eval* function allows for the modelling of multi-input processor invocation. In order to calculate depth mappings for compositions we provide depth mappings for each individual step, by assuming that these steps are specialised Taverna processors that are evaluated with their corresponding function. A straightforward way to model specialised processors is to exploit the *procFun* relation ( introduced earlier in Definition 6.2) to inform the Taverna engine on which function shall be used for a step's execution.

**Definition 6.25.** (*init* function) $init : \mathbb{N} \times \mathcal{T} \rightarrow \mathcal{T}$ is a function that replaces items (that are of a designated depth) within a (nested) input collection with their string-based representation. *init* can be realised using *eval* by passing *makeStr* (defined next) as the processor function parameter.

$$(init_l \quad v) = (eval_l \quad makeStr \quad v) \tag{6.3}$$

**Definition 6.26.** (| |) *makeStr* : $\mathcal{T} \rightarrow S$ is a function that returns the string-based representation of a given input value. If the input is a string than the result is simply equal to the input. If the input is a (nested) collection than we assume the result is some string-based representation of this collection.

Note that *init* is single input processor and depth predictions and mappings for given *eval* in Definition 6.16 also apply to *init*.

The purpose of *init* is to truncate depths within data collection that correspond to data values that will be consumed by the follow-on processor evaluation step (*eval* in Figure 6.8), for which an input space is being prepared. We assume such a behaviour for our convenience as it simplifies depth mappings for follow-on (*Dot* and *Cross* product) steps. As a result of truncation, depth wise size of *init*'s output represents the dimension of the input space for an individual one of multiple inputs that will be consumed by the by the follow-on processor evaluation step.

*cross* and *dot* product steps are responsible for building the overall input space out of individual input spaces. The LHB formula associated with a processor (as given earlier in Figure 6.1) is a recipe for Taverna for building the input space. Taverna will evaluate a formula in a left associate and bottom-up manner. Meaning that if in a formula a *dot* or *cross* product operator has more than two operands Taverna will take the dot/cross of the first two and then use this result to dot/cross with the remaining operands. Bottom-up evaluation means that an operand can itself be subformula, in those cases Taverna will first evaluate the sub-formula. The mapping of the LHB formula of *concat*4*Str* processor is the mini-process comprised of *dot* and *cross* product steps in Figure 6.8. In our specifications of *dot* and *cross* product, given next, we use the utility function *makeTuple*.

**Definition 6.27.** (| |) *makeTuple* : $S \times S \rightarrow S$ is a function that accepts two strings representing data values or tuples of values and returns a string represention of a tuple of the two input values. Example tuples created while preparing the input space for the *concat*4*Str* are given in Figure 6.8.

**Definition 6.28.** (*cross* function) *cross* : $\mathcal{T} \times \mathcal{T} \rightarrow \mathcal{T}$ is a cartesian product function for nested lists. The functional specification for *cross* given below is comprised of a two phased evaluation based on the use of an additional function *cross*2 : $S \times \mathcal{T} \rightarrow \mathcal{T}$.

$$\begin{aligned}(cross \quad b \quad a) &= \begin{cases} [(map \quad (cross \quad b) \quad a)] & \text{if } |a| > 0 \\ [(map \quad (cross2 \quad a) \quad b)] & \text{if } |a| = 0 \end{cases} \\ (cross2 \quad a \quad b) &= \begin{cases} (makeTuple \quad a \quad b) & \text{if } |b| = 0 \\ [(map \quad (cross2 \quad a) \quad b)] & \text{if } |b| > 0 \end{cases}\end{aligned} \tag{6.4}$$

In the first phase we traverse down the first input collection until a string item is reached. Afterwards the second phase (*cross*2) starts, where using the item located in the first phase we traverse down the second collection until a string item is reached. A tuple is created out of the two string items located.

**Definition 6.29.** (Depth Prediction and Mapping Rules - *cross* product) For a two-input processor $p$ with $in(p) = \{\langle a,1,s\rangle, \langle b,2,s\rangle\}$ $out(p) = \{\langle c,s\rangle\}$ realised with *cross* the following apply:

| |
|---|
| **If** $\Delta Dep(p.a) = n$, $\Delta Dep(p.b) = m$ **then** $\Delta Proc(p) = n+m$.<br><br>The sum of the delta depths on the two inputs of *cross* becomes a delta for the *cross* product step. |
| **If** $d_j^{p.a}$, $d_j^{p.c}$ **then** $d_j^{p.a} \rightarrow d_j^{p.c}$.<br><br>The depths of the first input will map to same indiced depths of its output. |
| **If** $\Delta Dep(p.a) = n$, $d_j^{p.b}$, $d_{j+n}^{p.c}$ **then** $d_j^{p.b} \rightarrow d_{j+n}^{p.c}$.<br><br>The depths of the second input will map to depths of the output with an index shift factor equal to the delta depth of the first input. |

**Definition 6.30.** (*dot* function) $dot : \mathcal{T} \times \mathcal{T} \rightarrow \mathcal{T}$ is a zip function for nested lists. The functional specification for *dot* is as follows (note here that we map *dot* function onto two lists).

$$(dot \quad a \quad b) = \begin{cases} (makeTuple \quad a \quad b) & \text{if } |a| = |b| = 0 \\ [(map \quad dot \quad a \quad b)] & \text{if } |a| = |b| > 0 \end{cases} \tag{6.5}$$

**Definition 6.31.** (Depth Prediction and Mapping Rules - *dot* product) For a two-input processor $p$ with $in(p) = \{\langle a,1,s\rangle, \langle b,2,s\rangle\}$ $out(p) = \{\langle c,s\rangle\}$ realised with *dot* function the following apply:

| |
|---|
| **If** $\Delta Dep(p.a) = n$, $\Delta Dep(p.b) = n$ **then** $\Delta Proc(p) = n$. <br><br> The delta depths on the two inputs of *dot* becomes a delta for the *dot* product step. |
| **If** $d_j^{p.a}$, $d_j^{p.c}$ **then** $d_j^{p.a} \to d_j^{p.c}$. <br> **If** $d_j^{p.b}$, $d_j^{p.c}$ **then** $d_j^{p.b} \to d_j^{p.c}$. <br><br> In terms of depth mappings depths of both the first and second input will map to same indiced depths of the output. |

Once the input space is prepared then the we apply the processor by using a single input recursive evaluator *eval*, for which depth predictions and mappings were given in Definition 6.16. This evaluator will simply traverse down each list until it finds non-list items (tuples), then invokes the function associated with the multi-input processor (e.g. $concat4Str$) using this tuple. As *eval* is preceded with the preparation of the overall input space. In terms of depth mappings each depth in the input space will be mapped to same indiced depth of the output of processor evaluation.

The fine-grained modelling of Taverna's behaviour in executing workflows provides us with the following guidelines for devising analysis rules:

- Depth mappings would need to be adjusted in cases of dataflow links that correspond to wrapped iteration.

- Determining the range of depths that will get truncated at processor input initialisation are key in detecting breaks in factorial design. As these depths will not be mapped to any depth in the output of processor application.

- If we find the mappings of depths from the individual input spaces to the overall input space then this mapping could also be used as the mappings from each individual input space to the output. (Note that *eval* imposes no shifts on depths).

- The mappings from individual input space to the overall input space amounts to computing the mappings for the LHB formula structure.

## 6.4 Rule-Based Workflow Analysis

### 6.4.1 Approach Overview

We adopt a declarative logic programming approach for the static analysis of Taverna workflow descriptions. The analysis is *static*, as it is performed solely over the workflow description, without running the workflow. We use Datalog [dat12] programmes to represent the analysis. Programs are comprised of 1) the extensional database (EDB), which is a collection of facts, and 2) the intensional database (IDB), which is a set of rules used to deduce facts. A modular view of the programme structure of our analysis is given in Figure 6.9.

- We represent the Workflow Description as a set of EDB facts.

- We provide Depth Prediction rules, which, for a given depth configuration for a workflow's input's, will deduce the following predictions for its execution: what kinds of composition each dataflow link will imply, what the (depthwise) size of the input space for each processor will be, what will the (depthwise) size of each processor output be and also the (depthwise) size of workflow outputs. The Depth Prediction Rules utilise facts produced by a sub-module of rules that are dedicated to the calculation of processor input space, and the mapping from their input space to output according their respective LHB Formulas.

- We allow users to define an experimental Context by specifying it as an extensional fact, which is a combination of a workflow input port and a depth. A context definition represents a named parameter space that will be used to drive iteration in a workflow, a context therefore represents the collection of parameters which are later to be used in PDDS queries as indices to reach results of interest for a workflow. An example of a context is the galaxy name list ($list\_cig\_name$) parameter of our case study workflow.

- We provide Reaching Rules that compute depth mappings for a given context. Reaching rules check whether mapping can be sustained throughout the workflow, or is discontinued.

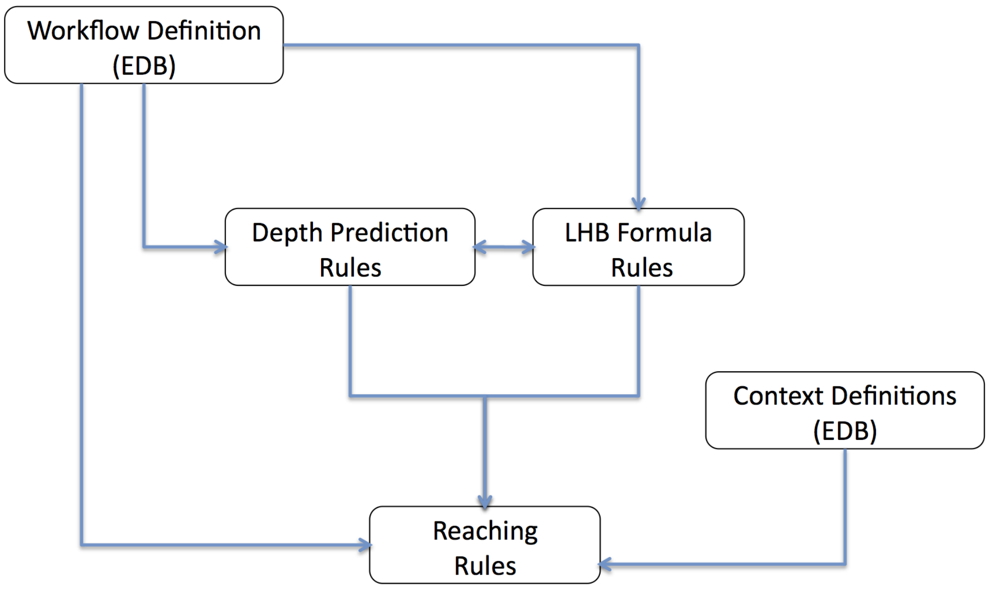


Figure 6.9: Modules of rules used for our analysis and their dependencies.

The rules given as part of our predicted provenance model *PP* (Definition 6.5) map to a number of rules in the Datalog implementation. In the following section we discuss Datalog rules in rule blocks that make-up each module. For reference in Table 6.1 we provide mappings from rules in the abstract (predicted provenance) model *PP* to Datalog rule blocks.

| **Rule Definition in *PP*** | **Datalog Rule Block** |
|---|---|
| 6.10. (Delta Depth Calculation Rules) | DP3 |
| 6.11. (Depth Prediction Rules - Single Input Processor) | LHB1 |
| 6.12. (Depth Prediction Rules - Processor Outputs) | DP4 |
| 6.15. (Depth Definition Rule) | R1 |
| 6.16. (Depth Mapping Rule - Single Input Processor Evaluation) | LHB2, R4 |
| 6.24. (Depth Prediction and Mapping Rules - link) | DP1, DP2, R2 |
| 6.29. (Depth Prediction and Mapping Rules - cross product) | LHB1 |
| 6.31. (Depth Prediction and Mapping Rules - dot product) | LHB1 |
| 6.22. (Broken Factorial Design) | R3 |

Table 6.1: Mappings from Abstract Model Rules to Datalog Rule Blocks.

### 6.4.2 Datalog Rule Structure

Several researchers have previously used Datalog for provenance representation and querying [dat12] and consistency checking [MDB$^+$13]. Moreover, in the field of workflow verification there exists research on modelling workflows with well-studied process formalisms such as Petri Nets [VDA98] or π-calculus [CGG09] and using temporal-logic based analyses over these process representations. We have opted for Datalog for modelling analysis rules as the broken factorial design pattern we were trying to detect does not have a temporal dimension. Datalog programs are collections of rules, which are Horn clauses or if-then expressions [AHV95]. A Datalog program consists of a finite set clauses of the form:

$$A_0 \quad :- \quad A_1, \quad ... \quad ,A_m \quad (m \geq 0) \tag{6.6}$$

where each $A_i$ is a positive atom of the form $r(t_1,...,t_k)$ where each $t_i$ is a variable or a constant. $:-$ is read as "$if$". There can be two forms of clauses:

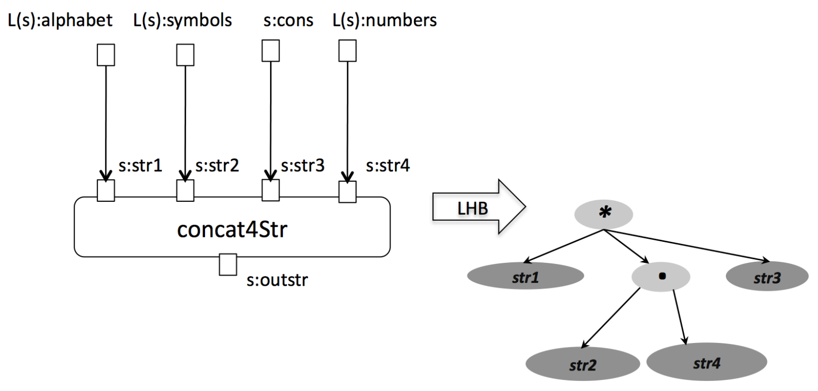
- facts, that correspond to the case when $m = 0$
- rules, that correspond to the case when $m > 0$

A rule is comprised of a head and a body. The RHS of the rule clause is the *body*, the LHS is the *head*. Rule Head is an atom and the Body is comprised of the conjunction (AND) of zero or more atoms. The rule implies that atom $A_0$ is $true$ if atoms $A_1$ to $A_k$ are $true$.

### 6.4.3 Workflow Description Facts

For illustration a fragment of the workflow given earlier in Figure 6.2 that includes the $concat4Str$ processor and the corresponding EDB facts are given in Table 6.2. We adopted the vocabulary from Wfdesc [BZG$^+$15] model for naming predicates in our extensional database. With our facts we represent, workflows, their input and output ports. Workflow involve processes (Taverna's processors), processes also have input and output (ports). The EDB also provides the defined depths for all ports. As a kickstart to the depth prediction rules (described in the next section), the EDB also involves predicted depth for the input ports of the workflow, which equal their defined depths.

For each processor the EDB contains facts that represent the hierarchical structure of the LHB formula. Note that the siblings nodes in a LHB tree are ordered according to their order of appearance in the LHB formula.



```
workflow(w1).
workflowInput(w1,alphabet). workflowInput(w1,symbols).
workflowInput(w1,cons). workflowInput(w1,numbers).

definedDepth(w1,alphabet,1). predictedDepth(w1, alphabet,1).
. . . . . . . . . . . . . . . . . . . . . . . . . . .

process(concat4Str).
processInput(concat4Str,str1). processInput(concat4Str,str2).
processInput(concat4Str,str3). processInput(concat4Str,str4).
processOutput(concat4Str,outstr).

dataLink(dl1,w1,alphabet,concat4Str,str1).
dataLink(dl2,w1,symbols,concat4Str,str2).
dataLink(dl3,w1,cons,concat4Str,str3),
dataLink(dl4,w1,numbers,concat4Str,str4).


%LHB FORMULA TREE STRUCTURE
hasLhbRoot(concat4Str,uid1).

lhbNode(uid1, cross, concat4Str).
lhbNode(uid21, str1, concat4Str).
lhbNode(uid22, dot, concat4Str).
lhbNode(uid23, str3, concat4Str).
lhbNode(uid31, str2, concat4Str).
lhbNode(uid32, str4, concat4Str).

hasChild(uid1, uid21, 0).
hasChild(uid1, uid22, 1).
hasChild(uid1, uid23, 2).
hasChild(uid22, uid31,0).
hasChild(uid22, uid32,0).
```

Table 6.2: A fragment of workflow and its representation with Datalog predicates.

### 6.4.4 Depth Prediction Rules

Depth prediction, the rules of which is given in Table 6.3 occurs incrementally, through dataflow links and processors.

- Rule Block DP-1: As the predicted depths of workflow input ports have been provided as part of the EDB, this will initiate the rules that determine the composition types of links from these input ports to processors' input ports. As per depth prediction rules given earlier (Definition 6.24), what determines a composition (link) type is the difference between the defined depth of the link's target and the actual depth encountered at the source (i.e. the predicted depth of the source). The facts inferred for our example workflow are depicted in Figure 6.10, the predicates *wrapped*, *iterated* and *smooth* are used to assert the composition type of each dataflow link.

- Rule Block DP-2: The kinds of composition for each dataflow link informs what the predicted depth of the target of the dataflow link will be. For iterated and simple composition the predicted depth of the target will equal the predicted depth of source. For wrapped composition a depth adjustment occurs so that the predicted depth of target is equal to the defined depth of target.

- Rule Block DP-3: When we have information on the predicted depths of a processor's input ports, then we can determine per rules outlined earlier (Definition 6.10) whether an input port is initialised with a single input or a space of inputs. The *deltaDepth* facts deduced for each port of $concat4Str$ is given in Figure 6.10.

- Rule Block DP-4: In order to make a prediction for the output port of a processor, we need to sum up two pieces of information (recall Definition 6.12). First is the defined depth of the processor's output port, this information is available in the EDB. The second is the size of the overall input space, this information is produced by rules in the LHB Formula module. There are two rules in Rule Block DP-4 (see Table 6.3). One is designed to handle the cases with processors that have inputs, and the other is designed for processors without inputs. For the latter case the defined depth of the output becomes also the predicted depth. Rules in block DP-4 infer the predicted depth for output $outstr$ of $concat4Str$ as 2 (see Figure 6.11) as the defined depth of $outstr$ is 0, and the size of input space is 2 (we will present the rules for its calculation next).

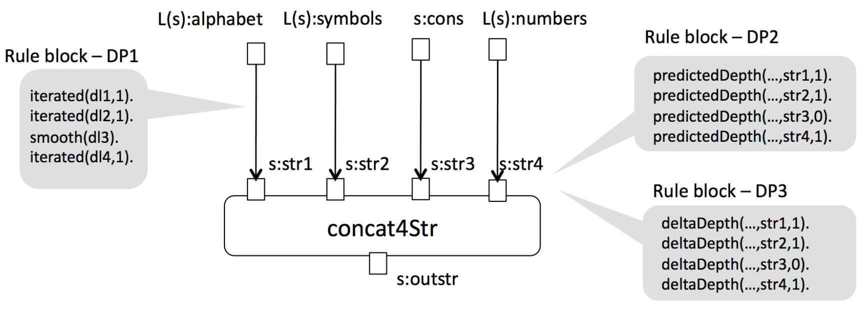


Figure 6.10: Illustration of Deductions of the Depth Prediction rules.

### 6.4.5 List Handling Formula Rules

List Handling Behaviour (LHB) Formula rules, which are given in Table 6.4 perform two computations, 1) the calculation of the overall size of the input space, and 2) the calculation of the mappings from depths in individual input parameter spaces to the depths in overall input space. We make use of the LHB Formula tree for each processor given in the EDB as guidelines to perform these computations. Equations (6.29) and (6.31) given earlier for *cross* and *dot* product provide the formulas for what the size of their output will be depending on the size of input. Normally, if we apply these formulas bottom up to the LHB tree, the output size of the top node will be the size of the overall input space. Note that the equations also provide us with a formula to compute mappings from depths of inputs these operators to outputs. In order to compute the input space size and the mappings simultaneously for a formula tree, our rules (Rule Block LHB-1) in Table 6.4 prescribe a depth-first pre-order traversal of the LHB tree as illustrated in Figure 6.11.

We use two predicates to accumulate the input space size throughout the traversal, these are *sizeLhs* and *sizeCumulative* predicates. *sizeLhs* represents the size of the input space as defined by the portion of the formula to the left of the node. *sizeCumulative* represents the size of lefthand side combined with the size of space of the current node. The first child of each node inherits *sizeLhs* from their parent. Note that the (leaf) port nodes represent an individual input space that is of size *deltaDepth* computed for that port. When *sizeLhs* reaches a port node *sizeCumulative* is inferred by adding *sizeLhs* with the *deltaDepth* for the (port) node.

*sizeLhs* for a non-first child is computed with information from its left sibling depending on the parent operator node the siblings belong.

- Recall from Definition 6.29 that the size of output of binary cross product is the

**DP-1: Determine the kind of composition for data links.**

```
wrapped(ID,D):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO, SNK_POR),
predictedDepth(SRC_PRO,SRC_POR,Z),
definedDepth(SNK_PRO,SNK_POR,Y), Z<Y, D=Y-Z.

iterated(ID,D):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO, SNK_POR),
predictedDepth(SRC_PRO,SRC_POR,Z),
definedDepth(SNK_PRO,SNK_POR,Y), Z>Y, D=Z-Y.

smooth(ID):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO, SNK_POR),
predictedDepth(SRC_PRO,SRC_POR,Z),
definedDepth(SNK_PRO,SNKPOR,Y), Z=Y.
```

**DP-2: Propagate predicted depth through a dataflow link, there is only an adjustment in the case of wrapped composition.**

```
predictedDepth(SNK_PRO, SNK_POR,RES):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO, SNK_POR),
wrapped(ID,D),
predictedDepth(SRC_PRO,SRC_POR,Z),
RES=D+Z.

predictedDepth(SNK_PRO, SNK_POR,Z):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO,SNK_POR),
iterated(ID,_),
predictedDepth(SRC_PRO,SRC_POR,Z).

predictedDepth(SNK_PRO, SNK_POR,Z):-
dataLink(ID,SRC_PRO, SRC_POR,SNK_PRO,SNK_POR),
smooth(ID),
predictedDepth(SRC_PRO,SRC_POR,Z).
```

**DP-3: Calculate the delta depth for a port.**

```
deltaDepth(PRO, POR,Z):-
definedDepth(PRO, POR,DEFD),
predictedDepth(PRO, POR, PREDD),
Z= PREDD-DEFD.
```

**DP-4: Calculate the predicted depths for outputs of activity using the total size of the input space.**

```
predictedDepth(PRO,O_POR,Z):-
hasLhbRoot(PRO,R),sizeCumulative(R,RT),
processOutput(PRO,O_POR),
definedDepth(PRO,O_POR,DD),Z=DD+RT.

predictedDepth(PRO,O_POR,DD):-
hasLhbRoot(PRO,null),
processOutput(PRO,O_POR),
definedDepth(PRO,O_POR,DD).
```

Table 6.3: Rules for predicting the iterated executions and the corresponding depth adjustment.

sum of the sizes of its inputs. Therefore, in our rules if the parent is a *cross* product then the *sizeCumulative* of the left sibling becomes the *sizeLhs* of the right sibling to reflect the additive nature of *cross* product.

- Recall from Definition 6.31 that the size of output of binary dot product is equal to the individual size of either input. Therefore in our rules if the parent is a *dot* product then the *sizeLhs* of the left sibling becomes the *sizeLhs* of the right sibling.

Finally the *sizeCumulative* of some parent (operator) is computed when sizes for all its children have been computed. The maximum size computed for a child becomes the *sizeCumulative* of parent. As a result of traversal, the *sizeCumulative* for the top operator node becomes the size of the overall input space.

The *sizeCumulative* that we have computed for each leaf (port) node is also an indicator of its depth mapping. With the rules in Rule Block LHB-2 we make this information more explicit using *depthMapping* predicate. Note that if the *deltaDepth* for a port is zero then the input is comprised of a single value and when a space of tuples is built up from multiple inputs the same value for this input will be used for the entire set of tuples. So this input maps to the entire input space. We denote this asserting that its *depthMapping* is 0. If *deltaDepth* is greater than zero then its dimensions (depths) will map to depths in the input space. The value of *sizeCumulative* for a port node denotes the size of the input space inclusive of that port node exclusive of others that come after it in a formula. Therefore we can deduce that the depth with indice *deltaDepth* for the individual input pace maps to the depth with indice *sizeCumulative* in the overall space. Depth mapping information are used in reaching rules, which we describe next.

### 6.4.6 Context and Reaching Rules

We use a predicate named *context* to allow users define experimental contexts in the EDB. A context can be seen as a named depth definition. For our running example let us assume that we define each input item (parameter) in lists of alphabet and symbols to be contexts (Each item is of type *s*, therefore of depth 0). This represented with the ground facts $context(ctxA, w1, alphabet, 0)$. and $context(ctxS, w1, symbols, 0)$.

By building on such definitions the reaching rules presented in Table 6.5 can be described as follows:

- Rule Block R-1: We use context definitions to kickstart the computation of the reach of contexts. By default a context reaches the port it is defined on. As

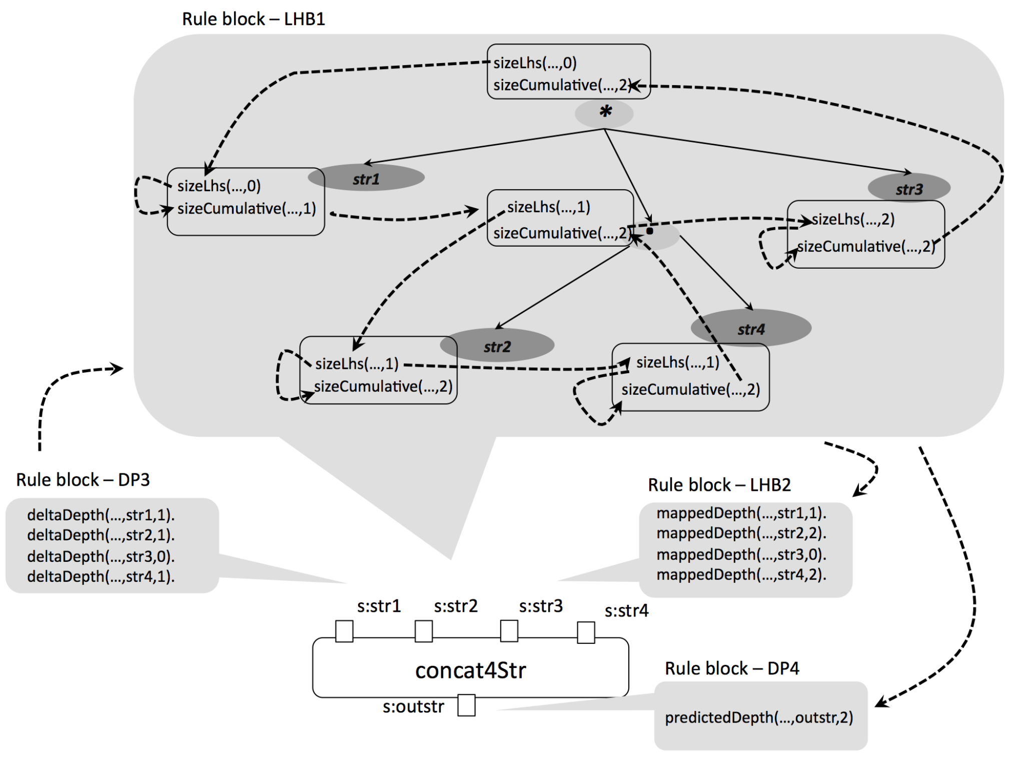


Figure 6.11: Illustration of Deductions of the List Handling Formula rules.

illustrated in Figure 6.12 context *ctxA* reaches depth indiced 1 at the workflow input port *alphabet* and similarly *ctxS* reaches depth indiced 1 at workflow input port *symbols*.

- Rule Block R-2: We propagate reaching from the source of a dataflow link to its target by taking into account the composition types of links. For simple and iterated composition, reaching propagates to the same indiced depths at the target port. For wrapped composition reaching propagates to depths of target with a positive indices shift equal to the amount of the wrapping adjustment made by the link.

- Rule Block R-3: When a context reaches a processor input port depending on the depth it reaches we can determine whether it will reach processor outputs. Recall that while performing depth prediction we calculated *depthMappings* for each depth in the input spaces of individual input ports. So at this stage we need to check whether the depth that a context reaches is a depth within the input space. If it is within input space it will be preserved (or mapped to outputs), if it

```
#const dotnode = dot.
#const crossnode = cross.
```

**LHB-1: Computing the overall size of input space.**

```
sizeLhs(R,0):-  hasLhbRoot(P,R), R != null.

sizeLhs(FIRSTCHILD,VAL) :-
lhbNode(PARENT,_,_),
hasChild(PARENT, FIRSTCHILD, 0),
sizeLhs(PARENT,VAL).

sizeCumulative(NODEID,Z) :-
lhbNode(NODEID, NAME, PROC),
NAME != dotnode,
NAME != crossnode,
sizeLhs(NODEID,VAL),
deltaDepth(PROC, NAME, NDEL),
Z=NDEL+VAL.

sizeLhs(SIBLING2, VAL):-
lhbNode(PARENT,dotnode,_),
hasChild(PARENT, SIBLING1, N),
hasChild(PARENT, SIBLING2, NEXT),
sizeLhs(SIBLING1,VAL), NEXT= N+1.

sizeLhs(SIBLING2, VAL):-
lhbNode(PARENT,crossnode,_),
hasChild(PARENT, SIBLING1, N),
hasChild(PARENT, SIBLING2, NEXT),
sizeCumulative(SIBLING1,VAL), NEXT= N+1.

sizeCumulative(PARENT, VAL):-
hasChild(PARENT, LASTCHILD, X),
#max{V : hasChild(PARENT,_,V)} =Y,
sizeCumulative(LASTCHILD,VAL), X==Y.
```

**LHB-2: Computing depth mappings from each individual input to overall input space.**

```
depthMapping(PROC, PORT, 0):-
lhbNode(NODEID,PORT,PROC),
sizeCumulative(NODEID, VAL), deltaDepth(PROC, PORT, 0).

depthMapping(PROC, PORT, VAL):-
lhbNode(NODEID,PORT,PROC), sizeCumulative(NODEID, VAL),
deltaDepth(PROC, PORT, ND), ND>0.
```

Table 6.4: Rules calculating the overall size of input space and the depth adjustments per input based on LHB formula.

reaches a depth beyond delete depth it will be truncated/discontinued. In Figure 6.12 both *ctxA* and *ctxS* reach respective processor input ports at depth indice 1, note that this is an indice within the boundary of the *deltaDepth* for the input ports, which is also 1, therefore both contexts will be preserved at the processor *concat*4*Str*, and they will be propagated to this processor's output ports.

- Rule Block R-4: We determine to which depth in a processor's output port a context reaches by using the *depthMapping* information associated with input ports. Note that the *depthMapping* for the input port *str*1 is 1, therefore *ctxA* reaches depth 1 in the port *outStr*. On the other hand the mapping for *str*2 is 2, hence *ctxS* is forwarded to depth 2 in *outStr*. Note now our example contexts are mapped to different depths at the same port.

The case when reaching cannot be propagated occurs when a context reaches a depth that is beyond the input space, which denotes that it is not a driver of iteration but a part of data value to be consumed by one invocation of processor. At the *List_To_String* step in Figure 6.12 *ctxS* reaches depth 2 of the input port. Meanwhile *deltaDepth* for this port is 1, in other words *List_To_String* will iterate over a list inputs it can consume. As a result *ctxS* get truncated/discontinued at the *List_To_String* step.

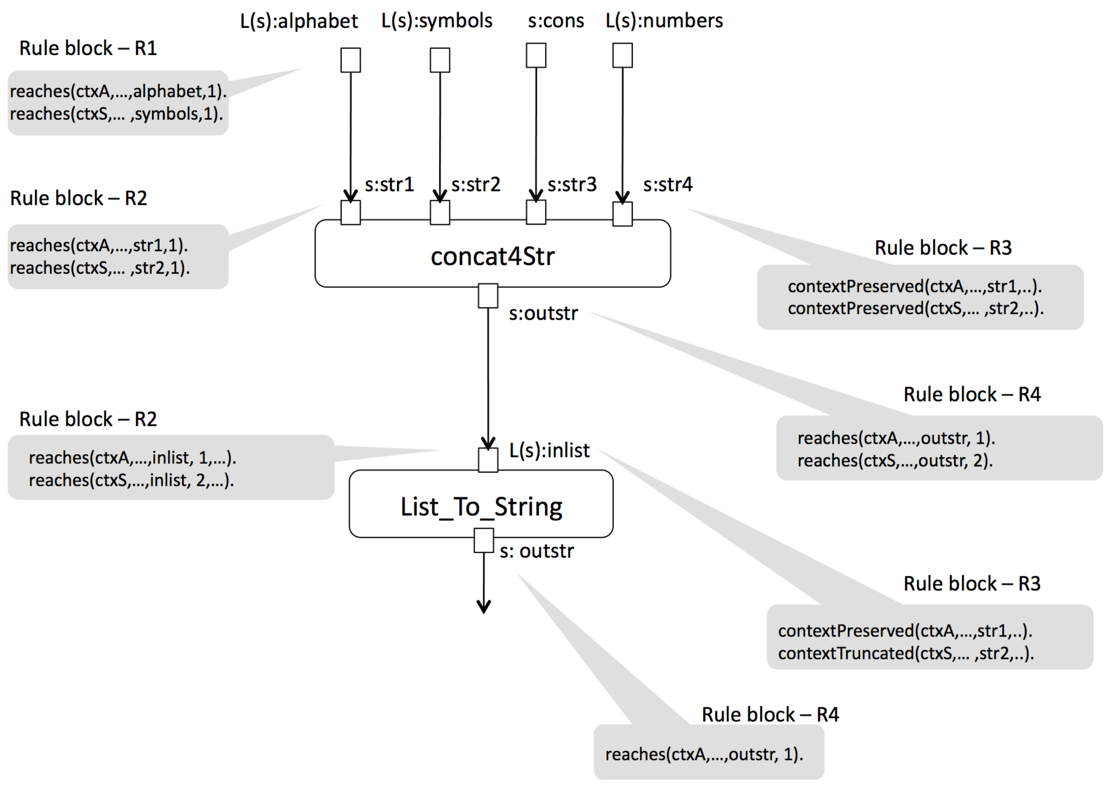

L(s):alphabet
L(s):symbols
s:cons
L(s):numbers
Rule block – R1
reaches(ctxA,...,alphabet,1).
reaches(ctxS,... ,symbols,1).
s:str1
s:str2
s:str3
s:str4
Rule block – R2
reaches(ctxA,...,str1,1).
reaches(ctxS,... ,str2,1).
concat4Str
Rule block – R3
contextPreserved(ctxA,...,str1,..).
contextPreserved(ctxS,... ,str2,..).
s:outstr
Rule block – R4
reaches(ctxA,...,outstr, 1).
reaches(ctxS,...,outstr, 2).
Rule block – R2
reaches(ctxA,...,inlist, 1,...).
reaches(ctxS,...,inlist, 2,...).
L(s):inlist
List_To_String
s: outstr
Rule block – R3
contextPreserved(ctxA,...,str1,..).
contextTruncated(ctxS,... ,str2,..).
Rule block – R4
reaches(ctxA,...,outstr, 1).

Figure 6.12: Illustration of Deductions of the Reaching rules.

To this end we described our rules that for a set of given input depths calculate what the predicted depths of workflow's outputs will be and what the mappings among

**R-1: Kickstart reaching definition.**

```
reaches(CTX,PRO, PORT,Z):-
context(CTX,PRO,PORT,RMPOS),
predictedDepth(PRO,PORT,PRED_DEP),
Z=PRED_DEP - RMPOS.
```

**R-2: Propagate the reach of a context through a dataflow link.**

```
reaches(CTX,SNKPRO, SNKPOR,Z):-
reaches(CTX, SRCPRO, SRCPOR, Z),
smooth(DL1),
dataLink(DL1, SRCPRO,SRCPOR, SNKPRO, SNKPOR).

reaches(CTX,SNKPRO, SNKPOR,Z):-
reaches(CTX, SRCPRO, SRCPOR, Z),
iterated(DL1,_),
dataLink(DL1, SRCPRO,SRCPOR, SNKPRO, SNKPOR).

reaches(CTX,SNKPRO, SNKPOR,Y):- reaches(CTX, SRCPRO, SRCPOR, Z),
wrapped(DL1,X),
dataLink(DL1, SRCPRO,SRCPOR, SNKPRO, SNKPOR),
Y=X+Z.
```

**R-3: Determine whether a context reaching a processor input will reach an output depth or is discontinued.**

```
contextTruncated(CTX,PRO, PORT,Z):-
processInput(PRO,PORT),
reaches(CTX, PRO, PORT, CTX_POS),
deltaDepth(PRO, PORT, LMPOS),
CTX_POS > LMPOS, Z= CTX_POS - LMPOS.

contextPreserved(CTX,PRO, PORT,Z):-
processInput(PRO,PORT),
reaches(CTX, PRO, PORT, CTX_POS),
deltaDepth(PRO, PORT, LMPOS),
CTX_POS <= LMPOS, Z= LMPOS - CTX_POS.
```

**R-4: Propagate the reach of a context from inputs ports of an of an activity to its output ports.**

```
reaches(CTX,P1, OUT1,Z):-
processInput(P1,IN1),
processOutput(P1,OUT1),
contextPreserved(CTX,P1,IN1,CDEL),
depthMapping(P1,IN1,FFAC),
Z= FFAC-CDEL.
```

Table 6.5: Rules for inferring the reach of a context throughout workflow.

depths will be for processor's inputs and outputs. We also described reaching rules, that check whether a named depth in the workflow input port can be mapped continuously throughout the processors in the workflow or whether there exists a processor at which mapping is discontinued. Our rules encapsulated the formulas laid out in the Taverna execution model given in Section 6.3.

## 6.5 Implementation

We have implemented analysis rules using the DLV Datalog implementation [LPF05]. We have developed a tool that converts workflow descriptions in Wfdesc [BZG$^+$15] into a set of EDB facts. Note that Wfdesc supports an abstract dataflow viewpoint of workflows and it does not model information needed to execute workflows. As a result the LHB Formulas of processors are note captured in Wfdesc. We have extended Wfdesc with a simple property to carry LHB formulas from their representation in Taverna Scufl language through to EDB facts. The source for the converter tool can be accessed from the code repository at [Alp15b]. The EDB facts for the simple workflow (given earlier in Figure 6.2) and all inferred IDB facts by our rules are provided in Appendix D. A trace which illustrates that a context defined for the galaxy name input parameter for the case study workflow and the deductions that show that this context is truncated at the $Flatten\_List$ step are can be found in the source code repository at [Alp15b].

We will now review related work on workflow and provenance analysis, which will be followed by a discussion of the use of our workflow analysis results.

## 6.6 Related Work

As mentioned earlier Missier et al [MPB10] have provided the initial insight that, for Taverna workflows, we can pre-compute anticipated lineage based on locations of data in nested input and output collections, as Taverna provides us with its iteration semantics. This work exploits the definitions of the cross product and the iterative processor evaluator to provide a location mapping formula among indices of prospective input/output data collections. Authors have shown that the cost of answering lineage queries using location-mappings would be resilient to increases in workflow size, when compared to the naive query answering based on traversal of lineage. In [DKBL15] Dey et al describe a similar approach for the Kepler workflow system [LAB$^+$06]. Kepler supports a number of different dataflow mechanics to coordinate the execution of analytical steps and their communication. Different mechanics are represented by different workflow execution Directors. The Synchronous Dataflow (SDF) is one such director in which activities consume/produce data tokens to/from containers (similar to ports) with defined rates. Authors provide a location-based lineage computation formula based on token consumption/production rates associated with each activity. Similar to Missier, Dey et al demonstrate that location-based provenance provides benefits in lineage query answering in settings where the workflow size increases.

The benefit of both Missier's [MPB10] and Dey's [DKBL15] location-based provenance approaches is demonstrated in cases where workflows are very large (100+ steps) and all workflow steps are iterated over respective input collections. As a result authors demonstrate gains with synthetic datasets. In our approach we have focused on what can go wrong in querying the provenance of real-world workflows. Our case study has shown that when the iteration structure is not sustained it significantly effects the accuracy of provenance query results. Therefore we have focused on eliciting the case to the contrary of Dey and Missier.

Within a workflow the flow of data among modules is explicit. Recent research in provenance explores the use of static analysis and/or dynamic instrumentation techniques in cases where the flow is implicit in programmes or scripts. One motivation is obtaining a provenance abstraction capturing data's flow or the process's dependencies. In noWorkflow [MBC$^+$14] authors analyse Python scripts to extract function-call hierarchies, which they use to create a view over provenance traces collected by runtime instrumentation of the functions' reads and writes to the file system. In [SGB14] authors employ a taint tracking framework, which instruments programme executions and records which computations are affected by tainted data sources. Here the authors use names of files read by programmes as taint marks and show how such an approach can create fine grained lineage among files, where the programme during its execution writes to one file the data read from another file. In [GCP13] authors apply source code analysis to programmes that underpin each analytical activity in workflows. The motivation here is obtaining provenance that is of finer-granularity than what is available in standard black-box workflow provenance. Through such analysis authors create a record of prospective provenance where each statement in a programme is modelled as a transformational process and the variables read and updated by the transformation are its inputs and outputs.

The focus of these works is having some basic provenance for a system that has not been designed with such features. As the focus is on collecting provenance, these works do not focus how that provenance can be used or what the patterns (fine grained lineage coming from iterations) or anti-patterns (broken factorial design) in provenance can be. Whereas in our work, we have a provenance capability, and we focus on these patterns to increase the fitness of provenance for a particular querying scenario (PDDS).

A technique used in optimising compilers, named data-flow analysis [ASU86], is method-wise similar to our approach. Data-flow analysis is a systematic way of collecting information about a program's states (possible variable values) for distinct points in the control-flow graph. Compilers utilise a number of well-know data-flow analysis types such as "Reaching definitions" or "live variable analysis". Here programme statements are associated with transfer functions that encapsulate if/how the statement alters programmes state information (e.g. variable definitions). A *join* operation is defined to compute state, when multiple programme branches come together. The outcome of data-flow analysis is used to optimise compilation, as in dead-code elimination, which excludes from compilation those statements that assign to a variable, which is never read afterwards.

## 6.7 Discussion

As a first remedy to the problems unearthed in our PDDS case study we presented a static analysis approach to detect broken factorial design in Taverna workflows. We demonstrated that if Taverna iterates over a collection of input items, it creates a corresponding output collection and that this correspondence between collections is an assurance that no output is created using multiple items from the input collection. We have formulated this correspondence as a depth mapping rule among input and outputs of workflow processors. We have broken down Taverna's execution behaviour into a set of restricted classes of computations for which we can define depth mapping rules. We have provided a set of Datalog rules, that are built on the mappings outlined for each class of restricted computation. We illustrated with a small example

- how our rules deduce the predicted depths of data for a given input depth configuration
- how we create depth mappings from inputs to output ports of processors
- how for a particular input depth we can traverse the workflow to determine whether mapping is possible throughout all downstream processors or it is discontinued at a particular processor.

The discontinued depth mapping detected by our analysis rules may not always be pointing to a problem to be fixed. The $Flatten\_List$ processor in our case study workflow, which presents the break point in factorial design is a commonly used data adapter step in Taverna workflows. The nested collection structure created through iterated analyses are represented as nested folder structures and files in Taverna workflow system's data storage layer. In order to reduce the complexity of this physical representation, scientists often create coarser grained data items by flattening nested lists into a single list or a single string to facilitate easier integration of data into textual reports or experimental bundles. In cases where the flattening is a terminal step in a workflow which is intended for reporting it would not have a derogatory affect on PDDS queries. It remains part of our future work to detect how often broken factorial design occur in workflows and whether it has been encoded intentionally as in cases of generating reporting friendly data representations.

A notable aspect of our analysis approach is that it allows the search for an $n-by-1$ pattern in provenance at the workflow level rather than at the individual activity level. If we are to inspect an individual activity by looking at the defined depths of their inputs/outputs, and observe that it consumes a collection and produces a single item or another collection, then we can simply deduce that it would create an $n-by-1$ or $n-by-m$ pattern by its execution. On the other hand this information alone is not sufficient to deduce this pattern poses a threat to PDDS in the context of the overall workflow and workflow input parameters which are query anchor point for PDDS. Our analysis allows for this check to be done at the workflow level.

The intended use for our workflow analysis capability is to provide feedback to workflow designers. On the other hand a prerequisite for such feedback to be appreciated is wider use of workflow provenance, i.e. scientists actually using provenance for PDDS. Lack of tools that exploit workflow provenance, in Taverna and also in other workflows systems, is an important factor that limits the use of provenance. Our anecdotal interactions with scientists in the EU Wf4ever project [HDZ$^+$14] [GSRRS13] has shown that scientists perceive workflow provenance as an esoteric form of technical metadata that is a dump/export of workflow execution. In this viewpoint the only use that they attribute to workflow provenance is for experimental audit, where the dump/export of a particular analysis may be viewed and inspected by a reviewer of the investigation. In the dissertation's conclusions we discuss emerging regulations and practices in scientific data sharing that we think is creating momentum towards increased use of provenance traces of computational data processing.

An apparent limitation of our approach is its specificity to Taverna's way of realising factorial designs (collections and iterations). As reviewed in Chapter 2, the level of support in scientific workflow systems for factorial design is variable, and the mechanisms with which this support is implemented is highly diverse. Therefore it is a challenge to devise an analysis approach that would fit multiple systems. There are, however, emerging efforts that aim to capture the common core in scientific workflow languages in a Common Workflow Language [AT15]. Investigating whether our approach can be generalised, to such languages remains part of our future work, which we discuss in Chapter 6.

## 6.8 Chapter Conclusion

We described a static analysis technique over Taverna workflow descriptions to detect breaks in factorial design. We showed how we exploit the well-defined execution semantics of Taverna system to anticipate the structure of provenance traces, and detect whether there are cases where multiple inputs descending from multiple points in the parameter space will get processed together, thereby breaking factorial design. We will discuss future research in the final chapter of dissertation.

# Chapter 7

# Provenance Annotation

## 7.1 Chapter Introduction

In this chapter we describe our solution in bringing domain-specific descriptions into provenance. We begin in Section 7.2 by outlining our scope: what we aim to annotate, what kind of information the annotations can contain, from where that information can be sourced. In our approach to annotation, we make a number of assumptions based on Workflow Motifs (reported in Chapter 3). In Section 7.3 we revisit workflow Motifs discuss that they reveal

- the activities that encapsulate scientifically significant computation producing data with embedded domain specific descriptions.
- the activities that perform data adaptation causing several copies of data to be present within a provenance trace.

Consequently Motifs hint us on how annotation can occur, where the workflow activities' data creation and copying behaviour, can be matched up with annotation creation and propagation behaviour.

In Section 7.4 we provide an overview of our approach with configurable generic operators for the creation of annotations where the operators perform a middle-man duty by transferring data indexed through provenance to external label creator functions, and then transfer those labels as decoration on to the provenance trace, and also for the propagation of labels among data derivatives created via data adapter steps. In Section 7.4.1 we outline the simple Label model we adopt for carrying domain-specific annotations. In Section 7.4.2 we provide the procedural specification of labelling operators, and their configuration. In Section 7.4.3 we discuss how labelling behaviour associated with activities in scientific workflows can be organised into labelling pipelines that are built using the activity composition structure of the scientific workflow. We outline a basic procedure for creating Labelling Pipelines for Scientific Workflows.

In Section 7.6 we revisit the provenance driven data selection case-study presented earlier in Chapter 5. We observe the improvement that labels brings in realising PDDS queries, we assess the implications of label propagation on the accuracy of labels. The related work is reviewed in Section 7.7.

Our work on provenance annotation has been published in the following papers:

- **P. Alper**, C. A. Goble, and K. Belhajjame. On assisting scientific data curation in collection-based dataflows using labels. In Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science (WORKS 13), pages 716, Denver, Colorado, November 2013, ACM.

- **P. Alper**, K. Belhajjame C. A. Goble, and P. Karagoz. *Label*Flow: Exploiting Workflow Provenance to Surface Scientific Data Provenance. In Proceedings of the 14th International Annotation and Provenance Workshop (IPAW), pages 84-96, DLR Cologne Germany, June 2014, Springer.

## 7.2 Our Scope in Annotation

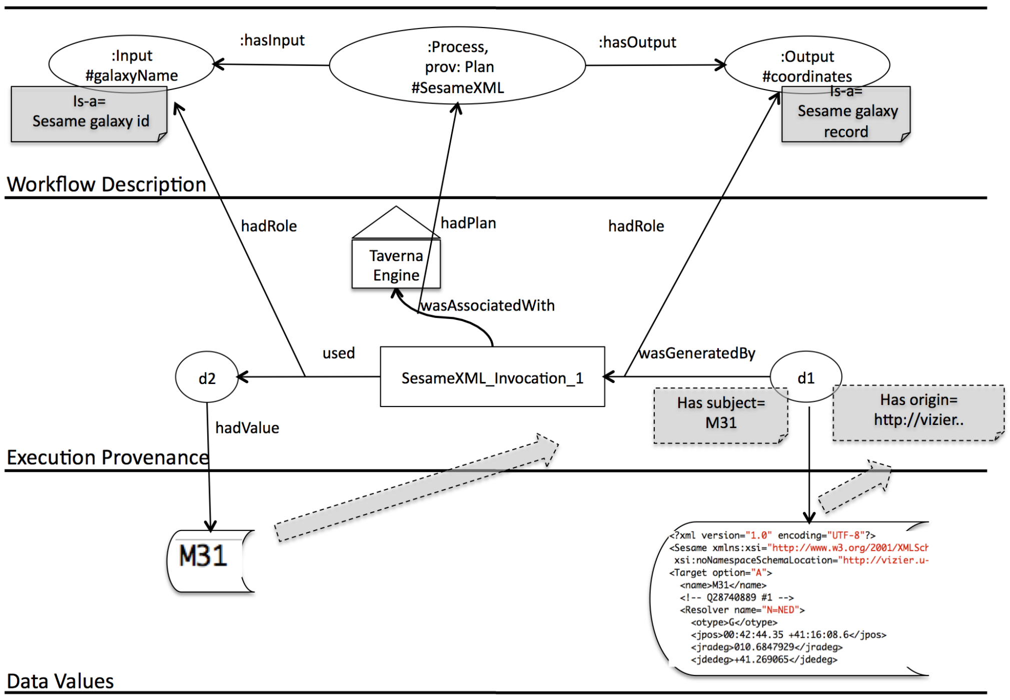
Annotation has been adopted as a solution to bring domain specific description onto provenance, on either side of the provenance gap.

**Annotation of experiment reports.** This is achieved primarily by manual curation, where the scientist uses domain vocabularies, reporting guidelines and tools to create domain specific descriptions upon shared datasets [WOH$^+$12][SKAC14][SRSF$^+$12] [FAJS05] [TFS$^+$08]. The responsibility that the scientist has is recollecting the details of how the analysis was set up and enacted. For this, the scientist sifts through data files, file names, lab notebooks to recollect experimental context and represents this information (often in a structured form) as annotations.

**Annotation of experimental bundles.** Bundles contain resources (data, tools, workflows) used within an experiment to enable its preservation for future re-enactment or audit by external parties. Recalling from Chapter 2, the Research Objects toolkit [BCG+12] Galaxy's Published Histories [GNT10] are examples of bundles. The frameworks that support the creation of bundles also support their annotation. Typically these annotations are intended to create a broader contextual boundary for the experiment. This is achieved by creating groups/aggregates with annotation that bind resources locally found within the bundle with resources outside the bundle (pdf manuscripts hosted on publisher sites, 3rd party analytical resources utilised such as community databases or web services).

**Annotation of workflow provenance.** The importance of domain-specific metadata has been acknowledged early-on in provenance research; The Provenance Challenge, which adopts a workflow-based data processing scenario, provides 9 queries to assess the capabilities of provenance systems [MLA+08]. 5 out of 9 of those queries are based on restrictions on either data values or *annotations*, which are *assumed* to exist. Domain specific annotations on workflow provenance can be the categorised as **Static** and **Dynamic**. This is illustrated in Figure 7.1, where at the top layer we have a fragment of a workflow description (depicting a Taverna processor) in the Wfdesc model. At the middle layer we have the execution provenance for this processor in PROV. For displaying Wfdesc and PROV we utilise the notations described earlier in Section 2.6.3. Specifically Wfdesc is displayed as a "node and directed-arc" RDF graph [WLC14] in compact form where *type* information per node is displayed as bold labels over the node. PROV is displayed as a graph where nodes denote PROV's Activity, Agent and Entity concepts with shape conventions [Gro12]. Other elements of the PROV model, such as usage and generation and association are denoted with arcs. At the data layer in Figure 7.1 we have data values stored within files (denoted with the disk shape) in the file system. We denote domain specific metadata informally with grey-shaded note boxes in Figure 7.1 at both the workflow description and execution provenance layers.

At the workflow description layer we have provided two sample annotations as examples of static metadata, over ports of a processor denoting the domain specific types of information that will appear at those ports. E.g. Specifying that an input parameter is a galaxy id. These represent fixed characteristics of the actual data and actual analysis activities that will appear upon workflow execution. When the workflow is run, activity and data records will be generated at the execution provenance layer. Note that multiple executions of the same workflow will share the same workflow description, and static metadata has validity across all executions. The annotation features in



```
<?xml version="1.0" encoding="UTF-8"?>
<Sesame xmlns:xsi="http://www.w3.org/2001/XMLSch
 xsi:noNamespaceSchemaLocation="http://vizier.u-
<Target option="A">
  <name>M31</name>
  <!-- Q28740889 #1 -->
  <Resolver name="N=NED">
    <otype>G</otype>
    <jpos>00:42:44.35 +41:16:08.6</jpos>
    <jradeg>010.6847929</jradeg>
    <jdedeg>+41.269065</jdedeg>
```

Figure 7.1: Illustration of Static and Dynamic Metadata at different layers of workflow provenance.

workflow systems, we reviewed earlier in Chapter 2, primarily capture static metadata.

At the execution provenance layer we have provided two sample annotations, as examples of dynamic metadata, denoting characteristics of a particular output data generated a via sesame database lookup activity. Dynamic metadata corresponds to attributes of data (or activities) that can change from run to run. For instance depending on the input parameters used to configure the invocation of a sesame database lookup, each result will contain data about a different galaxy. While there is ample support for static metadata in workflow systems there is much less support for dynamic metadata. Only in systems that adopt controlled resource environments like Galaxy and Wings are there capabilities that allow users to manually annotate input datasets, and at workflow run-time some of those annotations are propagated to results. Our PDDS case-study illustrated that because workflow proliferate the analyses through repetitions and data collections, obtaining dynamic annotations manually quickly becomes infeasible.

In the scope of this research we focus on:

- creating dynamic domain specific metadata over workflow provenance,

- creating metadata on fine-grained data nodes in provenance,
- obtaining on metadata that can be provisioned from in situ resources, i.e. values of data nodes in execution provenance.

## 7.3 Revisiting Motifs

Our empirical analysis reported in Chapter 3 showed that workflow activities can be categorised with **Motifs** that denote activity functionality. In our annotation approach we:

- operate with the assumptions that existing workflows and the Motif categorisations allow us to make.
- devise solutions that cater to existing realities of workflows.

We use the characterisation brought by Motifs to determine 1) where in a provenance trace we have data artefacts that could be used to create domain specific metadata and 2) to which data artefacts in the provenance trace that metadata can be extended/propagated.

We revisit the case-study workflow of Chapter 5 here in Figure 7.2 to understand how Motifs can inform annotation. The workflow in Figure 7.2 has been annotated (with callout boxes) to denote Activity Functional Motifs. Table 7.1 lists all Activity Functional Motifs, which can be associated with an annotation behaviour. The scientifically significant activities are hotspots of data that can be used for provisioning domain specific metadata. For our example workflow the steps for obtaining of Galaxy information from external repositories (those having the *Data Retrieval* Motif) and the local extinction calculation (*Data Analysis* Motif) are examples of such hotspots in our case-study workflow. Therefore **metadata generation** can occur using the traces of these steps.

The Motif analysis and also the PDDS case-study illustrated that data's point of creation within a workflow and subsequent use by a follow-on scientifically significant step can be separated by several adapter steps. In Chapter 3 (in Section 3.8.2) we identified that for a subset of adapter Motifs, activity functionality implies certain transparency over the relation between activity inputs and outputs. We identified that adapter steps commonly build their outputs by copying (parts of) inputs, further, for most adapter steps input-output can be viewed as a part-whole relation. We enumerated

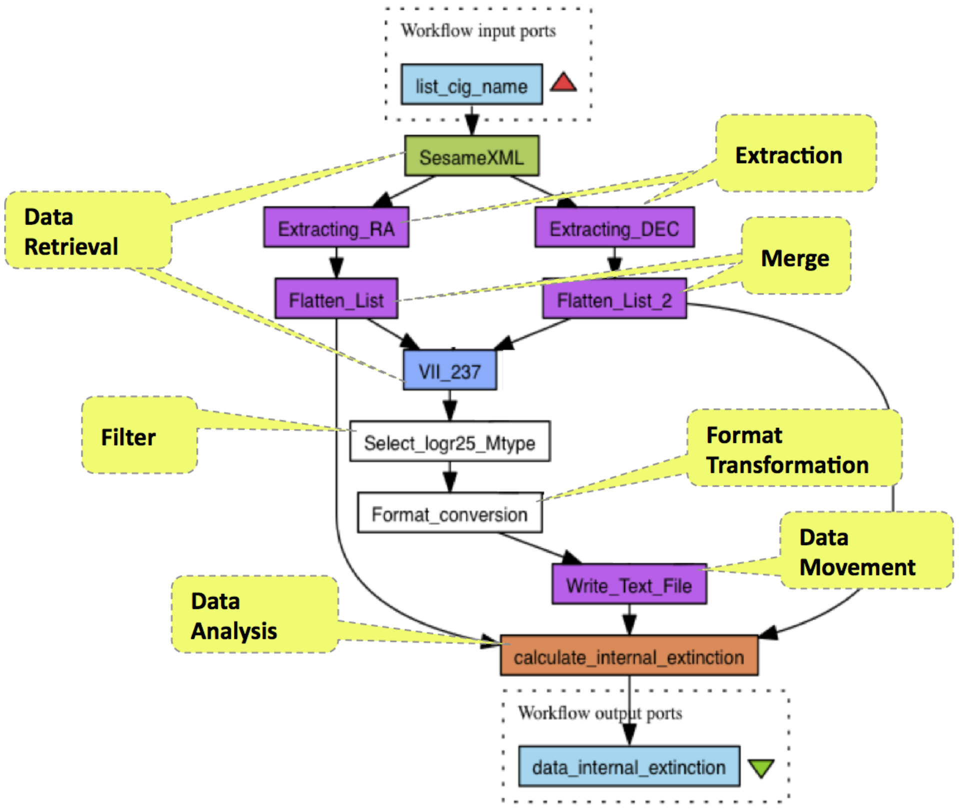


Figure 7.2: Case-Study workflow from Astronomy domain with Motif annotations over activities.

what the implied relations for those value-copying data adapters would be (reiterated here in Table 7.1). We identified that these implied relations convert certain black-box activities into grey-boxes in terms of lineage. We exploit this grey-box transparency to extend the reach of annotations over a data artefact to its copies, through **propagation of metadata**.

We assume that activities both scientifically significant and adapter can be associated with an extended form of Motif annotation, we call Labelling Specification, that can inform the annotation behaviour that shall be taken for that activity. We further make the following assumptions:

- We assume that the ability to create domain-specific metadata out of data values is available in the form of external annotator functions. Therefore we assume the labelling specification for scientifically-significant activities points to the activity's associated annotator function.

- For the adapter steps the labelling specification states among which input and output ports of activity the value copying occurs.

The detailed model for Labelling Specifications and how they can be created by extending Motif annotations has been left out of scope of our research.

Using the information available in workflow provenance, data values, and assumed annotation functions our research seeks answers to the following:

- At what granularity should metadata extraction operate?

- How can we systematically apply metadata extraction functions over workflow provenance traces containing multiple activities connected by dataflow links?

- If we liberally propagated annotations to data copies created by adapters and also among data collections and items, what would the implication be in terms of accuracy of annotations?

## 7.4 Approach Overview

We associate labelling behaviour with activities. This choice is intentional so as to create metadata that captures the experimental context. An activity presents us with a contextual boundary: the activity's inputs are parameters used to obtain the outputs,

| **Motif** | **Value-Copying** | **Example in Figure 7.2** | **Labelling** |
|---|---|---|---|
| `Data Analysis`<br>`Data Retrieval`<br>`Data Visualization` | N/A | “SesameXML”,<br>“VII_237”,<br>“calculate_int_extinction” | generate |
| `Augmentation` | $I \xrightarrow{m-1} O$<br>I *isPartOf* O | Not present in example. | propagate |
| `Extraction` | $I \xrightarrow{1-m} O$<br>I *hasPart* O | “Extract_DEC”,<br>“Extract_RA” | propagate |
| `Split` | $I \xrightarrow{1-1} O$<br>I *hasPart* O | Not present in example. | propagate |
| `Merge` | $I \xrightarrow{1-1} O$<br>I *isPartOf* O | “Flatten_List” | propagate |
| `Filter` | $I \xrightarrow{1-1} O$<br>I *hasPart*O | “Select_logr25_Mtype” | propagate |
| `Combine` | $I \xrightarrow{m-1} O$<br>I *isPartOf* O | Not present in example. | propagate |

Table 7.1: Workflow Motifs, whether they imply Value Copying, and the corresponding Labelling Behaviour

so therefore the inputs represent the context, or the experimental setting for the generation of outputs. Scientifically significant activities in workflows often generate data in standardised data formats from respective domains. The VOTable in Astronomy [OWD+04] or the GBIF format in Biodiversity [gbi15] are just a few examples. These formats commonly carry the context for the data within. For instance, if we query the GBIF repository for a certain species record, the result provided in GBIF format will contain the query, species name, location etc we used in our query. An alternative approach, for us, then could have been associating labelling capability with the ports of a workflow, which we know during execution will carry data in such standardised metadata rich formats. We have opted to operate at the activity level for two reasons. First; standardised formats often contain extensive metadata, most of which do not map to any configuration or input parameter of the workflow. Second; we need a generalised solution that also caters for cases where a data (output) does not carry the context value within itself, but instead another data (input) represents the context for it.

We capture core labelling behaviours with *MINT* and *PROPAGATE* operators. From a high level perspective:

- *MINT* obtains labels by invoking the external labelling function associated with a workflow activity. The label generator function expects all data artefacts that were used and generated by a particular invocation of the activity. Mint operator is responsible for accessing the PROV compliant provenance trace to obtain the inputs and outputs of an activity and forward these to the labelling function. The labelling function will create labels by extracting metadata from data values and return them to the mint operator, which in turn attaches those labels with designated output data artefacts of the activity.
- *PROPAGATE* transfers labels from designated inputs of an activity to designated outputs by creating clones of labels. Note that labels may need to be transferred from multiple input ports to an output port. In this case the propagate operator will create a union set of labels from multiple inputs.

These two operators have been designed with re-usability in mind. They operate over PROV complaint traces, and can potentially be used to decorate the traces of workflows (from different systems) that adhere to the simple **dataflow** model. In such workflows the data derivation structure in the provenance trace is deterministically shaped by dataflow structure in the workflow. Therefore if we enact labelling behaviour associated with workflow steps in the (topological) order they appear in the workflow DAG, then we can effectively decorate the entire trace.

In addition, we provide two Taverna specific operators also for label propagation, namely the *DISTRIBUTE* and *GENERALISE* operators. These have been designed to maximise the reach of labels in a provenance trace. These operators cater for the collection-oriented nature of data, and Taverna's ability to compose activities with mismatched structured data types, which may result in iterated and wrapped compositions discussed earlier in the Chapter. While the *MINT* and *PROPAGATE* operators are associated with activities, *DISTRIBUTE* and *GENERALISE* are associated with dataflow links that link-up two ports, which by definition produce and consume data of mismatching structured types. In cases where the activity at one end of a dataflow link produces a collection, and the other end consumes an item, *DISTRIBUTE* is responsible for propagating labels from the top-level collection to each item at specified depth. In cases where the activity at one end produces individual items in a collection, and the other end consumes the collection *GENERALISE* is responsible for propagating labels from items to the enclosing collection at a specified depth.

Before introducing operators in detail we describe the Label model we adopt for carrying domain-specific metadata.

### 7.4.1 Label Model

The information model for labels is given in the UML [BRJ05] class diagram in Figure 7.3. Formally;

**Definition 7.1.** (Label) A label is an information object that is an instance of class *LabelInstance* depicted in Figure 7.3. Labels bear a *definition*, *target* and a *value* attribute. The *target* and *value* both of type *String* represent a simple key-value based metadata structure. The *target* uniquely identifies a data artefact, which the label describes and the *value* holds the annotation content. A label refers to its kind through the *definition* attribute, which is of type *LabelDefinition*.

**Definition 7.2.** (Label Definition) A label definition is an information object that is an instance of class *LabelDefinition* depicted in Figure 7.3. A label definition bears a *name* attribute, of *String* type, which uniquely identifies a particular label kind.

Note that the only information that is known by operators about labels when transporting them is their definition. Otherwise operators are oblivious to the actual metadata (*value*) carried by labels. Operators use label definitions when gathering labels from multiple sources prior to propagation for computing label equality. A label's identity is determined by a combination of its definition and value hash. This identity information is used when aggregating multiple labels through a *Union* function. We have kept *LabelDefinition* as a separate class to cater for extensibility. For the purposes of this dissertation labels can have *String* values only and we use set *Union* for aggregating them. In future extensions other data types for label values (e.g. *Integer* or *Date*), and other operations for aggregation can be supported by additional attributes in the *LabelDefinition* class.

**Definition 7.3.** (Label Vector) A Label Vector is an information object that is an instance of class *LabelVector* depicted in Figure 7.3. A label vector bears a *name* attribute of type *String* and a collection of label definitions that comprise the vector.

We use label vectors to configure the execution of labelling operators. A label vector informs label propagation operators to the kinds of labels they should pick up from parts of a provenance graph and propagate to other parts. Label and label vector definitions would be specific to each scientific domain or investigation, and can be used to decorate workflows from these domains. A label vector could be seen as a primitive localised metadata profile akin to data tagging profiles adopted in various scientific domains [HCW08] [gbi15]. In the following section we detail our labelling framework, *Label*Flow and its constituent operators.

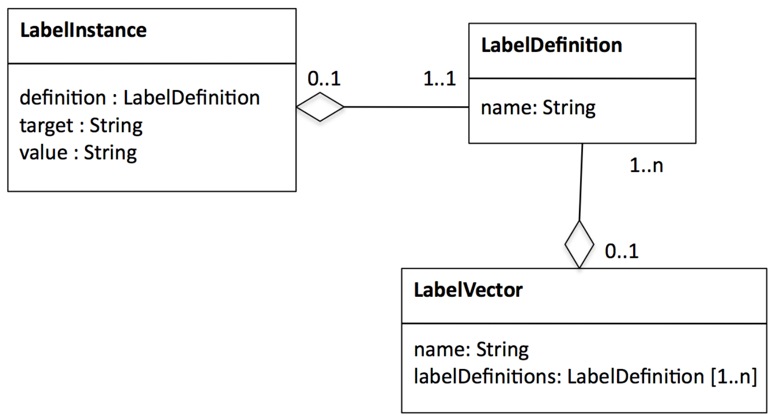


Figure 7.3: UML Class Diagram denoting information model of Labels.

## 7.4.2 *Label*Flow Framework

*Label*Flow is a generic framework, which requires configuration for use in a particular domain. Formally;

**Definition 7.4.** *Label*Flow is the tuple $< O, T >$, where:
$O = \{MINT, PROPAGATE, GENERALISE, DISTRIBUTE\}$ is the set of labelling operators.
$T$ is a tool that can take as input a scientific workflow $w$ and generate a labelling pipeline for that workflow $\Pi_w$.

**Definition 7.5.** A particular configuration of *Label*Flow is the 8-tuple
$< O, T, F, w, \Pi_w, P_w, D_w, L_w >$ where in addition to the above definitions:
$F$ is a set of domain-specific labelling functions.
$w$ is a Motif annotated scientific workflow.
$\Pi_w$ is the labelling pipeline for $w$.
$P_w$ is a provenance store that contains a particular execution of $w$.
$D_w$ is a data store that contains data artefacts generated during a particular execution of $w$.
$L_w$ is an initially empty label store that supports the information model given in Section 7.4.1 to hold labels generated during execution of $\Pi_w$ .

In order to describe the behaviour of the labelling operators and labelling pipelines that are comprised of those operators we use UML Activity diagram notation [BRJ05].

Figure 7.2 provides a subset of elements from this notation and their definitions as per UML reference model.

We will now describe $MINT$, $PROPAGATE$, $GENERALISE$, $DISTRIBUTE$ operators in detail. We adopt the following presentation method: (1) We give a high-level behavioural view of operators as UML activity diagrams. (2) Each activity involves an atomic action that is realised by a call to corresponding labelling operations (concrete implementation of the labelling behaviour). We provide signatures for these labelling operations using UML's operation syntax. (3) We give the procedural specification of each operation as algorithms. (4) We describe the auxiliary methods utilised by algorithms to access the provenance, data and label stores, also using UML's operation syntax.

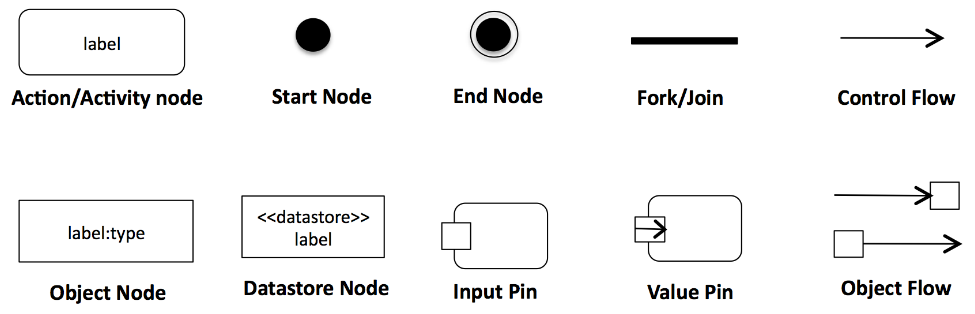
**Definition 7.6.** (MINT Operator) is a computational process, whose behaviour is given in Figure 7.4 (a) as a UML activity that accepts as input a *processorId* and a $functionId$, which are of type $String$, and a $targetList$ which is a set of $String$s. The *processorId* is the identifier of a (scientifically significant) processor [1], whose provenance trace and associated data artefacts will be exploited to provision labels. The $functionId$ is the identifier for the domain-specific label provisioning function. $targetList$ contains identifiers of those output ports of the designated processor, that will be the target of labels provisioned.

All inputs to the $MINT$ activity, as seen in Figure 7.4 (c), are forwarded to the $Mint$ action, which is realised by an operation call with the interface given in Table 7.3. In addition to inputs this action reads data store $D_w$ and provenance trace $P_w$, and upon execution submits the labels it generates to the label store $L_w$. The procedure that underpins the $Mint$ action, provided in Algorithm 2, is as follows: we obtain all invocation records of the processor designated by the *processorId*, for each invocation record we obtain all data related to that invocation (inputs/ outputs)[2]. We submit data to the labelling function named $functionId$. The function returns a set of labels that are to be associated with the target outputs, finally we associate these labels with all data artefacts that have appeared at a port in the $targetList$.

**Definition 7.7.** (PROPAGATE Operator) is a computational process, whose behaviour is given in Figure 7.4 (b) as a UML activity that accepts as input a *processorId* of type $String$, and a $srcList$ and a $targetList$ which are sets of $String$s. The *processorId* is

[1] In workflow $w$, for which *Label*Flow has been configured.

[2] As Taverna uses the file system for its data storage layer, these are references to files.



An activity diagram is a graph of nodes denoting a process comprised of steps of computation and flows of (primarily) control (and optionally) data among steps.

An action/activity node (rounded rectangle) denotes a computational step. An action is an atomic step which is not further broken into sub-steps, whereas an activity is a group of actions or sub-activities.

Start node (solid circle) is a control node at which flow starts when an activity is invoked.

End node (hollow circle with solid circle inside) is a control node that stops all flows in an activity.

Fork node (thick line segment) is a control node that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple concurrent flows. Join node is a control node that has multiple incoming edges and one outgoing edge and is used to synchronise incoming concurrent flows.

Control flow edge (arrow) is an edge denoting flow of control from one activity to another.

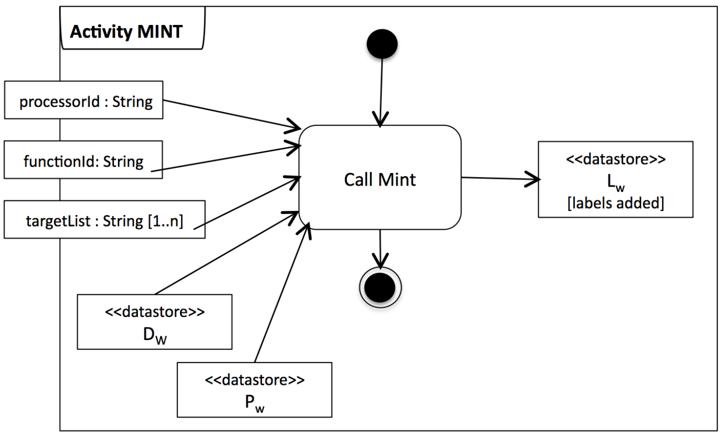
Object node (rectangle) is an edge denoting flow of data from one activity to another.

A data store nodes (rectangle) are stereotyped object nodes, which denote non-transient data that is persisted during the computational process.

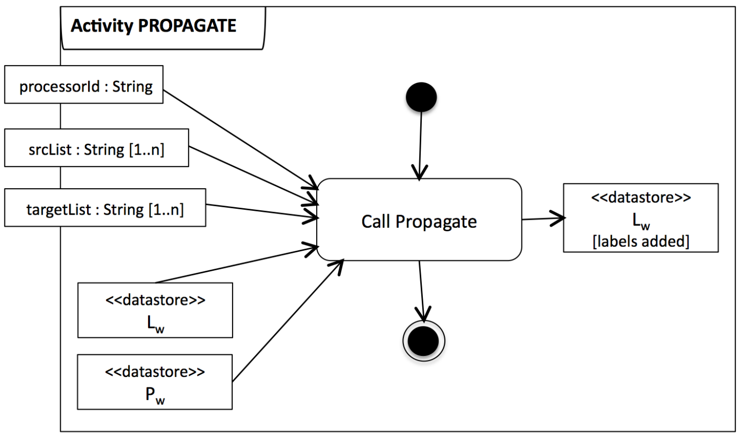
Object flow edge (arrow) is an edge denoting flow of data during a computational process. An object flow edge is one that connects two nodes, where at least one is an object node. A value pin is special kind of input pin defined to provide constant values as input.

Pins (small rectangle at edge of rounded rectangle) are object nodes used to denote inputs/outputs to activities. A value pin is a special kind of input pin, which denotes constant-valued inputs to an activity.

Table 7.2: UML Activity Diagram elements notation and definitions.



(a) MINT



(b) PROPAGATE

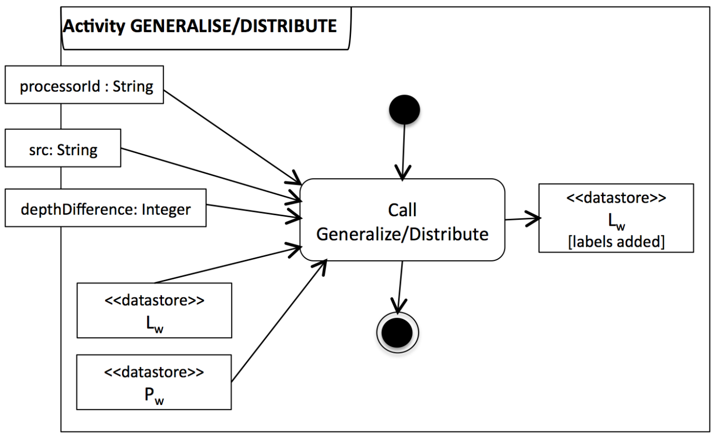

Activity GENERALISE/DISTRIBUTE
processorId : String
src: String
depthDifference: Integer
<<datastore>> Lw
<<datastore>> Pw
Call Generalize/Distribute
<<datastore>> Lw [labels added]
(c) GENERALISE/DISTRIBUTE

Figure 7.4: UML Activity Diagrams denoting behaviour of Labelling Operators.

| |
|---|
| `Mint(processorId:String, functionId:String, targetList:String[1..n])` |
| `Propagate(processorId:String, srcList:String[1..n], targetList:String[1..n])` |
| `Generalise(processorId:String, src:String, depthDifference:Integer)` |
| `Distribute(processorId:String, src:String, depthDifference:Integer)` |

Table 7.3: Signatures for labelling operations.

**Algorithm 2** Procedural Specification of Mint Operation.

```
procedure $Mint(processorId, functionId, targetList)$
    $labellingFunction \leftarrow getFunctionFromRegistry(functionForPro)$
    for each $activity$ in $getInvocations(processorId, provStore)$ do
        $activityData \leftarrow getAllActivityData(activity, provStore, dataStore)$
        $Labels \leftarrow labellingFunction.invoke(activityData)$
        for each $output$ in $targetList$ do
            $outData \leftarrow getActivityOutData(activity, output)$
            $BoundLabels \leftarrow bindLabelsToData(outData, Labels)$
        end for
        $submitLabels(BoundLabels, labelStore)$
    end for
end procedure
```

the identifier of an (adapter) processor, which has a $Motif$ implying a value copying relation from its inputs to its outputs (recall Table 7.1). The *srcList* contains identifiers of those input ports of the designated processor, from which labels are to be picked up. The $targetList$ contains identifiers of output ports to which labels shall be propagated.

All inputs to the $PROPAGATE$ activity, as seen in Figure 7.4 (b), are forwarded to the $Propagate$ action, which is realised by an operation call with the interface given in Table 7.3. In addition to inputs this action reads the label and provenance stores $L_w$ and $P_w$ and updates the label store $L_w$ with propagated labels. The procedure that underpins the $Propagate$ action, provided in Algorithm 3 is as follows: the invocation record of processors with designated *processorId* are obtained, for each invocation record we obtain the labels of data nodes at a source port in the *srcList* , we aggregate them with set $Union$ and , finally we associate these labels with all data artefacts that have appeared at a port in the $targetList$.

**Algorithm 3** Procedural Specification of Propagate Operation.

```
procedure $Propagate(processorId, srcList, targetList)$
    for each $activity$ in $getInvocations(processorId, provStore)$ do
        $LabelDefs \leftarrow getLabelVector()$
        for each $output$ in $targetList$ do
            for each outData in $getActivityOutData(activity, output, provStore)$ do
                $Labels \leftarrow \varnothing$
                for each $input$in $srcList$ do
                    for each inData in $getActivityInData(activity, input, provStore)$ do
                        $Labels \leftarrow Labels \cup getLabels(inData, LabelDefs, labelStore)$
                    end for
                end for
                $CloneLabels \leftarrow clone(Labels)$
                $BoundLabels \leftarrow bindLabelsToData(outData, CloneLabels)$
            end for
            $submitLabels(BoundLabels, labelStore)$
        end for
    end for
end procedure
```

**Definition 7.8.** (DISTRIBUTE/GENERALISE Operators) are computational processes, whose behaviour is given jointly in Figure 7.4 (c) as a UML activity that accepts as input a *processorId* and an *src* of type *String* and a $depthDifference$ which is of type *Integer*. $DISTRIBUTE$ / $GENERALISE$ operators are designed to propagate labels up and down the structure hierarchy of collection-typed data artefacts in provenance. While the $MINT$ and $PROPAGATE$ are labelling proxies for processors, these are labelling proxies for dataflow links in the workflow, specifically those links with data type structure mismatches between the ports at two ends of the datalink. The *processorId* and *src* parameters jointly identify an output port of a particular processor (the source end of a mismatched datalink). The level of mismatch is specified with the $depthDifference$.

Consider the case where one processor by definition produces an output of type $L^2(s)$ and is linked to a follow-on processor that consumes input of type $s$. This case corresponds to a $depthDifference$ of 2 among two ends of a dataflow link. In order to adjust for this mismatch we would have to push down the labels associated with the output collection occurring at port *src* of the designated processor to the collection's items that are two-level deep in the data structure. We achieve this by using the $DISTRIBUTE$ operator. The reverse procedure of pulling up labels is performed by the *GENERALISE* operator. All inputs to the $DISTRIBUTE$ / *GENERALISE* activity, as seen in Figure 7.4 (c), are forwarded to the corresponding action, which is realised by an operation call with interface(s) given in Table 7.3. Note that similar to the $PROPAGATE$ operator these processes read the label and provenance stores $L_w$ and $P_w$ and update the label store $L_w$. The procedures that underpin the $Distribute$ and $Generalise$ actions are given in Algorithm 4 and 5 respectively. Note that when generalising, we gather labels from multiple child items, we therefore perform set union on those labels prior to associating with a parent collection.

**Algorithm 4** Procedural Specification of Distribute Operation.

```
procedure $Distribute(processorId, src, depthDifference)$
    $LabelDefinitions \leftarrow getLabelVector()$
    for each outData in $getAllGeneratedOutputs(processorId, src, provStore)$ do
        $Labels \leftarrow getLabels(outData, LabelDefinitions, labelStore)$
        for each item in $getItems(outData, depthDifference, provStore)$ do
            $CloneLabels \leftarrow clone(Labels)$
            $BoundLabels \leftarrow bindLabelsToData(item, CloneLabels)$
            $submitLabels(BoundLabels, labelStore)$
        end for
    end for
end procedure
```

## 7.4.3 Labelling Pipelines

In order to put the capability encapsulated by operators into action we use labelling pipelines. We formally define the elements of pipeline generation as follows:

| |
|---|
| `getInvocations(processorId:String, provStore:String):String[0..n]` Obtains identifiers of all the *PROV* : *activity* nodes in the trace that are documented to have occurred using *processorId* as a *PROV* : *plan*. |
| `getAllGeneratedOutputs(processorId:String, port:String, provStore:String):String[0..n]` Obtains identifiers of all the *PROV* : *entity* (data) nodes in the trace that have been in a qualified *PROV* : *generation* relationship with some activity, where the activity has occurred according to a *PROV* : *plan* of identifier *processorId* and the generated data had role (*PROV* : *hadRole*) *port*. |
| `getAllActivityData(activityId:String, provStore:String) :String[0..n]` Obtains identifiers all the *PROV* : *entity* (data) nodes in the trace that have been in a *PROV* : *usage* or a *PROV* : *generation* relationship with the designated *activityId*. |
| `getActivityOutData(activityId:String, port:String, provStore:String):String[0..n]` Obtains identifiers of the *PROV* : *entity* (data) nodes in the trace that are in a *PROV* : *generation* relation with the designated *activityId*, where the generation is qualified stating that the data node played the role (*PROV* : *hadRole*) identified with *port*. |
| `bindLabelsToData(dataId:String, labels:LabelInstance[0..n]) :LabelInstance[0..n]` Returns a copy of the labels, where the target of each copy is set to the designated data record. |
| `clone(labels:LabelInstance[0..n]):LabelInstance[0..n]` Creates a copy of all the labels in the input set . |
| `submitLabels(labels:LabelInstance[0..n], labelStore:String)` Stores all the label instances in the designated label space. |
| `getItems(coll:String, depthDifference:Integer, provStore:String):String[0..n]` Obtains identifiers of *PROV* : *entity* (data) nodes in the trace that are contained (*PROV* : *hadMember*) by the designated *PROV* : *Collection coll* at *depthDifference* level deep. |
| `getLabels(item, labelDefinitions, labelStore:String) :LabelInstance[0..n]` Obtains all the labels, whose target is the designated item. |
| `getEnclosingCollections(items:String[0..n], depthDifference:Integer, provStore:String):String[0..n]` Obtains the identifiers of *PROV* : *Collection* nodes in the trace, which at *depthDifference* level deep contain (*PROV* : *hadMember*) the designated items. |

Table 7.4: Procedures utilised by labelling operators to access and update the provenance, data and label spaces.

**Algorithm 5** Procedural Specification of Generalise Operation.

```
procedure $Generalise(processorId, src, depthDifference)$
    $LabelDefinitions \leftarrow getLabelVector()$
    $OutData \leftarrow getAllGeneratedOutputs(processorId, src, provStore)$
    for each coll in $getEnclosingCollections(OutData, depthDifference, provStore)$ do
        $Labels \leftarrow \varnothing$
        for each item in $getItems(coll, depthDifference, provStore)$ do
            $Labels \leftarrow Labels \cup getLabels(item, LabelDefinitions, labelStore)$
        end for
        $CloneLabels \leftarrow clone(Labels)$
        $BoundLabels \leftarrow bindLabelsToData(coll, CloneLabels)$
        $submitLabels(BoundLabels, labelStore)$
    end for
end procedure
```

**Definition 7.9.** (Annotated Workflow) In Definition 6.2 of Chapter 6, a workflow was defined with the triple $w =< PRO, POR, LINK >$, where $PRO$ denoted the set of processors, $POR$ the set of ports, and $LINK$ the set of dataflow links among ports. We extend this definition to represent an annotated workflow as $w =< PRO_{ann}, POR, LINK >$ where $PRO_{ann}$ denotes a set of annotated processors. Each processor $p_{ann} \in PRO_{ann}$ is denoted with $p_{ann} =< id, m_p >$, where $m_p$ denotes the annotation, i.e. the Labelling Specification for processor $p_{ann}$.

**Definition 7.10.** (Labelling Specification) is an information object that is an instance of concrete classes given in Figure 7.5. Note that among these classes *MintSpec* and *PropagateSpec* encapsulate information passed as input to $MINT$ and $PROPAGATE$ operators given earlier (Figure 7.4), whereas the $AdjustSpec$ encapsulates inputs to $Generalise$ and $Distribute$ operators. A labelling specification $m_p$ of an annotated processors $p_{ann} \in PRO_{ann}$ in a workflow $w$ can be of types $MintSpec$ or $PropagateSpec$ only.

**Definition 7.11.** (Labelling Pipeline Generator) $T$ is a tool provided as part of *Label*Flow that accepts as input an annotated workflow $w$ and produces as output a labelling pipeline $\Pi_w$ for $w$.

**Definition 7.12.** (Labelling Pipeline) is a specification for a computational process comprised of (1) sub-processes based on calls to $MINT$, $PROPAGATE$, $GENERALISE$ and $DISTRIBUTE$ operators (as per Definitions 7.6, 7.7, 7.8) and (2) control-flow relations among those processes. So $\Pi_w =< OP, CTRLINK >$

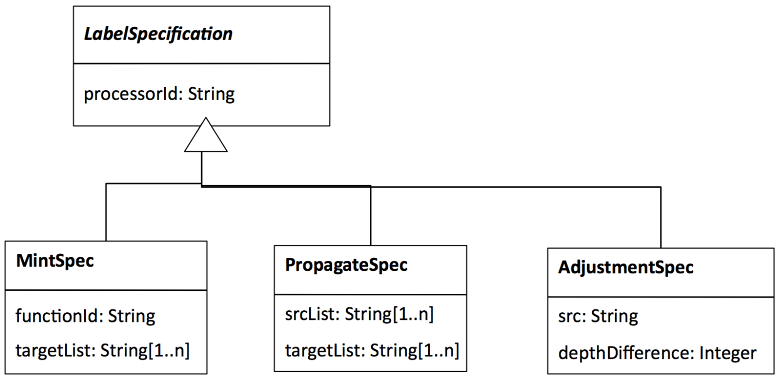


Figure 7.5: UML Class Diagram denoting information model of Labelling Specifications.

We will first illustrate labelling pipelines and later discuss how pipeline generator works.

#### 7.4.3.1 Example Labelling Pipeline

For the case-study workflow given earlier in Figure 7.2 the labelling pipeline is given in Figure 7.6 using UML Activity Diagram notation. For each of the three scientifically significant activities in the workflow, namely *SesameXML*, *VII_237* and *calculate_ internal_ extinction* (in Figure 7.2) there is a corresponding *MINT* process in the labelling pipeline (in Figure 7.6). This is because all three processors had associated labelling specifications (extended Motif annotations) of type *MintSpec* (in Figure 7.5). Label specifications of these processors have become sets of constant-valued inputs (denoted with value-pins ) for each corresponding *MINT* process in the pipeline. For example for the *SesameXML* step in the workflow, the corresponding *MINT* process is configured with:

- String value of "SesameXML" for the *processorId* input parameter.
- String value of "SesameLabeller" for *functionId* input parameter,
- a Set containing the String value "return" for *targetList* input parameter.

Using this input triple the *MINT* process is undertaken by calling the operation detailed in Algorithm 2, which will decorate data outputs that appear at the port named "return" of processor "SesameXML" with labels obtained through invocation of domain-specific function "SesameLabeller". Recall from the specification of *MINT* process (in Figure 7.4) that that it reads from data and provenance stores $D_w$ and $P_w$ and writes to the label store $L_w$. And, unlike all other operators, *MINT* does not read from the label store as it generates labels in the first place. As a result *MINT* processes can start simultaneously upon the start of labelling pipeline (denoted with a fork of control links from start node to all three *MINT* processes).

For some of the data adapter steps in our case-study workflow, namely *Extract_RA*, *Extract_DEC*, *Select_logr_Mtype*, *Flatten_List*, *Flatten_List_2*, we have *PROPAGATE* processes in the labelling pipeline. The labelling specifications, of type *PropagateSpec*, associated with these adapter processors, has become input configurations for the *PROPAGATE* processes in the pipeline. The labelling specifications denote from which input ports (*srcList*) to which output ports (*targetList*) label propagation should occur. Note that the *Format_ conversion* step in our workflow, despite being a data adapter having the *FormatTransformation* Motif, does not have a corresponding labelling process in the pipeline. This is because, as per Table 7.1, *FormatTransformation* is not a Motif for which a labelling behaviour has been defined. Consequently *Format_ conversion* step does not have an associated labelling specification and therefore has no footprint in the labelling pipeline. As per its specification (in Figure 7.4) the *PROPAGATE* process reads from and write to the label store $L_w$. In order for a *PROPAGATE* process to run, all other processes in the pipeline that decorate data at ports in the *srcList* parameter of that *PROPAGATE* process shall be completed. This requirement is represented with control flow links among relevant processes in the pipeline.

The pipeline in Figure 7.6 also contains a *GENERALISE* and *DISTRIBUTE* processes to propagate labels along data structure hierarchies in cases of mismatched data types at the two ends of a dataflow link. One example is the *GENERALISE* process, which is configured to propagate labels of the *nodeList* output of *Extract_RA* processor one level up to their enclosing collection, as it is these collections that get consumed by the follow-on processor *Flatten_List* in the workflow. There is a difference in the way *GENERALISE* / *DISTRIBUTE* processes are included in a labelling pipeline when compared to the way *MINT* and *PROPAGATE* processes are included. *MINT* and *PROPAGATE* processes are directly informed by annotations in the form of labelling specifications (either a *MintSpec* or a *PropagateSpec*) associated with processors in the workflow. On the other hand there is no such annotation for the *GENERALISE* / *DISTRIBUTE* processes. Their inclusion happens through an analysis of dataflow links in the workflow and the corresponding creation of labelling specifications of type *AdjustmentSpec* (Figure 7.5). We discuss the details of labelling pipeline creation in the next section.

#### 7.4.3.2 Pipeline Generation Procedure

The procedure followed by the Labelling Pipeline Generator is given in Algorithm 6. Inputs $wfProcessors$ and $wfDatalinks$ are the set of Processors and Datalinks that make up a workflow $w$. The procedure initialises two empty collections$pipelineOps$ and *pipelineCtrlLinks* to hold the Labelling Operators and the Control Links within the result pipeline $\Pi_w$. The procedure is comprised of two phases for populating these two collections.

The first phase begins by traversing all processors of $w$ to check whether that have associated with them a labelling specification. If that is the case then the labelling specification is transferred to $\Pi_w$, more specifically it will be added to the$pipelineOps$ collection. In case a processor in the workflow has no labelling specification associated, then it will simply be skipped. As a follow-on step we eliminate dangling $PROPAGATE$ operators. Dangling operators are those that are configured to obtain labels from source ports, where no labelling operator is configured to populate. The final step in creation of labelling operators is the addition of *GENERALISE* or $DISTRIBUTE$ type adjustment operators. We do this by iterating over every datalink in workflow $w$ (items of $wfDatalinks$). We check whether the source of the datalink is being labelled by any of the $MINT$ or $PROPAGATE$ type operators. If that is the case, and if the datalink is one which is unbalanced due to mismatched datatypes of ports at its two ends then we create a corresponding adjustment specification either $DISTRIBUTE$ or *GENERALISE* and add it into the$pipelineOps$ collection.

In the second phase we create control flow links. We do this by iterating over operators in the pipeline, finding each the operator's predecessor operators and creating control links among them. The predecessors of a $PROPAGATE$ operator can be multiple and they are those that have as their labelling target a port that is in the source port list of $PROPAGATE$. Adjustment type operators *GENERALISE* or $DISTRIBUTE$ have a single predecessor, which is the one that has as labelling target the source port of adjustment operator.

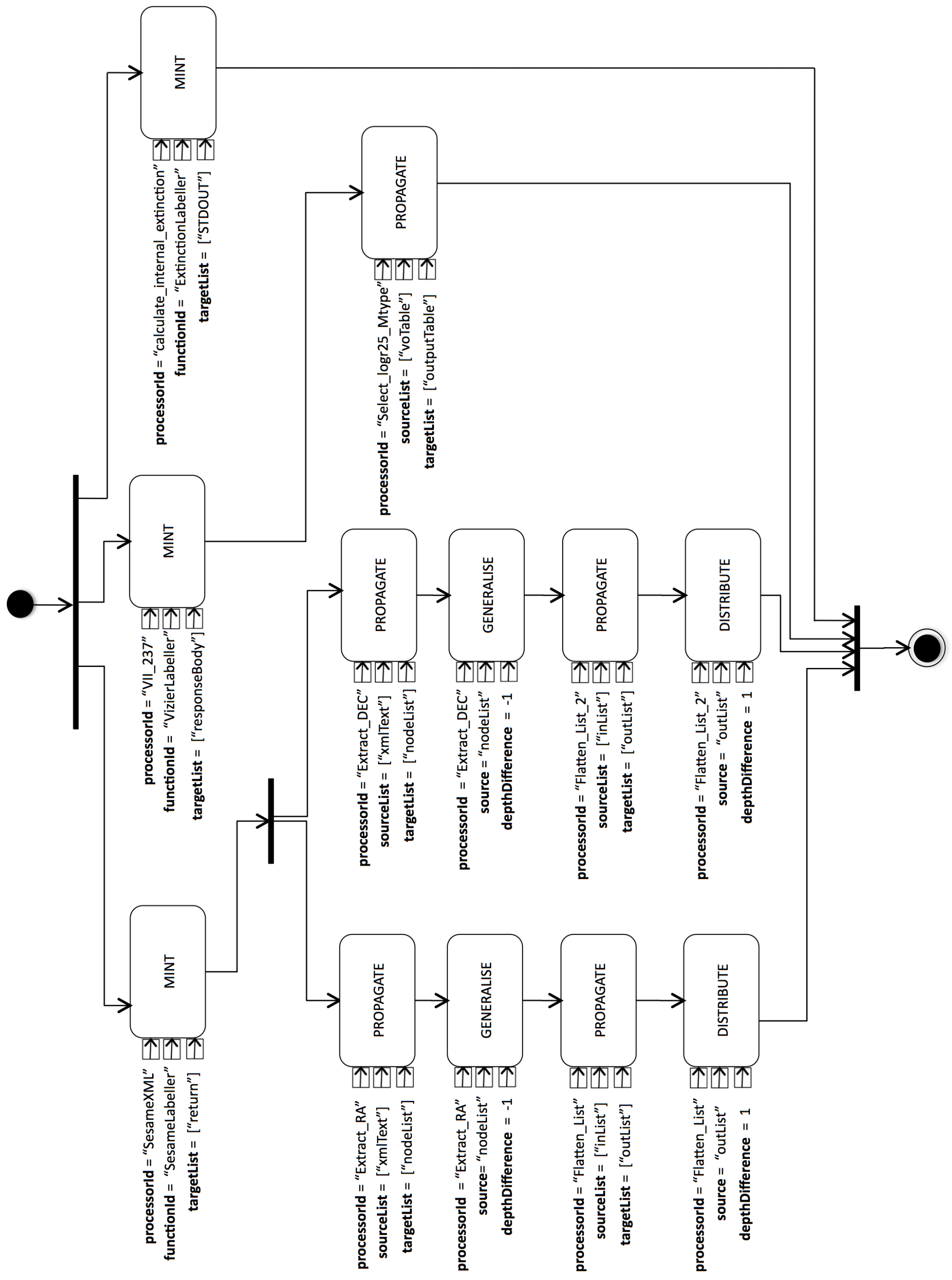

MINT
processorId = "calculate_internal_extinction"
functionId = "ExtinctionLabeller"
targetList = ["STDOUT"]
MINT
processorId = "VII_237"
functionId = "VizierLabeller"
targetList = ["responseBody"]
PROPAGATE
processorId = "Select_logr25_Mtype"
sourceList = ["voTable"]
targetList = ["outputTable"]
MINT
processorId = "SesameXML"
functionId = "SesameLabeller"
targetList = ["return"]
PROPAGATE
processorId = "Extract_DEC"
sourceList = ["xmlText"]
targetList = ["nodeList"]
GENERALISE
processorId = "Extract_DEC"
source = "nodeList"
depthDifference = -1
PROPAGATE
processorId = "Flatten_List_2"
sourceList = ["inList"]
targetList = ["outList"]
DISTRIBUTE
processorId = "Flatten_List_2"
source = "outList"
depthDifference = 1
PROPAGATE
processorId = "Extract_RA"
sourceList = ["xmlText"]
targetList = ["nodeList"]
GENERALISE
processorId = "Extract_RA"
source= "nodeList"
depthDifference = -1
PROPAGATE
processorId = "Flatten_List"
sourceList = ["inList"]
targetList = ["outList"]
DISTRIBUTE
processorId = "Flatten_List"
source = "outList"
depthDifference = 1

Figure 7.6: The labelling pipeline for the case-study workflow.

We represent the labelling pipeline from this procedure with the Wfdesc workflow model [BZG+15]. Note that the pipeline is comprised of operators and control flow links. The repetitive application of labelling for multiple invocations of processors and for multiple label kinds in a label vector are handled within the labelling operators. Therefore the basic model of Wfdesc in representing processes and their dataflows is sufficient for us in representing our pipeline. The details of how this abstract representation is mapped to a concrete executable form is provided in the following Section.

## 7.5 Implementation

The auxiliary functions we use to access the provenance store (given in Table 7.4) supports a PROV-O [BCC+12] based RDF representation of workflow execution provenance. Consequently we have chosen an RDF based representation of labels to facilitate their integration with provenance. Each label definition is represented with an OWL Datatype property. Label instances are RDF statements where the subject corresponds to the target of the label, the predicate is the datatype properties and objects are metadata values of type $xsd:string$.

The labelling operators and the auxiliary provenance access functions are implemented as Java API methods. The labelling pipelines are represented in an abstract manner with Wfdesc. This abstract representation can be converted to a concrete executable form using any workflow language that supports a simple data flow among activities, and can access resources exposed through Java APIs. For our case-study tests we interpret the abstract Wfdesc specification by traversing the activities in the topological order of their respective workflow elements in the scientific workflow, and make API calls to invoke respective operators. Note that the pipeline is only responsible for coordinating the execution of labelling operators, whereas the core of labelling takes place within operators.

## 7.6 Revisiting case-study

In order to assess the utility of labels in PDDS we have used *Label*Flow to annotate execution traces of our case-study workflow. As prerequisite to obtaining a labelling pipeline for this workflow we performed the following:

- we implemented three simple domain-specific labelling functions, one for each scientifically significant step in the workflow (as discussed in Section 7.4.3.1).

**Algorithm 6** Procedure followed by the pipeline generator tool.

```
procedure $convert(wfProcessors, wfDatalinks)$
    $pipelineOps \leftarrow \varnothing$
    $pipelineCtrlLinks \leftarrow \varnothing$
    ▷ PHASE 1: Create Labelling Operators
    for each processor in wfProcessors do
        if $hasLabellingSpec$(processor) then
            $pipelineOps \leftarrow getLabellingSpec(processor)$
        end if
    end for
    do
        $found \leftarrow false$
        $danglingOperator \leftarrow null$
        for each op in pipelineOps do
            if $isPropagate$(op) and $isDangling$(op) then
                $danglingOperator \leftarrow op$
                $found \leftarrow true$
                break
            end if
        end for
        if found then
            $remove(pipelineOps, op)$
        end if
    while $found$
    for each link in wfDatalinks do
        if $isLinkSourceLabelled$(link, pipelineOps) and $isImbalanced$(link) then
            $pipelineOps \leftarrow createAdjustmentSpec(link)$
            break
        end if
    end for
    ▷ PHASE 2: Create Control Links
    for each op in pipelineOps do
        if $isPropagate$(op) then
            for each src in $getSrcList$(op) do
                $predecessorOp \leftarrow getOperationWithTarget(src)$
                if predecessorOp <> null then
                    $pipelineCtrlLinks \leftarrow createCtrlLink(predecessorOp, op)$
                end if
            end for
        else if $isGeneralise$(op) OR $isDistribute$(op) then
            $predecessorOp \leftarrow getOperationWithTarget(getSrc(op))$
            $pipelineCtrlLinks \leftarrow createCtrlLink(predecessorOp, op)$
        end if
    end for
end procedure
```

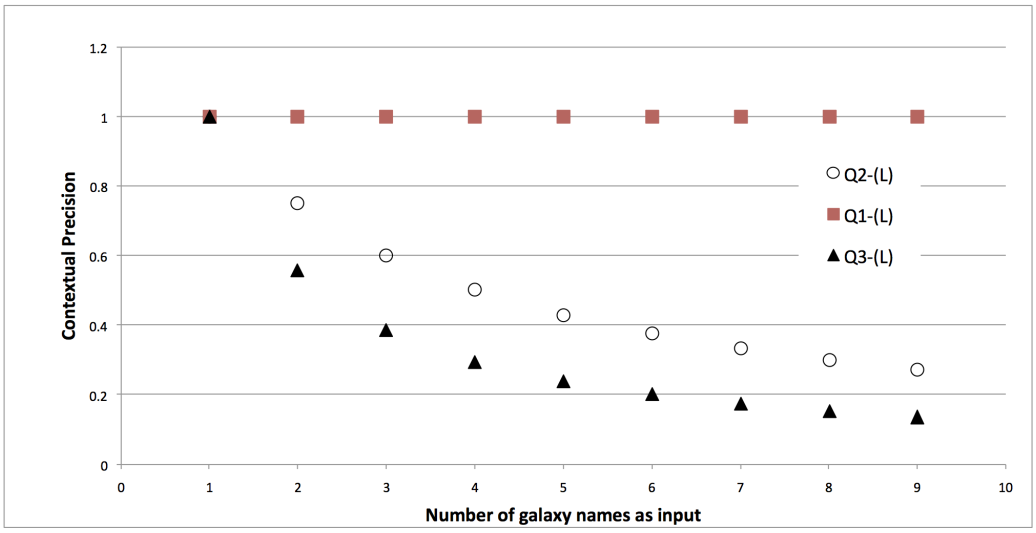


Figure 7.7: Contextual-Precision of label-based data selection queries.

These functions can parse the data consumed/generated by these activities and create labels that correspond to either input configurations (**context**) or **data origin**.

- we associated Labelling Specs with workflow activities according to the information model given in Figure 7.5. For the data adapter activities, for which a corresponding labelling behaviour is given (Table 7.1), we created *PropagateSpec*s and for the scientifically significant activities, we created *MintSpec*s, pointing to the labelling functions.

This time we implement the queries in Table 5.1 using labels (hence denoted with the "-L" suffix). The precision in obtaining relevant results for each query is given in Figure 7.7.

**Q1-L** Rather than using lineage as a pseudo mechanism to seek coordinates retrieved from Sesame Database, we now inquire about data origin directly using labels. We use $case:referenceURI$, $case:referenceCatalog$ datatype properties created for test purposes. Note we are now able to fully implement the query and seek the source catalog information about the coordinates. As Figure 7.7 shows, with label-based queries we are able to retrieve with 100% accuracy the data that comes from the Sesame database and its local copies.

**Q2-L** In this query we use the $case:hasSubject$ label to seek results about a particular Galaxy. Note that a typical characteristics of scientific data repositories is that they use heterogeneous identification schemes. So, in Astronomy a Galaxy has several identifiers from respective databases. The Visier and Sesame databases accesses within our example workflow use different identifiers. Therefore in our query, we need to refer to all possible identifiers of a Galaxy. As seen from Figure 7.7, the precision deteriorates as it was the case when the query was implemented using lineage. A combination of broken factorial design (at the *Flatten_List* step) and liberal label propagation causes inaccurate labels to be created. While each output from "SesameXML" bears the correct label denoting the associated galaxy, all items in the output of "Flatten_List" would bear a set of labels (for all galaxies), even though each contains the data of one. Recall from our case-study that iteration is not sustained at the *Flatten_List* step, in other words it is executed only once consuming all galaxy coordinates. Meanwhile as per its Motif, we know that this steps build its output by coalescing all items in the input collection. As a result our labelling pipeline will first generalise all labels and compute a label for the top-level collection element consumed by *Flatten_List*. This label will get propagated to *Flatten_List*'s output, which is a list. On the other hand, this list is not consumed as a whole by downstream activities, instead each item in it is used. Therefore each item inherits the labels from the enclosing list (through a distribute operator). As a result, we end up with inaccurately labelled items.

**Q3-L** By using labels we are now able to fully represent Q3. The labelling function for the *calculate_internal_extinction* step creates labels of type $case : hasMorphology$ to capture the context represented by the morphology input parameter. When we look at the precision of this query result it also deteriorates with increasing inputs. This is because the coordinate inputs to the extinction calculation are inaccurately labelled due to upstream *Flatten_List* step.

We will now review related work on provenance annotation, after that we will provide a critical discussion of the above presented results of label-based PDDS.

## 7.7 Related Work

*Label*Flow brings together two capabilities:

- the provisioning of domain-specific annotations by promoting parts of data values to become metadata.
- the propagation of annotations to other (close derivative) data nodes in the provenance trace.

We therefore review related work in these two categories.

### 7.7.1 Obtaining Annotations

In pioneer works on provenance annotation [MSZ$^+$10] [ZWG$^+$04], the primary focus has been on capturing (through manual annotations) the static metadata, characterising elements of a workflow description and propagating those characteristics to execution provenance.

Cao et al [CPS$^+$09] has been the first to focus on dynamic metadata. This work brings annotation capabilities to a desktop application that allows users to perform analyses by interacting with remote services available on a Grid. The authors propose the use of specialist *Annotator*s that crawl over data nodes in a provenance graph that are known to be of a specific domain type (e.g. a BLAST [AGM$^+$90] report). Annotators can parse data values in known formats and can create annotations using data values. Here the objective is to create metadata exhaustively by exploiting all possible metadata headers/fields in a data file. On the other hand the authors do not discuss how this rich metadata will be utilised by the application.

In [SSH08] Sahoo et al describe the SPADE system, where they highlight dynamic metadata, and they too exploit data artefacts as the source of metadata. The authors propose using "semantic provenance modules", similar to Cao's domain-specific annotators to create elaborate metadata. They propose such modules be inserted in-between analytical steps in workflows. Similar to Cao's work, SPADE focuses on providing one particular domain-specific ontology and elaborate metadata conforming to that ontology. Note that this approach requires altering the original scientific workflow to denote points of interruption, where the annotator will execute.

In a recent paper De Oliviera et al [dOSM15] question "how much domain data should be in provenance?". Their answer is that it should be under the control of the user. De Oliviera cites and takes influence from our work *Label*Flow in building a dynamic metadata provisioning capability tailored to support parallel scientific workflow systems. Similar to our labelling functions, they associate user-designed *Extractor* classes with outputs of selected workflow activities. Similar to SPADE, De Oliviera's approach requires alterations to the original workflow to denote the extractor class for activities. The need to embed the metadata generation capability into the workflow is justified by a need to have and query this metadata while the workflow is still running. In the parallel workflow setting workflow activities are long running, and one way to detect anomalies in the execution is to retrieve intermediary results based on domain specific characteristics and to inspect them.

The distinctive aspects of *Label*Flow against these approaches are:

- its non-intrusiveness to the execution of workflow. As the metadata is sourced from the data, as long as the data values are kept, annotation can take place as an offline process any later time.

- its focus on capturing the context that surrounds a particular analytical activity. All prior approaches focus on extracting metadata from the value of a data node and associating metadata with that data node. We focus on the cases where the context is spread out among input parameters and result datasets. As we use labelling functions that consume all data (input/output) associated with an activity, we provide a mechanism to weave back this spread out context.

### 7.7.2 Propagating Annotations

Attribute propagation has been first studied in the context of `part-whole` relations in Object-Centered systems and in Description Logics [AFGP96], where attributes of parts can be considered attributes of wholes and vice versa. In [HGP93] authors describe an attribute propagation mechanism in Object-Oriented databases that exploits the part-whole relations. Two types of propagation is outlined *invariant* and $transformational$, where the latter is typically used to aggregate attributes of parts to obtain an attribute for the whole (e.g. a car's weight is the sum the weight of all its components). Note that it is the same part-whole relation implied by a subset of our Motifs (discussed earlier in Chapter 3) that we exploit to perform annotation propagation.

Metadata propagation has been explored in digital library research. In [Gre09] authors accelerate the curation of shared research work products through propagation of basic metadata, such as authorship, subject, or publication date, from the research articles to their supplementary material (data, visualisations, charts). Such propagation may result in incorrect annotations (e.g. not all charts of a paper may have been authored by the same person). Authors propose that inaccuracies are later corrected via manual curation.

Propagating annotations of data artefacts to other data artefacts by exploiting provenance has been studied in the context of databases. Recall from Chapter 2 that database provenance was concerned with tracking $Why$, $How$ and $Where$. $Why$ explains which source tuple(s) cause a particular record to appear in a query result; $How$ explains by which relational operators the source tuples are combined; and *Where* explains from which source cells are the data values copied to the result. DBNotes [BCTV04] exploits *Where* provenance to propagate annotations on source cells to results of Select-Project-Join-Union queries. DBNotes uses set union to gather all annotations of source cell values to obtain an annotation set for the result. *Why* and *How* provenance have recently been applied to dataflows comprised only of data-querying modules, implemented with relational queries [ICF$^+$12] or PigLatin programs [ADD$^+$11]. In [BL06] authors describe a logic-based approach, which exploits *How* provenance to propagate of schema-level semantic annotations through relational query based activities. Rules for propagation of annotations through each relational operator is represented as a logic constraint. Authors speculate that such an approach can find applicability in semi-automated annotation of workflows.

What sets *Label*Flow apart from annotation propagation over white-box database provenance is that it operates over grey box provenance. In a scientific workflow setting, data structures and computations are diverse, hence we cannot make the restrictive assumptions on the structure of data (as relations and tuples), and the kinds of data-processing (relational queries). On the other hand our empirical analysis on workflows and the Motif categorisation has shown that certain types of computation (data adapters) are not entirely arbitrary, and their operation can be made explicit in a rough-cut manner through semantics annotations specifying, from which inputs to which outputs value-copying occurs. We will discuss the implications of the grey-box in the next section.

## 7.8 Discussion

The revisit of the case-study revealed that the combination of liberal label propagation with broken factorial design can lead to inaccurate labels, which shows that this (anti)pattern not only degrades the standalone use of provenance traces but also degrades the operation of provenance enhancement applications, as our annotation approach.

Our propagation of labels is liberal as it is a combination of the following behaviours:

- We act on partial information (grey-box transparency denoting some value-copying occurring between inputs and outputs of an activity). When an activity invocation consumes a collection of items with distinct labels, and produces another collection of items, grey-box transparency does not allow us to accurate propagate labels item-wise. So instead we first *GENERALISE* labels to the top level input collection and *PROPAGATE* them to the top level output collection.

- To further expand the reach of labels we *DISTRIBUTE* labels at the top level collection to each item.

Other provenance annotation approaches do not have the inaccuracy issue either because:

- they do not support propagation of labels as in SPADE [SSH08] or De Oliviera's [dOSM15] approach,
- they do not support a fine-grained provenance capability where annotations from distinct fine grained sources need to be managed as in the Galaxy workflow system metadata propagation features [GRH+05].
- or they require the user to not only supply the initial annotations but also the rules of propagation per workflow activity as in the Wings workflow system [GRK+11].

Rather than having these restrictions, a promising solution could be integrating workflow analysis with labelling. The workflow analysis rules can deduce whether lineage traces from multiple sources will be joined up at an activity invocation or not. We anticipate that by superimposing the label generation and propagation capabilities of activities as additional rules, the tool can also inform us whether labels from disparate sources will be joined up or not. This research falls in scope of our future work.

The cost involved in adapting our system is the manual annotation of workflow activities with labelling specifications and developing labelling functions for the focal data generation points in workflows. These are one-time costs. Both labelling specs and labelling functions can be reusable as the activities are underpinned by common components from local libraries or service catalogues [BTN+10] [MKRI15]. Consequently a labelling spec generated for a component is re-usable for all its occurrences in workflows.

Earlier we mentioned that we designed *MINT* and *PROPAGATE* operators with reusability in mind. Given that our operators decorate standard PROV traces, they have the potential to be used for labelling traces of workflow systems other than Taverna.

Assessing the re-usability of our operators remains part of our future work discussed in Chapter 6.

As we saw in related work, any attempt at automating the generation of dynamic metadata has to assume the existence of a metadata extraction capability, i.e. labelling functions. The cost of developing such functions can be minimised by exploiting existing libraries in parsing and transforming standardised scientific data formats.

With labels we have adopted a very simplistic model to represent metadata. This can be viewed as a middle-ground between having no explicit metadata and having fully-fledged ontology based representations that conceptually describe provenance artefacts [HKP$^+$09], and interlink them with information shared elsewhere [BBR$^+$13]. As evidenced by existing experiment bundling approaches [BZG$^+$15] rich annotations created manually by users are more suited to a coarse grained annotation approach, where the entire workflow or its entire inputs, outputs get annotated as part of a bundle. In our case we are attempting at annotation at a very fine grain. Our labels can be seen as taints over data created due to data's association with an activity or another data in a provenance trace. We have therefore opted for a simple representation.

## 7.9 Chapter Conclusion

We described an architecture where 1) we use workflows and provenance traces associated with annotation behaviour as a roadmap to collect and propagate domain specific metadata and 2) we use data values as the source of domain specific metadata in the form of labels. We described two generic operators obtaining labels and propagating it to data's close derivatives created through data adapter steps. We also described two Taverna specific operators that cater for the Collection-Item structure of data, and allowed propagation of labels from Collections to Items and vice versa.

We assessed the utility of the proposed architecture and labels using the PDDS case-study. We observed that labels allow to fully implement PDDS queries, which in the absence of metadata were only partially implemented. On the other hand we observed that broken factorial-design impacts the accuracy of labelling, as the propagation of labels among data derivatives is predicated over partial (grey-box) lineage information coming from the Motif's of workflow activities. We will discuss future research in the final chapter of the dissertation.

# Chapter 8

# Conclusions and Future Work

## 8.1 Brief Summary

**Scientific Workflows help systematise data-oriented analyses by several means.** They alleviate resource heterogeneity by providing established resource access mechanisms, they automate computational parts of analyses, thereby enabling systematic exploration of experimental variation and repetition. Moreover, workflow systems are equipped with execution tracing capabilities that capture provenance information documenting each step of analysis, and the data consumed and generated in each step. Provenance features of workflow systems bring step-wise transparency into investigations; enabling their monitoring, debugging and end-to-end audit-ability. All these capabilities result in streamlined, credible and repeatable data analysis processes for scientists [DF08]. **In parallel to the systematisation of how analyses are done, there are emerging efforts that aim to systematise how analyses are reported when analysis outputs are to be shared.** Systematisation in reporting signals a move from traditional narrated descriptions of experiments, as in Materials and Methods sections of articles to more structured forms of experimental metadata, describing the experimental set-up including configurations, varied parameters and methods. To assist scientists in the creation of such metadata there are emerging efforts in providing vocabularies, reporting frameworks and guidelines [SRSF$^+$12] [TFS$^+$08]. A natural expectation would be that investigations that are systematic in their encoding and enactment of the analysis, as with workflows, could equally be advantageous in reporting. In this dissertation we observe that this premise holds weakly. **While workflows and their provenance finds use during workflow design and enactment activities, this information is rarely used to support experiment reporting.** The work described in this thesis is based on a formulation of this disconnect as the *provenance gap*, where we identify challenges of workflow provenance that hinders its use in reporting.

More specifically, experiment reporting poses the following requirements for workflow provenance, which highlight challenges we tackle in this dissertation research:

- **Represent Factorial Design:** corresponds to the ability to represent experimental parameters and encode experimental variation into workflows, and to record provenance in a granularity that can capture distinct parameters and enactment of the analysis with those parameters. The challenge here is in understanding the level of support that workflow systems and their provenance infrastructures support such granularity of provenance recording.
- **Support Data Selection:** corresponds to the ability to pose queries over provenance traces to seek parameter nodes of interest and to traverse traces to access results that are obtained using those parameters. The first challenge here is understanding whether workflow provenance traces lend themselves to such a seek and access pattern and understanding the factors that hamper the realisation of such queries. The challenges that lie in provenance querying are:
  - the representation of data and analyses as opaque nodes in provenance lacking in attributes/characteristics that could facilitate their filtering.
  - the existence of coarse-grained analytical steps in provenance, which prohibit discrete traceability between distinct experimental parameters and results.
- **Embody Domain-Specific Information:** corresponds to having descriptions that characterise the parameters, the analyses and the results in a discipline-specific manner. These descriptions are required to make shared results understandable by others, they are also required as attributes with which provenance traces can be queried (for data selection). The challenge here is the genericity of workflow provenance, which stems from the generic nature of standard provenance models and the black-box nature of provenance collection in workflow systems. While manual annotation of workflows by scientists is a common way of obtaining domain-specific descriptions over workflows, manual annotation is hard to scale execution traces, requiring automation.
- **Be Abstracted From Details:** corresponds to the ability to distinguish between elements of the analysis that are report-worthy and those that can be considered a

detail. The challenge here arises from the dual role of workflows as automators of resource integration on one hand and as documenters of scientific analyses on the other. Integrator role causes workflows to be complex, a side-effect of the heterogeneity of the shared scientific resources. Workflows contain several steps dedicated to data adaptation among the analytical steps underpinned by heterogeneous resources. These steps are an experimental detail required for implementation undesired for reporting. Scientists tackle complexity by eliciting abstractions in workflow designs, which is a manual process.

In this dissertation we developed mechanisms that tackle these challenges. In the next section we summarise our research hypotheses, contributions, how we have met each research objective. A traceability table among these is given in Table 8.1 Our summary in Section 8.2 follows from this table.

## 8.2 Hypotheses and Contributions

**Formulation of Provenance Gap (C1.1) and Survey of Workflow Systems (C1.2):** Our focus here has been exploring our overarching hypothesis (H1) to check whether **workflow provenance is a potential information source that can support requirements of experiment reporting**. We approached this by making observations on experiment reports and projecting those onto workflow provenance, as reported in Chapter 2. This projection revealed commonalities, which is in support of our hypothesis but also revealed gaps. The gaps were attributable to common assumptions and common provenance modelling and management techniques adopted by workflow systems. The black-box assumption, the granularity and the consistency of modelling processes and data and parameters at the workflow description and workflow provenance layers, the data storage schemes of workflow systems are among the characteristics that underlie the gap. We enumerated these characteristics to comparatively survey workflow systems to understand the level of support they provide against reporting requirements. Contributions (C1.1) and (C1.2) meet our objectives in understanding requirements of Experiment Reporting (O1) and identifying a set of research challenges in threads of Analysis, Abstraction and Annotation of workflow provenance (O2). Work presented in Chapters 3 and 5 have supported us in meeting these objectives. More specifically high level challenges presented in Chapter 2 have been elicited into more concrete problems with:

- empirical workflow analysis presented in Chapter 3. This work has revealed reasons of workflow complexity and provided hints for a solution.
- the case study on querying of workflow provenance presented in Chapter 5. This work has identified the characteristics of workflow provenance (genericity) and the patterns it contains (broken factorial design), which are bottlenecks for querying.

**Motif Taxonomy (C2) and Quantified Motif Occurrences (C3):** A common factor contributing to the challenges of workflow provenance against reporting is the black-box assumption that allows modelling arbitrary computations abstracted as analysis activities in workflows and allows their provenance to be recorded by external observation of the activity's inputs and outputs. While this assumption allows basic provenance to be provided for diverse computations, it limits actionable information over provenance, thereby is a bottleneck for any approach that aims to exploit provenance. In our second hypothesis (H2) we pointed that **computations in scientific workflows are not entirely arbitrary and existing practices in workflow development exhibits common patterns that would allows us to categorise analytical activities**. In support of this we performed an empirical analysis of 260 workflows from 4 workflow systems and 10 domains, and as presented in Chapter 3:

- we have shown that a **high-level categorisation** of activity functionalities, data characteristics and design practices in workflows is possible. Moreover we have shown that data adaptation steps show common patterns of functionality allowing a more detailed characterisation to be made. We have captured these categorisations in the Workflow Motif taxonomy.
- we have quantified occurrences of motifs in the analysis cohort. Our survey has revealed that data adapter steps on average account for 63% of activities in workflows. This percentage reaches 70% in workflow systems like Taverna where workflows are built by combining resource in a heterogeneous open environment. These adapter steps have been identified as the prime contributor to workflow complexity. Our survey also revealed the current practices scientists adopt in abstracting workflows, which were creating composite (layered) workflows and bookmarking significant intermediary data points in the workflow.
- by comparing data adapter Motifs to Hull's early work on Shims [HSL$^{+}$04] we

have demonstrated that Motifs allow us to make a grey-box assumption on workflow activities by not only characterising activity function but also implying lineage transparency among inputs and outputs of certain adapters. We discussed that this relation is primarily a part-whole relation.

With contributions (C2) and (C3) we have met objective (O3) in being able to break the black-box assumption in scientific workflows with observations made over real-world practices and patterns.

| Hypotheses | Contributions | Presented In | Meets Objectives |
|---|---|---|---|
| H1 (base) | C1.1, C1.2 | Chapter 2,<br>Chapter 3 (supportive),<br>Chapter 5 (supportive) | O1, O2 |
| H2 (analysis) | C2, C3 | Chapter 3 | O3, O7 |
| H3 (abstraction) | C4, C5 | Chapter 4 | O4, O5, O6, O7 |
| H4 (analysis) | C6 | Chapter 6 | O4, O6, O7 |
| H5 (annotation) | C7 | Chapter 7 | O4, O5, O6, O7 |

Table 8.1: Hypotheses, Contributions and Research Objectives.

In Chapters 4, 6 and 7 of this dissertation we have presented techniques for Abstraction, Analysis and Annotation (AAA) of workflow provenance. In doing so we have met our objective (O4) for finding solutions in AAA. In implementing solutions we have paid attention to meeting our cross-cutting objectives in providing technology-independent solutions (O5), assess solutions with real-world workflows where possible (O6) and compare our work against the state of the art (O7). We will now discuss our contributions in AAA and how we have met cross-cutting objectives.

**Abstraction System Survey (C4) and Workflow Abstraction Framework (C5):** With the availability of actionable information on workflow activities in the form of Motifs we have first tackled the complexity challenge of workflow provenance (reported in Chapter 4). As computational provenance abstraction is a recent and active area of research we have performed a review of existing approaches. To do this we have reverse-engineered a blueprint from existing systems identifying the main components of computationally-assisted provenance abstraction. Using this blueprint, along research objective O7, we have provided a comparative survey of the state of the art in provenance abstraction (C4).

We have shown that algebraic graph re-writing is a suitable formalism to represent transformations, which we called primitives, for abstracting workflow description graphs. Using this formalism we have represented primitives that either eliminate activities or create activity groups. This formal presentation was also used to discuss integrity guarantees of each primitive. Our abstraction techniques operated over a workflow-system independent graph-based representation of workflows in line with objective O5.

In support of hypothesis (H3) **we have developed a workflow abstraction framework (C5) that is driven by user specified policies that identify insignificant activities based on their motifs and identify how such activities shall be abstracted away with primitives.**

We have evaluated the abstraction framework with real world Taverna workflows, in line with objective O6. The evaluation has allowed us to compare and understand the different abstractive effects of primitives. We observed that while elimination can fully abstract away activities denoted as insignificant it would not reduce length of an analysis's main data derivation path with the same ratio. Meanwhile grouping activities has a levelled reductive affect on the number of activities and the length of data derivation path.

We have assessed how much abstractions generated with a default set of policies agree with abstractions generated by users. An important finding here was that a report-worthy activity may not always be co-located with the report-worthy copy of that activity's input/output data. This highlighted that grouping activities as opposed to eliminating them is a more versatile/suitable approach for abstraction. We demonstrated that simple grouping rules-of-thumb were encodable for Life Science workflows, where abstractions generated by the framework were in high agreement with user's abstractions.

In order to understand the level of support that workflow provenance gives to data selection we have performed a case study that involved selected provenance queries from literature and workflow execution provenance from Taverna system (reported in Chapter 5). This study highlighted the characteristics of provenance in general and Taverna provenance in particular that hamper its use for data selection. We observed that:

- Data selection queries require domain specific attributes of data and activities, which in their absence (due to the genericity of provenance) cause queries to be implemented partially or in an adhoc manner by referring to data values or (substitute) generic attributes.

- Taverna's approach in giving workflow designers the full control of encoding factorial design into workflows can have negative side effects on usability of provenance for data selection. Scientists may cause breaks in factorial design when they configure the workflow in a way that would cause an analysis activity consumes multiple data artefacts descending from distinct input parameters. Such breaks mean that accuracy of query results becomes the reciprocal of the input size, rendering provenance minimally useful for data selection.

- Data selection queries seek data by its origin, which requires transparency over lineage relations found in a workflow provenance trace.

These findings served as the seeds of our follow-on hypotheses and research on provenance analysis and annotation.

**Taverna Workflow Analysis Rules (C6)** Our observations on Taverna's constructs to support factorial design, its well-defined execution behaviour, its modelling of data and processes at consistent granularities both at the workflow description and execution provenance layers have led to hypothesis (H4), where we stated that **we can analyse Taverna workflows to anticipate structure of the provenance traces that the workflow execution will produce. This analysis can check for the existence of prospective coarse grained activity invocations, which break factorial design.**

To do this, as reported in Chapter 6, we have exploited an insight that has been given by Missier et al [MPB10] that shows how iteration leads to a correspondence among input and output collections of activities, which in turn is a guarantee of discrete lineage traces among items in corresponding collections. We have called this correspondence a depth mapping. We have adopted the functional formalism used by Missier to represent Taverna's operational behaviour as a set of known computations. We have shown that depth mapping rules can be provided for each such computation. We have used a declarative logic programming language, specifically Datalog, to represent depth mapping rules, which, when combined with ground facts representing a particular workflow description, inform us of the workflow's provenance characteristics. In line with objective O6, we have analysed astronomy workflow used earlier in Chapter 5. This example analysis trace (given in full in Appendix D showed how our rules correctly identify the point in workflow design (activity and input port) that is observed earlier in Chapter 5 as the break in factorial design. In line with objective O7 we have reviewed the state of the art in workflow provenance analysis. We have observed that this field of work is only emerging and existing approaches [DKBL15] [MPB10] primarily identify patterns potentially observable in provenance traces and show how such patterns can be exploited in optimising provenance query evaluation. Against this setting we have observed that our techniques focus on detecting the broken factorial design pattern that is actually observed in real-world traces and that can seriously hamper provenance querying accuracy.

**Provenance Labelling Operators (C7)** Our motivation for bringing domain-specific metadata onto provenance stems from the findings of our case study on provenance driven data selection. Specifically the case study showed that lack of domain specific information, and the opacity of lineage relations in workflow provenance resulted in partial or adhoc implementation of PDDS queries. Therefore our approach to annotation has been shaped to tackle these shortcomings. More specifically, as reported in Chapter 7, we focused on annotations that capture the context as defined by the input parameters and activity configurations of a workflow in which scientific data sets are collected, analysed and visualised. Furthermore we have focused on fine-grained and dynamic metadata, as parameters, and therefore context can change through iterated activity executions in workflows.

Our findings in the case study pointed that context is implicitly captured in data values, or in values of generic attributes of workflows and provenance. In response to this we have sought to devise a mechanism that can make explicit this implicitly captured context in the form of domain-specific labels. In support of hypothesis (H5) **we have developed the *Label*Flow framework which embodies four labelling operators that encapsulate the generic portions of domain-specific annotation provisioning and propagation capability.** We have shown how Motifs due to the transparency they bring to activity functions and (in the case of certain Data Adapters) to lineage can be used to determine where in a provenance trace we have data with implicit domain-specific descriptions and to which other data artefacts those descriptions can be propagated. In line with objective O5, our labelling operators operate over standard PROV compliant provenance traces and adopt a domain-independent generic model Label model to decorate provenance.

In line with objective O6, to assess the framework we have used it to annotate the provenance traces of our case study (astronomy) workflow. We have shown how labels allow us to implement case-study queries systematically replacing prior adhoc means. On the other hand our assessment has shown that liberally propagating labels from inputs to outputs of data adapters and also among items and collections may lead to inaccurate labelling. For our case study where a point of broken factorial design also corresponded to a point where labels were propagated the accuracy (for dynamic labels created per input) became the reciprocal of the size of input parameters. This has prompted us that workflow analysis and provenance annotation may be integrated to foresee inaccurate labelling. In line with objective O7 we have reviewed the state of the art in provenance annotation. Our review revealed that focus on dynamic metadata, non-intrusiveness to existing practices of workflow design and execution as aspects that distinguish our technique from others.

## 8.3 Future Research

In our work presented in this dissertation we have only scratched the surface of the work needed to bridge the provenance gap. We see several directions for future research.

**Workflow Motifs.** Our work on Motifs and empirical findings has already been recognised as:

- concrete evidence for the dominance of data adaptation steps in workflows and therefore has served as motivation for work that aims to reduce the need for such activities by providing some of those functions at the point of publishing of data [HG13] or as motivation for workflow simplification by refactoring [CBCM+14].
- a method as well as a base classification to understand workflow tasks ran on distributed computing infrastructures [OJK+13] or workflows in the specific scientific domains [PBR+14].

Classifications of tasks and data within scientific data analysis processes can be found in domain-specific ontologies. The SWO Ontology [MBL+14] in the biomedical domain is one prominent example. Investigating whether the Motif classification can be aligned with existing domain-specific classifications, or whether it is suited as an upper-level classification is a possible future direction for research.

**Workflow Abstraction.** Given that scientists are held accountable for their findings and shared work products including the data and the experiment reports, we view workflow/provenance abstraction as a process in which the user is involved and has the final say on the abstraction created. Given that scientists have the means to encode abstractions into workflow designs, the role for computationally generated abstractions could be in supporting the user throughout the design process. In this regards we see a future research direction in creating *abstraction alternatives* and provide them as abstraction suggestions for users to choose from.

Relaxing the assumptions under which an abstraction machinery operates such as supporting an unordered abstraction policy, or the ability to override abstraction policies can give way to abstraction alternatives. Our experience with the GROOVE graph grammar system [Ren03] has shown that this system can be used to create the state space, i.e. all possible intermediary and final derivations implied by a graph and a set of graph transformations. Meanwhile the size of that space would grow exponentially with the size of system modelled. Therefore as part of future research the following can be investigated:

- is it possible to define the problem of creating abstraction alternatives at a scale suitable for the application of state space exploration?
- can abstraction alternatives be scored and can there be strategies for exploring or sorting abstraction alternatives?

**Workflow Analysis.** In our research by analysing a workflow description we were able to anticipate whether its execution will lead to the pattern where fine-grained (iterated) activity invocation is followed by a coarse grained invocation, which joins up lineage traces, thereby breaks factorial design. We see potential for future research on extending workflow analysis to involve correction suggestions for the design. Suggesting corrections would require more information than what is available in the workflow description. Consider our case-study workflow, where list flattening, which is a data adapter step was causing the break in factorial design. By augmenting the workflow description with semantic descriptions denoting the types of the inputs and outputs of activities, one may infer the input and output types of adapter steps, due to composition [BEP$^+$06], and one can suggest that a step such as list flattening, which consumes and produces same typed data, yet breaks factorial design may as well be eliminated.

**Provenance Annotation.** We foresee two directions of future work on Provenance Annotation. First is integrating our labelling approach with workflow analysis. In such an approach we preserve propagation behaviour and use the analysis to raise warnings about label accuracy, or for computing confidence information for labels.

The second area of work is investigating the use of labels beyond PDDS. We observe that emerging initiatives and practices in scientific data sharing, which we discuss in the next section, is creating requirements on the granularity and content experimental metadata such that the current practice of scientist-driven metadata creation needs to change towards automation.

## 8.4 Prospect of Impact

In the introduction to our dissertation we outlined that Data-Oriented science embodies multiple value-chains, not only for the creation and dissemination of scientific knowledge, but also for the creation and dissemination of other research outputs, most notably data [BWMP07]. As a result the practice of science has become more elaborate and demands systematisation in **performing** data-oriented analyses as well as systematisation in **reporting** those analyses. In this dissertation we have observed that the efforts of systematisation on both sides are at a disconnect. To help scientists in coping with increasing demands of data-oriented science [TA$^+$11] we hope that real-world applications will be developed to bridge the disconnect. We anticipate two directions of development.

**Applications that support workflow development.** Throughout the thesis we illustrated desired characteristics for workflow provenance that facilitate its potential use in reporting. A common pattern observed throughout our solutions was that workflow designs are a key enabler in obtaining those characteristics. Therefore we argue that **the process of workflow design needs to become provenance-aware** to pre-emptively strive for those characteristics. One mechanism could be workflow tooling providing feedback/tips to designers on the anticipated characteristics of provenance. Tooling can inform users on whether traces will be discrete or not, signalling whether traces can later be used for data selection. Or tooling can report the anticipated complexity of provenance and suggest abstraction. In support of this vision we communicated our findings to the Taverna development team in a two-part technical report [Alp14]. We also provided pointers to relevant findings of state of the art provenance research.

**Applications that support data publishing.** A crucial requirement in the data value chain is to track the usage and generation of shared data and so as to ensure that 1) data providers get credit for their contributions and 2) the origin and the provenance for an investigations source and result datasets are captured to ensure the understandability of findings and the reproducibility of investigations. Data Citation is the emerging mechanism to meet these requirements [MN12]. A recent declaration by Force11 [FOR14], a community of data publishers, scientists and librarians, states that Data Citations should support a set of core capabilities, which amongst others include:

- Access to metadata describing the conditions under which data has been generated. This metadata may itself involve a citation to the source/origin datasets that has been used for the generation of shared data.
- Identification and access to relevant granules of data that supports a particular claim. So, if a claim is to be made with respect to a particular experimental setting among a series of settings, then each data output from each setting shall be shared and citable distinctly.

We observe these requirements as indicators that future Experiment Reports will describe data-oriented analyses at finer granularities and at greater detail. As a result experimental metadata creation will become more elaborate, which we believe will require that **origin and context information bleeds-through computational processes used for data analysis**. Therefore we believe the area of data publishing and the semi-automated creation of data citations is a potential application area for workflow provenance.

# Bibliography

[ABGK13] Pinar Alper, Khalid Belhajjame, Carole A Goble, and Pinar Karagoz. Enhancing and abstracting scientific workflow provenance for data publishing. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops, Managing and Querying Provenance Data at Scale (BIGProv)*, pages 313–318. ACM, 2013.

[ABJF06] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *International Provenance and Annotation Workshop, IPAW*, pages 118–132, 2006.

[ABL10] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Provenance browser: Displaying and querying scientific workflow provenance graphs. In *Proceedings of 26th International Conference on Data Engineering (ICDE)*, pages 1201–1204, 2010.

[ABML09] Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludscher. Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs. In *21st International Conference on Scientific and Statistical Database Management (SSDBM)*, volume 5566, pages 237–254. 2009.

[ABS$^+$06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 1151–1154. VLDB Endowment, 2006.

[ACHZ09] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun

Zhao. Describing Linked Datasets. In *Proceedings of WWW2009, Workshop on Linked Data on the Web LDOW*, April 2009.

[ADD$^+$11] Yael Amsterdamer, Susan B. Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen. Putting lipstick on pig: Enabling database-style workflow provenance. *PVLDB*, 5(4):346–357, 2011.

[AFG$^+$03] Matthew Addis, Justin Ferris, Mark Greenwood, Darren Marvin, Peter Li, Tom Oinn, and Anil Wipat. Experiences with eScience workflow specification and enactment in bioinformatics. In *Proceedings of UK e-ScienceAll Hands Meeting*, pages 459–467, 2003.

[AFGP96] Alessandro Artale, Enrico Franconi, Nicola Guarino, and Luca Pazzi. Part-whole relations in object-centered systems: An overview. *Data and Knowledge Engineering*, 20(3):347–383, November 1996.

[AGM$^+$90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[AHM10] Ishtiaq AHMAD. Warp2D - 2D Time Alignment Workflow. myExperiment Workflow Repository `http://www.myexperiment.org/workflows/1283.html`, May 2010. Version 3.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level.* Addison-Wesley, 1995.

[AIG12] Mohamed Abouelhoda, ShadiAlaa Issa, and Moustafa Ghanem. Tavaxy: Integrating taverna and galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13(1), 2012.

[Alp13] Pinar Alper. Pack: Scientific Workflow Motifs - FGCS Special Issue Submission Data Set. myExperiment Repository, `http://www.myexperiment.org/packs/364.html`, January 2013.

[Alp14] Pinar Alper. Re-thinking Workflow Provenance against Data-Oriented Investigation Lifecycle. The University of Manchester Library e-Scholar, `https://www.escholar.manchester.ac.uk/uk-ac-man-scw:22489`, 2014.

[Alp15a] Pinar Alper. Pack: Dissertation Chapter 4 Supplementary Material. myExperiment Repository, `http://www.myexperiment.org/packs/704.html`, July 2015.

[Alp15b] Pinar Alper. PhD Related Source Code. GitHub repository at: `https://github.com/pinarpink/phd-sources`, December 2015.

[Apa09] Apache Software Foundation. Apache Axis2/Java - Next Generation Web Services. Website `http://ws.apache.org/axis2/`, July 2009.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[AT15] Peter Amstutz and Nebojsa Tijanic. Common Workflow Language Specification. `https://github.com/common-workflow-language/common-workflow-language`, July 2015. Draft 2.

[BBB+15] George Alterand George C Banks, Denny Borsboom, Sara D Bowman, et al. Transparency and Openness Promotion (TOP) Guidelines. Open Science Framework. . Centre For Open Science, `https://osf.io/9f6gx/`, October 2015.

[BBR+13] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, et al. Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2):599 – 611, 2013. Special section: Recent advances in e-Science.

[BCBDH08] Olivier Biton, Sarah Cohen-Boulakia, Susan B. Davidson, and Carmem S. Hara. Querying and Managing Provenance through User Views in Scientific Workflows. pages 1072–1081, 2008.

[BCC+12] Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. PROV-O: The PROV Ontology. W3C, `http://www.w3.org/TR/prov-o/`, 2012.

[BCD+07] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.

[BCG$^+$12] Khalid Belhajjame, Oscar Corcho, Daniel Garijo, Jun Zhao, Paolo Missier, et al. Workflow-Centric Research Objects: First Class Citizens in Scholarly Discourse. In *Proceedings of Sepublica2012*, pages 1–12, 2012.

[BCS$^+$11] Barbara T. Blaustein, Adriane Chapman, Len Seligman, M. David Allen, and Arnon Rosenthal. Surrogate parenthood: Protected and informative graphs. *PVLDB*, 4(8):518–527, 2011.

[BCTV04] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 900–911. Morgan Kaufmann, 2004.

[BDG$^+$12] Khalid Belhajjame, Helena Deus, Daniel Garijo, Graham Klyne, Paolo Missier, Stian Soiland-Reyes, and Stephen Zednik. PROV Model Primer. Technical report, W3C, 2012.

[BDKR09] Olivier Biton, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. Optimizing user views for workflows. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 310–323, New York, NY, USA, 2009. ACM.

[bea15] BeanShell - Lightweight Scripting for Java. `http://www.beanshell.org`, 2015. Accessed December 2015.

[BEP$^+$06] Khalid Belhajjame, Suzanne M. Embury, Norman W. Paton, Robert Stevens, and Carole A. Goble. Automatic annotation of web services based on workflow definitions. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, pages 116–129, 2006.

[BF05] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys*, 37(1):1–28, 2005.

[BFG96] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Issues in the practical use of graph rewriting. In *Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 38–55. Springer Berlin Heidelberg, 1996.

[BHQ$^+$01] Alvis Brazma, Pascal Hingamp, John Quackenbush, Gavin Sherlock, Paul Spellman, et al. Minimum information about a microarray experiment (MIAME) – toward standards for microarray data. *Nature Genetics*, 29(4):365–371, December 2001.

[BKC$^+$01] Daniel Blankenberg, Gregory V. Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: A Web-Based Genome Analysis Tool for Experimentalists. *Current protocols in molecular biology*, Chapter 19, January 2001.

[BKSRS12] Marco Brandizi, Natalja Kurbatova, Ugis Sarkans, and Philippe Rocca-Serra. graph2tab, a library to convert experimental workflow graphs into tabular formats. *Bioinformatics*, 28(12):1665–1667, 2012.

[BL06] Shawn Bowers and Bertram LudŁscher. A calculus for propagating semantic annotations through scientific workflow queries. In *In Query Languages and Query Processing (QLQP): 11th Intl. Workshop on Foundations of Models and Languages for Data and Objects, LNCS*, 2006.

[BMWL07] Shawn Bowers, Timothy McPhillips, Martin Wu, and Bertram Ludscher. Project Histories: Managing Data Provenance Across Collection-Oriented Scientific Workflow Runs. In *Data Integration in the Life Sciences*, volume 4544 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin Heidelberg, 2007.

[BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[BOHS07] Sarah Bourlat, Matthias Obst, Karin Heyer, and Kerstin Steelier. Application of ecological niche modelling and earth observation for the risk assessment and monitoring of invasive species in the Baltic Sea. Swedish LifeWatch Data Portal, `http://www.svenskalifewatch.se/Global/externwebben/centrumbildningar-projekt/sv-lifewatch/dokument/MC-2_BallastwaterOption_FinalReport_1.0.pdf`, July 2007.

[BRJ05] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series).* Addison-Wesley Professional, 2005.

[BTN+10] Jiten Bhagat, Franck Tanoh, Eric Nzuobontane, Thomas Laurent, Jerzy Orlowski, Marco Roos, Katy Wolstencroft, Sergejs Aleksejevs, Robert Stevens, Steve Pettifer, Rodrigo Lopez, and Carole A. Goble. Biocatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research*, 38(Web-Server-Issue):689–694, 2010.

[BWMP07] Christine L. Borgman, Jillian C. Wallis, Matthew S. Mayernik, and Alberto Pepe. Drowning in data: digital library architecture to support scientific use of embedded sensor networks. In *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2007, Vancouver, BC, Canada, June 18-23, 2007, Proceedings*, pages 269–277, 2007.

[BZG+13] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Aleix Garrido, Stian Soiland-Reyes, Pinar Alper, and Oscar Corcho. A Workflow PROV-corpus Based on Taverna and Wings. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops, ProvBench: Provenance Benchmark Challenge*, EDBT '13, pages 331–332, New York, NY, USA, 2013. ACM.

[BZG+15] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, Oscar Corcho, Jose Manuel Gomez-Perez, Sean Bechhofer, Graham Klyne, and Carole Goble. Using a suite of ontologies for preserving workflow-centric research objects. *Web Semantics: Science, Services and Agents on the World Wide Web*, 32(0):16 – 42, 2015.

[CAB+10] A. Chapman, M. D. Allen, B. Blaustein, L. Seligman, C. Wolf, M. Wolfe, and A. Rosenthal. PLUS: Provenance for Life, the Universe and Stuff. The MITRE Corporation Technical Papers, `http://www.mitre.org/sites/default/files/pdf/10_1364.pdf`, June 2010.

[CAB+14] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Seltzer, and Andy Hopper. A primer on provenance. *Queue*, 12(3):10:10–10:23, March 2014.

[Car05] Jorge Cardoso. Evaluating the process control-flow complexity measure. In *IEEE International Conference on Web Services*, pages –804, July 2005.

[CBCG+14] Sarah Cohen-Boulakia, Jiuqiang Chen, Carole Goble, Paolo Missier, Alan Williams, and Christine Froidevaux. Distilling structure in taverna scientific workflows: A refactoring approach. *BMC Bioinformatics*, 15(1):S12, 2014.

[CBCM+13] Sarah Cohen-Boulakia, Jiuqiang Chen, Paolo Missier, Carole Goble, Alan R Williams, and Christine Froidevaux. Distilling structure in taverna scientific workflows: A refactoring approach. *BMC Bioinformatics*, 2013. in-print.

[CBCM+14] Sarah Cohen-Boulakia, Jiuqiang Chen, Paolo Missier, Carole Goble, Alan Williams, and Christine Froidevaux. Distilling structure in taverna scientific workflows: a refactoring approach. *BMC Bioinformatics*, 15(Suppl 1):S12, 2014.

[CCL+10] Artem Chebotko, Seunghan Chang, Shiyong Lu, Farshad Fotouhi, and Ping Yang. Secure abstraction views for scientific workflow provenance querying. *Services Computing, IEEE Transactions on*, 3(4):322–337, Oct 2010.

[CCT09] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[CDS+07] Sayeed Choudhury, Tim DiLauro, Alex Szalay, Ethan Vishniac, Robert Hanisch, et al. Digital data preservation for scholarly publications in astronomy. *International Journal of Digital Data Curation, vol.2 no.2, p.20-30*, 2:20–30, November 2007.

[CF12] Fernando Chirigati and Juliana Freire. Towards Integrating Workflow and Database Provenance. In *Provenance and Annotation of Data and Processes*, volume 7525 of *Lecture Notes in Computer Science*, pages 11–23. Springer Berlin Heidelberg, 2012.

[CFS$^{+}$06] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cludio T. Silva, and Huy T. Vo. Vistrails: Visualization meets data management. In *In ACM SIGMOD*, pages 745–747. ACM Press, 2006.

[CFV$^{+}$08] Ben Clifford, Ian Foster, Jens-S. Voeckler, Michael Wilde, and Yong Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 20(5):565–575, April 2008.

[CG08] Vasa Curcin and Moustafa Ghanem. Scientific workflow systems - can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pages 1 –9, dec. 2008.

[CGG$^{+}$02] Vasa Curcin, Moustafa Ghanem, Yike Guo, et al. Discovery Net: Towards a Grid of Knowledge Discovery. In *Proceedings of 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 658–663. ACM, 2002.

[CGG09] Vasa Curcin, Moustafa M. Ghanem, and Yike Guo. Analysing scientific workflows with computational tree logic. *Cluster Computing*, 12(4):399–419, December 2009.

[CJ10] Adriane Chapman and H. V. Jagadish. Understanding provenance black boxes. *Distributed and Parallel Databases*, 27(2):139–167, January 2010.

[CJR08] Adriane P. Chapman, H. V. Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 993–1006, New York, NY, USA, 2008. ACM.

[CKKT11] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM symposium on Access control models* *and technologies*, SACMAT '11, pages 93–102, New York, NY, USA, 2011. ACM.

[CLFF10] Artem Chebotko, Shiyong Lu, Xubo Fei, and Farshad Fotouhi. Rdf-prov: A relational {RDF} store for querying and managing scientific workflow provenance. *Data and Knowledge Engineering*, 69(8):836 – 865, 2010.

[CLMG+10] José A. Cruz-Lemus, Ann Maes, Marcela Genero, Geert Poels, and Mario Piattini. The impact of structural complexity on the understandability of uml statechart diagrams. *Inf. Sci.*, 180(11):2209–2220, June 2010.

[CMD+14] V. Curcin, S. Miles, R. Danger, Y. Chen, R. Bache, and A. Taweel. Implementing Interoperable Provenance in Biomedical Research. *Future Gener. Comput. Syst.*, 34:1–16, May 2014.

[COGC+11] Paolo Ciccarese, Marco Ocana, Leyla Jael Garcia Castro, Sudeshna Das, and Tim Clark. An open annotation ontology for science on web 3.0. *Journal of Biomedical Semantics*, 2(2), 2011.

[Con16] Clinical Data Interchange Standards Consortium. The Analysis Data Model, Version 2.1. CDISC Portal, `http://www.cdisc.org/adam`, 2016. Accessed March 2016.

[CP14] James Cheney and Roly Perera. An analytical survey of provenance sanitization. In *5th International Provenance and Annotation Workshop, IPAW*, pages 113–126, Koln, Germany, 2014.

[CPS+09] Bin Cao, Beth Plale, Girish Subramanian, Paolo Missier, Carole A. Goble, and Yogesh Simmhan. Semantically Annotated Provenance in the Life Science Grid. In *1st International Workshop on the Role of Semantic Web in Provenance Management*. CEUR Proceedings, 2009.

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

[CSL$^+$08] Artem Chebotko, Chang Seunghan, Shiyong Lu, Farshad Fotouhi, and Ping Yang. Scientific workflow provenance querying with security views. pages 349–356. IEEE, 2008.

[CW01] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. In *The VLDB Journal*, pages 471–480. Morgan Kaufmann, 2001.

[Dat11] DataCite. Datacite metadata schema for the publication and citation of research data. Technical report, Mar 2011. 10.5438/0003.

[dat12] Datalog as a lingua franca for provenance querying and reasoning. In *Presented as part of the 4th USENIX Workshop on the Theory and Practice of Provenance*, Berkeley, CA, 2012. USENIX.

[dat15] Data observation network for earth (dataone) project website. `https://www.dataone.org`, 2015. Accessed: 10.Dec.2015.

[Dav11] Clive Davenhall. Curation Reference Manual, Chapter on Scientific Metadata. The Digital Curation Centre (DCC), `http://www.dcc.ac.uk/sites/default/files/documents/ScientificMetadata_2011_Final.pdf`, October 2011.

[DBK$^+$14] Saumen C. Dey, Khalid Belhajjame, David Koop, Tianhong Song, Paolo Missier, and Bertram Ludäscher. UP & DOWN: improving provenance precision by combining workflow- and trace-level information. In *6th Workshop on the Theory and Practice of Provenance, TaPP'14*, Cologne, June 12-13 2014.

[DBK$^+$15] Saumen Dey, Khalid Belhajjame, David Koop, Meghan Raul, and Bertram Ludascher. Linking prospective and retrospective provenance in scripts. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, Edinburgh, Scotland, July 2015. USENIX Association.

[dCCM09] Sérgio Manuel Serra da Cruz, Maria Luiza Machado Campos, and Marta Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. In *SERVICES I*, pages 259–266, 2009.

[DCMB15] Roxana Danger, Vasa Curcin, Paolo Missier, and Jeremy Bryans. Access control and view generation for provenance graphs. *Future Generation Computer Systems*, 49:8 – 27, 2015.

[DCVK$^+$13] Saumen Dey, Víctor Cuevas-Vicenttín, Sven Köhler, Eric Gribkoff, Michael Wang, and Bertram Ludäscher. On implementing provenance-aware regular path queries with relational query engines. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 214–223, New York, NY, USA, 2013. ACM.

[DF08] Susan B Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD Conference*, pages 1345–1350, 2008.

[DKBL15] Saumen Dey, Sven Koehler, Shawn Bowers, and Bertram Ludaescher. Computing location-based lineage from workflow specifications to optimize provenance queries. In *Provenance and Annotation of Data and Processes*, volume 8628 of *Lecture Notes in Computer Science*, pages 180–193. Springer International Publishing, 2015.

[DLP$^+$10] Ivo Dinov, Kamen Lozev, Petros Petrosyan, Zhizhong Liu, Paul Eggert, Jonathan Pierce, Alen Zamanyan, Shruthi Chakrapani, John Van Horn, D. Stott Parker, Rico Magsipoc, Kelvin Leung, Boris Gutman, Roger Woods, and Arthur Toga. Neuroimaging study designs, computational analyses and data provenance using the loni pipeline. *PLoS ONE*, 5(9):e13070, 09 2010.

[dMDE$^+$10] Paula de Matos, A. Dekker, M. Ennis, Janna Hastings, K. Haug, S. Turner, and Christoph Steinbeck. ChEBI: a chemistry ontology and database. *Journal of Cheminformatics*, 2(Suppl 1):P6+, 2010.

[DMST14] Andrew P. Davison, Michele Mattioni, Dmitry Samarkanov, and Bartosz Telenczuk. Sumatra: A Toolkit for Reproducible Research. In *Implementing Reproducible Research*, chapter 3, pages 57–78. Chapman & Hall, 2014.

[dOCVSaM14] Daniel de Oliveira, Flavio Costa, Kary Ocaa Vtor Silva and, and Marta Mattoso. Debugging scientific workflows with provenance:

Achievements and lessons learned. In *29th Brazilian Symposium on Databases*, 2014.

[dOSM15] Daniel de Oliveira, Vítor Silva, and Marta Mattoso. How much domain data should be in provenance databases? In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, Edinburgh, Scotland, July 2015. USENIX Association.

[dPHG$^+$13] Renato de Paula, Maristela Holanda, Luciana S. A. Gomes, Sérgio Lifschitz, and Maria Emilia M. T. Walter. Provenance in bioinformatics workflows. *BMC Bioinformatics*, 14(S-11):S6, 2013.

[DRGS08] David De Roure, Carole Goble, and Robert Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25:561–567, May 2008.

[DShS$^+$05] Ewa Deelman, Gurmeet Singh, Mei hui Su, James Blythe, Yolanda Gil, Carl Kesselman, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3), 2005.

[dSMGdS$^+$11] Mauro Enrique de Souza Muoz, Renato De Giovanni, Marinez Ferreira de Siqueira, Tim Sutton, Peter Brewer, et al. openModeller: a generic approach to species' potential distribution modelling. *GeoInformatica*, 15(1):111–135, 2011.

[dSMRD08] Paulo Pinheiro da Silva, Deborah L. McGuinness, Nicholas Del Rio, and Li Ding. Inference Web in Action: Lightweight Use of the Proof Markup Language. In *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 847–860. Springer, 2008.

[DZL11] Saumen C. Dey, Daniel Zinn, and Bertram Ludäscher. Propub: towards a declarative approach for publishing customized, policy-aware provenance. In *Proceedings of the 23rd international conference on Scientific and statistical database management*, SSDBM'11, pages 225–243, Berlin, Heidelberg, 2011. Springer-Verlag.

[DZL12] Saumen Dey, Daniel Zinn, and Bertram Ludaescher. Reconciling provenance policy conflicts by inventing anonymous nodes. In *The Semantic Web: ESWC 2011 Workshops*, volume 7117 of *Lecture Notes in Computer Science*, pages 172–185. Springer Berlin Heidelberg, 2012.

[ESLF09] Tommy Ellkvist, Lena Strömbäck, Lauro Didier Lins, and Juliana Freire. A first study on strategies for generating workflow snippets. In *Proceedings of the First International Workshop on Keyword Search on Structured Data*, KEYS '09, pages 15–20, New York, NY, USA, 2009. ACM.

[Exp12] Susana Sanchez Exposito. Workflow: Calculating the internal extinction with data from leda. myExperiment Repository, `http://www.myexperiment.org/workflows/2920/versions/2.html`, 2012.

[FAJS05] Eric H Fegraus, Sandy Andelman, Matthew B Jones, and Mark Schildhauer. Maximizing the Value of Ecological Data with Structured Metadata: An Introduction to Ecological Metadata Language (EML) and Principles for Metadata Creation. *Bulletin of the Ecological Society of America*, 86(3):158–168, July 2005.

[FKSS08] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering.*, 10(3):11–21, May 2008.

[FOR14] FORCE11 Data Citation Synthesis Group. Joint Declaration of Data Citation Principles - FINAL. `http://www.force11.org/node/4769`, 2014.

[fSDSC02] Consultative Committee for Space Data Systems (CCSDS). Reference Model for an Open Archival Information System (OAIS). Blue book. `http://public.ccsds.org/publications/archive/650x0b1.pdf`, January 2002.

[GAB$^+$12] Daniel Garijo, Pinar Alper, Khalid Belhajjame, et al. Common motifs in scientific workflows: An empirical analysis. In *In the proceedings of the IEEE eScience Conference*. IEEE CS, 2012.

[GAB13] Daniel Garijo, Pinar Alper, and Khalid Belhajjame. The workflow motif ontology. UPM Ontology Engineering Group, `http://vocab.linkeddata.es/motifs/`, June 2013. Revision 1.02.

[GAB$^+$14] Daniel Garijo, Pinar Alper, Khalid Belhajjame, Óscar Corcho, Yolanda Gil, and Carole A. Goble. Common motifs in scientific workflows: An empirical analysis. *Future Generation Computer Systems*, 36:338–351, 2014.

[gal13] Galaxy - published workflows. `https://usegalaxy.org/workflow/list_published`, 2013.

[GB84] Ira P. Goldstein and Daniel G. Bobrow. A Layered Approach to Software Design. In D. R. Barstow, H. E. Shrobe, and E. Sandewall, editors, *Interactive Programming Environments*, pages 387–413. McGraw-Hill, New York, 1984.

[GBG$^+$06] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam M. Szalkowski. GrGen: A Fast SPO-Based Graph Rewriting Tool. pages 383 – 397, 2006.

[gbi15] Site for the Global Biodiversity Information Facility (GBIF). Global Biodiversity Information Facility, `http://www.gbif.org`, 2015. Accessed: 10.Dec.2015.

[GBLZ$^+$15] Alejandra González-Beltrán, Peter Li, Jun Zhao, Maria S. Avila-Garcia, Marco Roos, Mark Thompson, Eelke van der Horst, Rajaram Kaliyaperumal, et al. From Peer-Reviewed to Peer-Reproduced in Scholarly Publishing: The Complementary Roles of Data Models and Workflows in Bioinformatics. *PLoS ONE*, 10(7), 2015.

[GBM$^+$11] Krzysztof Gorgolewski, Christopher D Burns, Cindee Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in Neuroinformatics*, 5, 08 2011.

[GBMSRS14] Alejandra Gonzalez-Beltran, Eamonn Maguire, Susanna-Assunta

Sansone, and Philippe Rocca-Serra. linkedisa: semantic representation of isa-tab experimental metadata. *BMC Bioinformatics*, 15(Suppl 14):S4, 2014.

[GCG+10] Yolanda Gil, James Cheney, Paul Groth, Olaf Hartig, Simon Miles, Luc Moreau, and Paolo Pinheiro daSilva. Final report of the w3c provenance incubator group, 2010. World Wide Web Consortium (W3C).

[GCG13] Daniel Garijo, Oscar Corcho, and Yolanda Gil. Detecting common scientific workflow fragments using templates and execution provenance. In *Proceedings of the Seventh International Conference on Knowledge Capture*, K-CAP '13, pages 33–40, New York, NY, USA, 2013. ACM.

[GCP13] Devarshi Ghoshal, Arun Chauhan, and Beth Plale. Static compiler analysis for workflow provenance. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, WORKS '13, pages 17–27, New York, NY, USA, 2013. ACM.

[GCT+12] Jeremy Goecks, Nate Coraor, The Galaxy Team, Anton Nekrutenko, and James Taylor. NGS analyses by visualization with Trackster. *Nature Biotechnology*, 30(11):1036–1039, November 2012.

[GDRB13] Carole Goble, David De Roure, and Sean Bechhofer. Accelerating scientists' knowledge turns. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 348 of *Communications in Computer and Information Science*, pages 3–25. Springer Berlin Heidelberg, 2013.

[Ge13] Paul Groth and Luc Moreau editors. PROV-Overview: An Overview of the PROV Family of Documents. W3C, `http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/`, 2013.

[GFG+09] Antoon Goderis, Paul Fisher, Andrew Gibson, Franck Tanoh, Katy Wolstencroft, David De Roure, and Carole A. Goble. Benchmarking workflow discovery: a case study from bioinformatics. *Concurrency and Computation: Practice and Experience*, 21(16):2052–2069, 2009.

[GG07] Ann G. Green and Myron P. Gutmann. Building partnerships among social science researchers, institutionbased repositories and domain specific data archives. *OCLC Systems & Services: International digital library perspectives*, 23(1):35–53, 2007.

[GG11] Daniel Garijo and Yolanda Gil. A new approach for publishing workflows: Abstractions, standards, and linked data. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 47–56, Seattle, 2011. ACM.

[GG12] Daniel Garijo and Yolanda Gil. Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data. In *Second International Workshop on Linked Science: Tackling Big Data (LISC), held in conjunction with the International Semantic Web Conference (ISWC)*, Boston, MA, 2012.

[GG14] Matthew Gamble and Carole Goble. Influence factor: Extending the prov model with a quantitative measure of influence. In *6th USENIX Workshop on the Theory and Practice of Provenance , TaPP'14*, Cologne, Germany, June 2014.

[GGCM12] Paul Groth, Yolanda Gil, James Cheney, and Simon Miles. Requirements for provenance on the web. *International Journal of Digital Curation*, 7(1), 2012.

[GGRF09] Yolanda Gil, Paul Groth, Varun Ratnakar, and Christian Fritz. Expressive reusable workflow templates. In *Proceedings of the Fifth IEEE International Conference on e-Science*, Oxford, UK, 2009.

[GGW$^{+}$09] Andrew Gibson, Matthew Gamble, Katy Wolstencroft, Tom Oinn, Carole Goble, Khalid Belhajjame, and Paolo Missier. The Data Playground: An Intuitive Workflow Specification Environment. *Future Generation Computer Systems*, 25:453–459, April 2009.

[Gio13] Renato De Giovanni. Workflow: Ecological niche modelling. myExperiment Repository, `http://www.myexperiment.org/workflows/3355/versions/27.html`, 2013.

[GKX$^+$11] Daniel Garijo, Sarah Kinnings, Li Xie, Lei Xie, Yinliang Zhang, Philip E. Bourne, and Yolanda Gil. Wings Drugome Workflow. `http://www.wings-workflows.org/drugome/index.php/Wings_workflows`, August 2011. Global Workflow.

[GL06] Volker Gruhn and Ralf Laue. Complexity metrics for business process models. In *Proceedings of the 9th International Conference on Business Information Systems, Lecture Notes in Informatics*, pages 1–12, 2006.

[GMWF11] Luiz M. R. Gadelha, Marta Mattoso, Michael Wilde, and Ian T. Foster. Provenance Query Patterns for Many-Task Scientific Computing. In *3rd Workshop on the Theory and Practice of Provenance, TaPP'11*, Heraklion Crete, Greece, June 2011.

[GNT10] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11:R86, 2010.

[Gob02] Carole Goble. Position statement: Musings on provenance, workflow workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Derivation and Provenance*, 2002.

[Gob11] Carole A. Goble. Accelerating scientists' knowledge turns. In *KDIR 2011 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Paris, France, 26-29 October, 2011*, page 7, 2011.

[Gol06] Jennifer Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In *International Provenance and Annotation Workshop, IPAW*, pages 101–108, 2006.

[GPEG$^+$10] Jose Manuel Gómez-Pérez, Michael Erdmann, Mark Greaves, Oscar Corcho, and Richard Benjamins. A framework and computer system for knowledge-level acquisition, representation, and reasoning with process knowledge. *International Journal of Human-Computer Studies*, 68(10), October 2010.

[GPS$^+$14] Nils Gehlenborg, Richard Park, Ilya Sytchev, Psalm Haseley, Stefan Luger, et al. Refinery platform: A foundation for integrative data visualization tools (poster), 2014.

[GRD$^+$06] Yolanda Gil, Varun Ratnakar, Ewa Deelman, Marc Spraragen, and Jihie Kim. Wings for Pegasus: A semantic approach to creating very large scientific workflows. In *18th Conference on Innovative Applications of Artificial Intelligence*, pages 10–11, 2006.

[Gre09] Jane Greenberg. Theoretical considerations of lifecycle modeling: An analysis of the dryad repository demonstrating automatic metadata propagation, inheritance, and value system adoption. *Cataloging and Classification Quarterly*, 47(3-4):380–402, 2009.

[GRH$^+$05] Belinda Giardine, Cathy Riemer, Ross C. Hardison, Richard Burhans, Prachi Shah, et al. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15:1451–1455, 2005.

[GRK$^+$11] Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro A. González-Calero, Paul T. Groth, Joshua Moody, and Ewa Deelman. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, 26(1):62–72, 2011.

[Gro05] Paul Groth. *The Origin of Data Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes*. PhD thesis, University of Southampton, UK, September 2005.

[Gro12] W3C Provenance Working Group. PROV Graph Layout Conventions. W3C, `https://www.w3.org/2011/prov/wiki/Diagrams`, december 2012.

[GSRRS13] Julian Garrido, Stian Soiland-Reyes, Jose Enrique Ruiz, and Susana Sanchez. Astrotaverna: tool for scientific workflows in astronomy. *Astrophysics Source Code Library*, 1:07007, 2013.

[GST$^+$02] Jim Gray, Alexander S. Szalay, Ani R. Thakar, Christopher Stoughton, and Jan Vandenberg. Online scientific data curation, publication, and archiving. In *SPIE Astronomy Telescopes and Instruments*, number MSR-TR-2002-74, page 6, August 2002.

[GT12] A. Gehani and D. Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings*, pages 101–120, 2012.

[Guo13] Phillip Guo. Data Science Workflow: Overview and Challenges. Communications of the ACM Blogs, `http://cacm.acm.org/blogs/blog-cacm/169199-data-science-workflow-overview-and-challenges/fulltext`, October 2013.

[GWCS09] Jane Greenberg, Hollie C. White, Sarah Carrier, and Ryan Scherle. A Metadata Best Practice for a Scientific Data Repository. *Journal of Library Metadata*, 9(3-4):194–212, November 2009.

[GWG+07] Carole Goble, Katy Wolstencroft, Antoon Goderis, Duncan Hull, Jun Zhao, Pinar Alper, et al. Knowledge Discovery for Biology with Taverna. In ChristopherJ.O. Baker and Kei-Hoi Cheung, editors, *Semantic Web*, pages 355–395. Springer US, 2007.

[GWMF12] Luiz M. Gadelha, Jr., Michael Wilde, Marta Mattoso, and Ian Foster. Mtcprov: A practical provenance query framework for many-task scientific computing. *Distrib. Parallel Databases*, 30(5-6):351–370, October 2012.

[HBM+08] David A. Holland, Uri Braun, Diana Maclean, Kiran-Kumar Muniswamy-Reddy, and Margo Seltzer. Choosing a data model and query language for provenance. In *Proceedings of the 2nd International Provenance and Annotation Workshop*, Salt Lake City, Utah, June 2008 2008.

[HCW08] Robert L. Hurt, Adrienne Gauthier Lars Lindberg Christensen, and Ryan Wyatt. Sharing images intelligently: The Astronomy Vizualisation Metadata standard. In *Communicating Astronomy with the Public*, page 450, June 2008.

[HDZ+14] Kristina Hettne, Harish Dharuri, Jun Zhao, Katherine Wolstencroft, Khalid Belhajjame, Stian Soiland-Reyes, Eleni Mina, Mark Thompson, Don Cruickshank, Lourdes Verdes-Montenegro, Julian Garrido, David de Roure, Oscar Corcho, Graham Klyne, Reinout van Schouwen, Peter A 't Hoen, Sean Bechhofer, Carole Goble, and Marco Roos. Structuring research methods and data with the research object model: genomics workflows as a case study. *Journal of Biomedical Semantics*, 5(1):41, 2014.

[Hec13] Reiko Heckel. Foundations of Model Transformations: A Lambda Calculus for MDD? `http://www.mimuw.edu.pl/~globan08/content/GlobanReiko.pdf`, 2013. Slides of a talk given at the GLOBAN Summer School Warsaw, 22-26 Sept. 2008.

[HG13] Rinke Hoekstra and Paul Groth. Linkitup: Link Discovery for Research Data. In *Discovery Informatics: AI Takes a Science-Centered View on Big DataAAAI Fall Symposium Series*, 2013.

[HGB$^+$13] Sonja Holl, Daniel Garijo, Khalid Belhajjame, Olav Zimmermann, Renato De Giovanni, Matthias Obst, and Carole Goble. On specifying and sharing scientific workflow optimization results using research objects. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, WORKS '13, pages 28–37, New York, NY, USA, 2013. ACM.

[HGP93] Michael Halper, James Geller, and Yehoshua Perl. Value propagation in object-oriented database part hierarchies. In *Proceedings of the Second International Conference on Information and Knowledge Management*, CIKM '93, pages 606–614, New York, NY, USA, 1993. ACM.

[HHRV15] Abel Hegedus, Akos Horvath, Istvan Rath, and Daniel Varro. A model-driven framework for guided design space exploration. *Automated Software Engineering*, 22(3):399–436, 2015.

[HKP$^+$09] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer.* W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[HKT02] Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In *Proceedings of the First International Conference on Graph Transformation*, ICGT '02, pages 161–176, London, UK, 2002. Springer-Verlag.

[HSC$^+$13] Kenneth Haug, Reza M Salek, Pablo Conesa, Janna Hastings, Paula de Matos, Mark Rijnbeek, Tejasvi Mahendraker, Mark Williams, Steffen Neumann, Philippe Rocca-Serra, et al. Metabolightsan open-access general-purpose repository for metabolomics studies and associated meta-data. *Nucleic Acids Research*, 2013.

[HSL$^+$04] Duncan Hull, Robert Stevens, Phillip Lord, Christopher Wroe, and Carole Goble. Treating shimantic web syndrome with ontologies. In *AKT Workshop on Semantic Web Services*, 2004.

[HTT09] Tony Hey, Stewart Tansley, and Kristin M. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[HWB$^+$12] Kristina M. Hettne, Katherine Wolstencroft, Khalid Belhajjame, Carole A. Goble, Eleni Mina, et al. Best Practices for Workflow Design: How to Prevent Workflow Decay. In *Proceedings of the 5th International Workshop on Semantic Web Applications and Tools for Life Sciences*, Paris, France, November 2012.

[ICF$^+$12] Robert Ikeda, Junsang Cho, Charlie Fang, Semih Salihoglu, Satoshi Torikai, and Jennifer Widom. Provenance-based debugging and drill-down in data-oriented workflows. In *IEEE 28th International Conference on Data Engineering (ICDE )*, pages 1249–1252, Washington, DC, USA (Arlington, Virginia), April 2012.

[JSV$^+$08] Rob Jelier, Martijn J Schuemie, Antoine Veldhoven, Lambert C J Dorssers, Guido Jenster, and Jan A Kors. Anni 2.0: a multipurpose text-mining tool for the life sciences. *Genome Biology*, 9(6):R96–R96, June 2008.

[KDG$^+$08] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun

Ratnakar. Provenance trails in the wings-pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, April 2008.

[KIT10] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 951–962, New York, NY, USA, 2010. ACM.

[KMB08] Hajo N. Krabbenhöft, Steffen Möller, and Daniel Bayer. Integrating ARC Grid Middleware with Taverna Workflows. *Bioinformatics*, 24(9):1221–1222, 2008.

[KSH12] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[Kul15] Robert Kulawik. Biostif web map application. Fraunhofer IAIS, `http://biostif.at.biovel.eu/biostif/main.jsp`, 2015.

[Kup07] Jan-Hendrik Kuperus. Nested quantification in graph transformation rules. Masters Thesis. University of Twente. `http://essay.utwente.nl/581/`, June 2007.

[LAB$^+$06] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B. Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[LAB$^+$09] Bertram Ludäscher, Ilkay Altintas, Shawn Bowers, Julian Cummings, et al. Scientific Process Automation and Workflow Management. In *Scientific Data Management*, Computational Science Series, chapter 13. Chapman & Hall, 2009.

[LAWG05] Phillip Lord, Pinar Alper, Chris Wroe, and Carole Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *The Semantic Web: Research and Applications*, volume 3532 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin Heidelberg, 2005.

[LB14] Barbara Lerner and Emery Boose. Rdatatracker: Collecting provenance in an interactive scripting environment. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*, Cologne, June 2014. USENIX Association.

[LHL13] B.Y. Lee, L.A. Haidari, and M.S. Lee. Modelling during an emergency: the 2009 {H1N1} influenza pandemic. *Clinical Microbiology and Infection*, 19(11):1014 – 1022, 2013.

[Li11] Peter Li. Using standardized bioinformatics formats in Taverna workflows for integrating biological data. The Bioinformatics Knowledge Blog, `http://bioinformatics.knowledgeblog.org/2011/06/21/using-standardized-bioinformatics-formats-in-taverna/workflows-for-integrating-biological-data/`, June 2011.

[Loh14] Steve Lohr. For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. `http://common-workflow-language.github.io/draft-2/`, August 2014.

[LPF05] Nicola Leone, Gerald Pfeifer, and Wolfgang Faber. The DLV Project - A Disjunctive Datalog System (and more). *http://www.dbai.tuwien.ac.at/proj/dlv/*, 2005.

[LPVM15] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, pages 1–37, 2015.

[LR06] Frank Leymann and Dieter Roller. Modeling business processes with BPEL4WS. *Information Systems and e-Business Management (ISeB)*, 4(3):265–284, Juli 2006.

[LRL$^+$12] Richard Littauer, Karthik Ram, Bertram LudŁscher, William Michener, and Rebecca Koskela. Trends in use of scientific workflows: Insights from a public repository and recommendations for best practice. *International Journal of Digital Curation*, 7(2):92–100, 2012.

[MAT10] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[MBC$^+$14] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. noworkflow: Capturing and analyzing provenance of scripts. In *5th International Provenance and Annotation Workshop (IPAW)*, Cologne, Jun 2014.

[MBG$^+$14] Paolo Missier, Jeremy Bryans, Carl Gamble, Vasa Curcin, and Roxana Danger. ProvAbs: model, policy, and tooling for abstracting PROV graphs. In *5th International Provenance and Annotation Workshop, IPAW*, Koln, Germany, 2014. Springer.

[MBJC14] Brian Matthews, Vasily Bunakov, Catherine Jones, and Shirley Crompton. Investigations as research objects within facilities science. In *Theory and Practice of Digital Libraries – TPDL 2013 Selected Workshops*, volume 416 of *Communications in Computer and Information Science*, pages 127–140. Springer International Publishing, 2014.

[MBL$^+$14] James Malone, Andy Brown, Allyson Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of Biomedical Semantics*, 5(1):25+, 2014.

[McC76] Thomas J. McCabe. A Complexity Measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320, Dec 1976.

[MCF$^+$11] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, et al. The open provenance model core specification (v1.1). *Future Gener. Comput. Syst.*, 27(6):743–756, June 2011.

[MCGP08] M. Esperanza Manso, José A. Cruz-Lemus, Marcela Genero, and Mario Piattini. Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study. In *Models in Software Engineering, Workshops and Symposia at MODELS*, pages 303–313, Toulouse, France, September 28 - October 3 2008.

[McW08] Hamish McWilliam. Pack: EMBL-EBI Web Services (retired). myExperiment Repository, `http://www.myexperiment.org/packs/6.html`, July 2008.

[MDB$^+$13] Paolo Missier, Saumen Dey, Khalid Belhajjame, Victor Cuevas, and Bertram Ludaescher. D-PROV: extending the PROV provenance model with workflow structure. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance TAPP'13*, Lombard, IL, 2013.

[Men00] Tom Mens. Conditional Graph Rewriting as a Domain-Independent Formalism for Software Evolution. In *Applications of Graph Transformations with Industrial Relevance*, volume 1779 of *Lecture Notes in Computer Science*, pages 127–143. Springer Berlin Heidelberg, 2000.

[Men05] Tom Mens. Graph-Transformation-Based Support for Model Evolution. `www.cs.le.ac.uk/events/segravis/material/mens-slides.pdf`, July 2005. Presentation given at Summer School on Generative and Transformational Techniques in Software Engineering, Braga, Portugal.

[Mer15] Merriam-Webster: Dictionary and Thesaurus. `http://www.merriam-webster.com`, 2015.

[MG11] Paolo Missier and Carole Goble. Workflows to open provenance graphs, round-trip. *Future Gener. Comput. Syst.*, 27(6):812–819, June 2011.

[MGBRS$^+$15] Eamonn Maguire, Alejandra Gonzalez-Beltran, Philippe Rocca-Serra, Susanna-Assunta Sansone, and Milo Thurston. BioSharing - a web-based curated and searchable registry of content standards, database and policies in the life sciences. 01 2015.

[MGM$^+$08a] Simon Miles, Paul T. Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. Extracting causal graphs from an open provenance data model. *Concurrency and Computation: Practice and Experience*, 20(5):577–586, 2008.

[MGM$^+$08b] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. The Provenance of Electronic Data. *Communications of the ACM*, 51:52–58, 2008.

[MGRtH35] Sara Migliorini, Mauro Gambini, Marcello La Rosa, and Arthur H.M. ter Hofstede. Pattern-based evaluation of scientific workflow management systems. Technical report, Queensland University of Technology, February 2011. Url: http://eprints.qut.edu.au/39935/.

[Mil56] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[Min13] Eleni Mina. Workflow: Annotate gene list with top ranking concepts. myExperiment Repository, `http://www.myexperiment.org/workflows/3921.html`, 2013.

[MKRI15] Hervé Ménager, Matúš Kalaš, Kristoffer Rapacki, and Jon Ison. Using registries to integrate bioinformatics tools and services into workbench environments. pages 1–6, 2015.

[MLA$^+$08] L. Moreau, B. Ludäscher, I. Altintas, et al. The first provenance challenge. *CCPE*, 20(5):409–418, April 2008.

[MMH$^+$15] Jessica Mink, Robert G. Mann, Robert Hanisch, Arnold Rots, Rob Seaman, Tim Jenness, Brian Thomas, and William OḾullane. The Past, Present, and Future of Astronomical Data Formats. In *Astronomical Society of the Pacific Conference Series*, volume 495, page 11, September 2015.

[MN12] Hailey Mooney and Mark Newton. The Anatomy of a Data Citation: Discovery, Reuse, and Credit. *Journal of Librarianship and Scholarly Communication*, 1(1), May 2012.

[Mon06] Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.

[Mor15] Luc Moreau. Aggregation by provenance types: A technique for summarising provenance graphs. In *Graphs As Models 2015*, volume 181, pages 129–144. Electronic Proceedings in Theoretical Computer Science, April 2015.

[MPB10] Paolo Missier, Norman W. Paton, and Khalid Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 299–310, New York, NY, USA, 2010. ACM.

[MPL11] Paolo Missier, Norman Paton, and Peter Li. Search computing. chapter Workflows for Information Integration in the Life Sciences, pages 215–225. Springer-Verlag, Berlin, Heidelberg, 2011.

[MRHBS06] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[MSFS11] Phillip Mates, Emanuele Santos, Juliana Freire, and Cláudio T. Silva. CrowdLabs: Social Analysis and Visualization for the Sciences. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 555–564, Berlin, Heidelberg, 2011. Springer-Verlag.

[MSK$^+$15] Timothy M. McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, et al. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *CoRR*, abs/1502.02403, 2015.

[MSRO$^+$10] Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Alexandra Nenadic, Ian Dunlop, Alan Williams, Tom Oinn, and Carole Goble. Taverna, reloaded. In *Proceedings of Scientific and Statistical Database Management Conference (SSDBM)*, volume 6187 of *Lecture Notes in Computer Science*, pages 471–481. Springer Berlin Heidelberg, 2010.

[MSZ$^+$10] Paolo Missier, Satya S. Sahoo, Jun Zhao, Carole Goble, and Amit Sheth. Janus: From Workflows to Semantic Provenance and Linked Open Data. In *Provenance and Annotation of Data and Processes*, volume 6378 of *Lecture Notes in Computer Science*, pages 129–141. Springer Berlin Heidelberg, 2010.

[MWT$^{+}$10] Marta Mattoso, Claudia Werner, Guilherme Horta Travassos, Vanessa Braganholo, Eduardo Ogasawara, Daniel Oliveira, Sergio Cruz, Wallace Martinho, and Leonardo Murta. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92, 2010.

[mye13] myExperiment Group Tags. `http://www.myexperiment.org/groups`, 2013. Accessed June 2013.

[nat15] Scientific Data, open-access journal. Nature Publishing Group `http://www.nature.com/sdata/`, 2015.

[Neo12] Neo4j. Neo4j - the worlds leading graph database. `http://www.neo4j.org/`, december 2012.

[NLG99] Walid A Najjar, Edward A Lee, and Guang R Gao. Advances in the dataflow computational model. *Parallel Computing*, 25(1314):1907 – 1929, 1999.

[NXB$^{+}$09] Qun Ni, Shouhuai Xu, Elisa Bertino, Ravi Sandhu, and Weili Han. An access control language for a general provenance model. In *Proceedings of the 6th VLDB Workshop on Secure Data Management*, SDM '09, pages 68–88, Berlin, Heidelberg, 2009. Springer-Verlag.

[OFC$^{+}$15] Tope Omitola, Andr Freitas, Edward Curry, San ORiain, Nicholas Gibbins, and Nigel Shadbolt. Capturing Interactive Data Transformation Operations Using Provenance Workflows. In *The Semantic Web: ESWC 2012 Satellite Events*, volume 7540 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin Heidelberg, 2015.

[OJK$^{+}$13] Silvia D. Olabarriaga, Mohammad Mahdi Jaghoori, Vladimir Korkhov, Barbera van Schaik, and Antoine van Kampen. Understanding workflows for distributed computing: Nitty-gritty details. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, WORKS '13, pages 68–76, New York, NY, USA, 2013. ACM.

[OWD$^{+}$04] Francois Ochsenbein, Roy Williams, Clive Davenhall, et al. VOTable format defintion. http://www.ivoa.net/Documents/latest/VOT.html, August 2004.

[PB99] Asunción Gómez Pérez and Richard Benjamins. Applications of ontologies and problem-solving methods. *AI Magazine*, 20(1), 1999.

[PBR$^+$14] CT Pfaff, H Bruelheide, S Ratcliffe, C Wirth, and K Nadrowski. Readable workflows need simple data [version 2; referees: 3 approved with reservations, 1 not approved]. *F1000Research*, 3(110), 2014.

[PCB$^+$14] Gabriele Pierantoni, Eoin Carley, Jason Byrne, David Perez-Suarez, and Peter Gallagher. A workflow-oriented approach to propagation models in heliophysics. *Computer Science*, 15(3):271, 2014.

[PG07] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[pip15] BIOVIA Pipeline Pilot Overview. `http://accelrys.com/products/datasheets/pipeline-pilot/pipeline-pilot-overview.pdf`, 2015. Accessed October 2015.

[Pla11] Beth Plale. Challenges and Opportunities of Workflow Systems in Environmental Research. In *Water Information Research and Development Alliance (WIRADA) Science Symposium*, Melbourne, AU, 08 2011.

[PMBVdS10] Alberto Pepe, Matthew Mayernik, Christine L. Borgman, and Herbert Van de Sompel. From artifacts to aggregations: Modeling scientific life cycles on the Semantic Web. *Journal of the American Society for Information Science and Technology*, 61(3):567–582, March 2010.

[PMMB12] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler. Hi-Fi: Collecting High-fidelity Whole-system Provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC ’12, pages 259–268, New York, NY, USA, 2012. ACM.

[PNNJ05] A. Powell, M. Nilsson, A. Naeve, and P. Johnston. Dublin core metadata initiative - abstract model, 2005. White Paper.

[pre11] *PREMIS Data Dictionary for Preservation Metadata*. version 2.1 edition, January 2011.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[PWH$^+$11] Beth Plale, Eran Chinthaka Withana, Chathura Herath, Kavitha Ch, Yuan Luo, and Felix Terkhorn. Strengths and Weaknesses of Sub-workflow Interoperability. Indiana Uuniversity Ttechnical Report, `https://www.cs.indiana.edu/ftp/techreports/TR699.pdf`, 2011. TR699.

[R C13] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[RBR$^+$05] Chris Rusbridge, Peter Burnhill, Seamus Ross, Peter Buneman, David Giaretta, Liz Lyon, and Malcolm Atkinson. The Digital Curation Centre: a vision for digital curation. Paper for From Local to Global Data. In *InteroperabilityChallenges and Technologies: 20 th - 24 th June 2005, Sardinia, Italy. IEEE Piscataway, NJ, USA*, pages 31–41, 2005.

[Ren03] Arend Rensink. The GROOVE simulator: A tool for state space generation. In *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 - October 1, 2003, Revised Selected and Invited Papers*, pages 479–485, 2003.

[RFP09] William Ribarsky, Brian Fisher, and William M. Pottenger. Science of analytical reasoning. *Information Visualization*, 8(4):254–262, December 2009.

[RG10] David De Roure and Carole Goble. Anchors in shifting sand: the primacy of method in the web of data. Event Dates: 26-27 April, 2010, March 2010.

[RLB00] Peter Rice, Ian Longed, and Alan Bleasby. Emboss: the european molecular biology open software suite. *Trends in Genetics*, 16(6):276–7, 2000.

[RM08] Hajo Reijers and Jan Mendling. Modularity in Process Models: Review and Effects. In *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 20–35. Springer Berlin Heidelberg, 2008.

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

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[SAM10] Kenneth F Schulz, Douglas G Altman, and David Moher. Consort 2010 statement: updated guidelines for reporting parallel group randomised trials. *BMJ*, 340, 2010.

[SBCBL14] Johannes Starlinger, Bryan Brancotte, Sarah Cohen-Boulakia, and Ulf Leser. Similarity search for scientific workflows. *Proceedings of the VLDB Endowment*, 7(12):1143–1154, August 2014.

[SBD$^+$09] Michael Stonebraker, Jacek Becla, David J. DeWitt, Kian Tat Lim, David Maier, Oliver Ratzesberger, and Stanley B. Zdonik. Requirements for science data bases and scidb. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*, 2009.

[SGB14] Manolis Stamatogiannakis, Paul T. Groth, and Herbert Bos. Looking inside the black-box: Capturing data provenance using dynamic instrumentation. In *Provenance and Annotation of Data and Processes - 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9-13, 2014. Revised Selected Papers*, pages 155–167, 2014.

[SGBB01] Robert D. Stevens, Carole A. Goble, Patricia Baker, and Andy Brass. A Classification of Tasks in Bioinformatics. *Bioinformatics*, 17(2):180–188, 2001.

[SHMG10] Jacek Sroka, Jan Hidders, Paolo Missier, and Carole Goble. A formal semantics for the Taverna 2 workflow model. *Journal of Computer and System Sciences*, 76(6):490 – 508, 2010.

[SKAC14] Carly Strasser, John Kunze, Stephen Abrams, and Patricia Cruse. DataUp: A tool to help researchers describe and share tabular data. *F1000Research*, 3(6), 2014.

[SKS$^+$08] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Cláudio Silva. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, April 2008.

[SL08] Nadine Schuurman and Agnieszka Leszczynski. Ontologies for bioinformatics. *Bioinformatics and Biology Insights*, 2:187–200, 2008.

[SLN$^+$09] Herbert Van Sompel, Carl Lagoze, Michael L. Nelson, Simeon Warner, Robert Sanderson, and Pete Johnston. Adding escience assets to the data web. *CoRR*, abs/0906.2135, 2009.

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

[SPG05] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[SPG08] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Karma2: Provenance management for data-driven workflows. *Int. J. Web Service Res.*, 5(2):1–22, 2008.

[SR10] Chris A. Silles and Andrew R. Runnalls. Provenance-Awareness in R. In *Provenance and Annotation of Data and Processes*, volume 6378 of *Lecture Notes in Computer Science*, pages 64–72. Springer Berlin Heidelberg, 2010.

[SRHB14] Stian Soiland-Reyes, Piotr Holubowicz, and Finn Bacall. scufl2-wfdesc 0.3.7, April 2014. See https://github.com/wf4ever/scufl2-wfdesc/releases for the command line tool download.

[SRSF$^+$12] Susanna-Assunta Sansone, Philippe Rocca-Serra, Dawn Field, Eamonn Maguire, et al. Toward interoperable bioscience data. *Nat Genet*, 44(2):121–126, February 2012.

[SSH08] Satya S. Sahoo, Amith Sheth, and Cory Henson. Semantic provenance for escience: Managing the deluge of scientific data. *Internet Computing, IEEE*, 12(4):46–54, 2008.

[Sta06] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[SWKH10] Jacek Sroka, Piotr Wlodarczyk, Lukasz Krupa, and Jan Hidders. Dfl designer: collection-oriented scientific workflows with petri nets and nested relational calculus. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, Wands '10, pages 5:1–5:6, New York, NY, USA, 2010. ACM.

[SZE$^+$14] Tam P. Sneddon, Xiao Si Zhe, Scott C. Edmunds, Peter Li, Laurie Goodman, and Christopher I. Hunter. Gigadb: promoting data dissemination and reproducibility. *Database*, 2014, 2014.

[TA$^+$11] Carol Tenopir, Suzie Allard, et al. Data sharing by scientists: Practices and perceptions. *PLoS ONE*, 6(6):e21101, 06 2011.

[Tay06a] Ian Taylor. Triana generations. In *e-Science*, page 143, 2006.

[Tay06b] Mark B. Taylor. STILTS - A Package for Command-Line Processing of Tabular Data. *Astronomical Data Analysis Software and Systems XV ASP Conference Series*, 351:266, 2006.

[TB94] Gabriele Taentzer and Martin Beyer. Amalgamated graph transformations and their use for specifying agg an algebraic graph grammar system. In *Graph Transformations in Computer Science, International Workshop*, volume 776 of *Lecture Notes in Computer Science*, pages 380–394, Dagstuhl, Germany, January 4-8 1994. Springer.

[TF08] Curt Tilmes and Albert J. Fleig. Provenance Tracking in an Earth

Science Data Processing System. In *Second International Provenance and Annotation Workshop, IPAW*, pages 221–228. Springer, June 2008.

[TFS$^+$08] Chris F Taylor, Dawn Field, Susanna-Assunta Sansone, Jan Aerts, et al. Promoting coherent minimum reporting guidelines for biological and biomedical investigations: the MIBBI project. *Nat Biotechnol*, 26(8):889–896, 2008.

[TMG$^+$07] Daniele Turi, Paolo Missier, Carole A. Goble, David De Roure, and Tom Oinn. Taverna Workflows: Syntax and Semantics. In *e-Science and Grid Computing, IEEE International Conference on*, pages 441–448, Dec 2007.

[TMN$^+$10] Wei Tan, Ravi K. Madduri, Aleksandra Nenadic, Stian Soiland-Reyes, Dinanath Sulakhe, Ian T. Foster, and Carole A. Goble. cagrid workflow toolkit: A taverna based workflow tool for cancer grid. *BMC Bioinformatics*, 11:542, 2010.

[VDA98] W. M. P. VAN DER AALST. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 08(01):21–66, 1998.

[vdAH05] Will M. P. van der Aalst and Arthur H. M. Hofstede. YAWL: Yet Another Workflow Language. *Information Systemse*, 30(4):245–275, June 2005.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[vHVDW13] Seth van Hooland, Ruben Verborgh, and Max De Wilde. Cleaning Data with OpenRefine. In Adam Crymble, Patrick Burns, and Nora McGregor, editors, *The Programming Historian*. August 2013.

[VSK$^+$08] Javier Vazquez-Salceda, Sergio lvarez, Tamas Kifor, et al. EU PROVENANCE Project: An Open Provenance Architecture for Distributed Applications. In *Agent Technology and e-Health*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 45–63. Birkhuser Basel, 2008.

[W3C12] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation. World Wide Web Consortium (W3C), `http://www.w3.org/TR/owl2-overview/`, December 2012.

[WDG$^+$15] John Wieczorek, Markus Doring, Renato De Giovanni, Tim Robertson, and Dave Vieglais. Reference for the simple darwin core standard. `http://rs.tdwg.org/dwc/terms/simple/index.htm`, jun 2015.

[WHW$^+$11] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A Language for Distributed Parallel Scripting. *Parallel Computing*, 37(9):633–652, September 2011.

[WKM$^+$10] David Withers, Edward Kawas, Luke McCarthy, Benjamin Vandervalk, and Mark Wilkinson. Semantically-Guided Workflow Construction in Taverna: The SADI and BioMoby Plug-Ins. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin Heidelberg, 2010.

[WLC14] David Wood, Markus Lanthaler, and Richard Cyganiak. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, urlhttps://www.w3.org/TR/rdf11-concepts/, February 2014.

[WM90] Y. Richard Wang and Stuart E. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *Proceedings of 16th International Conference on Very Large Data Bases (VLDB)*, pages 519–538, Brisbane, Queensland, Australia, August 13-16 1990. Morgan Kaufmann.

[WOE$^+$99] Marc Wenger, Francois Ochsenbein, Daniel Egret, et al. The SIMBAD Astronomical Database - the CDS Reference Database for Astronomical Objects, 1999.

[WOH$^+$12] Katy Wolstencroft, Stuart Owen, Matthew Horridge, Wolfgang Mueller, Finn Bacall, Jacky L. Snoep, Franco du Preez, Quyen Nguyen, Olga Krebs, and Carole A. Goble. Rightfield: Scientific knowledge acquisition by stealth through ontology-enabled spreadsheets. In *EKAW'12*, pages 438–441, 2012.

[WS97] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings of 13th International Conference on Data Engineering, 1997*, pages 91–102, Apr 1997.

[WVW$^+$09] Ingo Wassink, Paul E Van Der Vet, Katy Wolstencroft, Pieter B T Neerincx, Marco Roos, Han Rauwerda, and Timo M Breit. Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services. *International Congress on Services*, 2009(5):314–321, 2009.

[YB05] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, September 2005.

[ZSM$^+$11] Jun Zhao, Satya Sanket Sahoo, Paolo Missier, Amit P. Sheth, and Carole A. Goble. Extending semantic provenance into the web of data. *IEEE Internet Computing*, 15(1):40–48, 2011.

[ZWG$^+$04] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using Semantic Web Technologies for Representing E-science Provenance. In *The Semantic Web ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science*, pages 92–106. Springer Berlin Heidelberg, 2004.

# Appendix A

# Extended Provenance Literature

Earlier we discussed that *Scripts* and *Command-Line* tools are established modes in data processing. Provenance support here is designed to be non-intrusive to existing established mechanisms. We discuss the implications of non-intrusiveness, and observe the additional challenges it would bring to using provenance from these categories in reporting. On the other hand *Database* queries represent a restricted class of computation that yields white box provenance. Our intent in reviewing database provenance here is to understand what are the implication of having white-box provenance to the potential end-uses.

## A.1 Tool Provenance

Provenance collection from command-line tools for scientific data processing is studied in Sumatra [DMST14], Galaxy [GRH$^+$05], and ES3 [TF08] systems. In this category process provenance is collected

- by making command line tools invocations via a layer of wrapping, as in the Sumatra and Galaxy systems
- by monitoring the computational environment to pick events related to scientific tooling that is traced, as in the ES3 system

When a tool invocation is traced the information collected typically contains the name of executable, command line arguments, the environmental settings (OS version, hardware platform) at the time of invocation, file checksums. Tool provenance records data at the granularity of files, though those files often contain multiple data records.

Tool provenance can be prone to the data-wise n-by-m problem. A common characteristics of command line tools is a batch mode of operation. Often these tools can be run at once over collections of files/folders, which results in the generation of corresponding collections of files. As the provenance collection framework has no means to observe the underlying computation other than logging file reads and writes, the use of tools in batch mode would result in the n-by-m pattern .

| **System** | **Process Granularity** | **Data Granularity** | **Factorial Design Supported** | **Process Layering** | **Prone to n-by-m** | **Domain-Specific Metadata** | **End Use** |
|---|---|---|---|---|---|---|---|
| Sumatra | Wrapped Tool Invocations | Coarse (files) | N | N | Y | Tags, Parameters | Comparison, Experiment Bundling |
| Galaxy | Wrapped Tool Invocations | Coarse (files) | N | N | Y | Attributes | Workflow generation, Experiment Bundling |
| ES3 | Selected sys calls | Coarse (files) | N | Y | Y | N | Comparison |

Table A.1: Comparative Table of Black-Box Provenance in Tools

The Galaxy system is not an interception based system like others. It is both a workflow system and a command-shell front-end that is widely-used in bioinformatics. Galaxy differs in its collection approach by requiring that tools as well as datasets are wrapped so that they become part of the Galaxy analysis environment. Galaxy acts as a front end for tool invocations and tracks provenance such as invocation parameters and the consumed and produced datasets (physically represented as files). As mentioned in the previous section the Galaxy system mandates the creation of domain-specific metadata during tool wrapping and data upload. On the other hand the ES3 and Sumatra systems provide key-value type placeholders for such metadata, and expect it to be manually supplied by scientists.

A distinct application are for tool provenance is the ability to convert historical interactions with tools into analysis recipes that could be used in future. This feature is supported in Galaxy which converts series of tool invocations into a workflow. Galaxy can create workflows out of tool invocation provenance. Another important application is comparison. During exploratory phases of investigations scientists try to decide on the correct parameter settings or to locate data subsets that they will later use exhaustively. Tool provenance is helpful in comparing input settings for selected outputs or vice versa. Galaxy's increased emphasis on domain-specific metadata pays of in data publishing. Tool execution histories can be published as online supplementary material sections for articles reporting data-oriented investigation built with Galaxy.

## A.2 Script Provenance

An emerging approach in trail-type provenance is prospective provenance collection from scripts and retrospective provenance collection from script executions. Script provenance has been studied in RDataTracker [LB14] and CXXR [SR10], noWorkflow [MBC$^+$14], yesWorkflow [MSK$^+$15], and MTCProv [GWMF12] systems. Approaches can be broadly categorised as intrusive and non-intrusive depending on whether they require change to existing scripts.

The RDataTracker and yesWorkflow systems make up the intrusive category. RDataTracker provides an R library that scientists can use to extend scripts with logging statements reporting the structure and the runtime characteristics of analytical computation. When the script gets executed RDataTracker library calls log when a block of processing (that the users wants to trail) begins and ends, the variable assignments consumed and created by these blocks. A similar approach focused to prospective provenance is taken in yesWorkflow, which allows the user to superimpose workflow like computational elements (in the form of portS, computational blocks, and data channels among blocks) onto scripts through annotating R and Matlab scripts. Unlike RDataTracker yesWorkflow annotations are not executable statements, instead they are directives to generate structured documentation, e.g. workflow-like diagrams for scripts.

The CXXR, noWorkflow, and MTCProv systems make up the non-intrusive category. These systems collect provenance through an execution interception approach. Script execution can be intercepted at the level of commands/statements directed the programming language shell as in the CXXR systems, which tracks R command shell statements. Another way is to intercept function calls; the MTCProv system traces function calls in the Swift parallel scripting language [WHW$^+$11], whereas NoWorkflow traces calls made to user defined functions in Python.

| System | Process Granularity | Data Granularity | Fine-Grained Lineage | Process Layering | Domain-Specific Metadata | End Use |
|---|---|---|---|---|---|---|
| RDataTracker | R Shell Commands | Fine (R variable binding) | N/A | Y | N | Debugging |
| CXXR | R Shell Commands | Fine (R variable binding) | N/A | N | N | Debugging |
| NoWorkflow | Phyton Functions | Coarse (files) | N/A | Y | N | Debugging |
| MTCProv | Functions, Procedures | Fine (variable bindings) | N/A | Y | N | Debugging |
| yesWorkflow | N/A | N/A | N/A | N | N | Script Documentation |

Table A.2: Comparative Table of Black-Box Provenance in Scripts

Unlike the dataflow paradigm, scripting languages follow the control flow paradigm where statements are bound by control flow constructs. This poses significant challenge for the modelling power of scripts as conceptualisations of data-oriented scientific analyses [DBK$^+$15]. Process provenance in this realm is fine-grained in the granularity of program shell command, as in the RDataTracker ad CXXR systems, or functions in the case of MTCProv and NoWorkflow systems.

Data is commonly modelled at a fine-grain in the form of values of variable bindings at the time of occurrence of processes. Contrary to workflow provenance where typical queries ask for downstream or upstream lineage of data artefacts that bind to particular ports (variable) at run time, provenance in this category is not intended for lineage traversal over variables. This is because variables can be updated to have different values during the execution of a script. (The Swift language is an exception as it only allows single-assignment variables). Instead the information collected is intended to give outline dependencies of function or statement invocations and an understanding of the environment (in terms of variable values ) when a particular invocation has occurred. To cater for most basic data lineage the noWorkflow system intercepts file I/O of functions (similar to tool provenance) in order to record dependency among function invocations and files.

Retrospective provenance from scripts is eagerly collected during script execution. Interception based approaches are non intrusive yet they may result in an overwhelming amount of trace information. The script re-engineering approach of RDataTracker is intrusive to script development , on the other hand it allows tracing selectively and nesting/layering process provenance.

Script provenance is intended to be used as a debugging aid. In CXXR, noWorkflow and RDataTracker systems authors emphasise the visualisation of provenance as audit plots to help script developer have a better understanding of script executions

## A.3 Database Provenance

Research on database provenance has shown that lineage can be accurately tracked for standard relational algebra operators, Select, Project, Join and Union (SPJU) [BCTV04] and extended ones such as aggregation [CW01]. Cui and Widom [CW01] have been the first to provide such procedures for tracking why-provenance in data warehouse transformations. An alternative of inversion is to eagerly record tuple dependencies while executing queries. This approach requires queries to be customised to cater for provenance, this is achieved either by SQL query rewriting as in Polygen [WM90] or DBNotes [BCTV04], or having specialised query languages such as the TriQL of Trio [ABS$^+$06] system or the ProQL of the ORCHESTRA system [KIT10]. The primary application area of fine-grained lineage ( why provenance) is debugging queries or datasets. One could check for validity and consistency of datasets by tracing the source records that contribute to an unexpected or erroneous record (e.g. a spike value in a database view) [CW01].

In DBNotes authors exploit where-provenance to propagate annotations through SPJU queries. In addition to tuple annotations, provenance liabilities might come from innate characteristics of data such as the origin/host database or schema it belongs to. Polygen is an early system that also adopts a query-rewriting approach for annotation propagation, this time in multi-source data integration scenarios. Here whenever records from a source database are retrieved they are tainted with their "origin", and those taints get propagated during query evaluation. A notable capability in Polygen is tracking the intermediate sources, i.e. those that contribute to the computation- for instance with joins- but do not supply data values to the result record i.e. their contributions later get projected out.

Trio and ORCHESTRA systems use how provenance also for annotation propagation. However this time the additional information regarding nature of dependencies (joint witnesses, alternative witnesses) is used to create annotation algebras to be taken into account during annotation propagation. So rather than simply forwarding annotations, new annotations can be computed from existing ones based on algebraic operators associated with each dependency type. The Trio system on the other hand provides one particular (hard-coded) annotation algebra for computing uncertainty scores for results using source scores.

| **System** | **Process Granularity** | **Data Granularity** | **Lineage Accuracy** | **Process Layering** | **Lineage Transparency** | **Domain-Specific Metadata** | **Trace Procedure** | **Trace Collection** | **End Use** |
|---|---|---|---|---|---|---|---|---|---|
| Cui-Widom | Queries and UDFs | Fine(tuples) | Partial | N | N | N | External | Lazy | Debugging |
| Trio | Queries | Fine(tuples) | Full | N | how-provenance | Y (Uncertainty) | External | Eager | Uncertainty Propagation |
| DBNotes | Queries | Fine (tuple cells) | Full | N | where-provenance | Y | Embedded | Eager | Annotation Propagation |
| Polygen | Queries | Fine (tuple cells) | Full | N | where-provenance | Y (Origin DB) | Embedded | Eager | Origin Propagation |
| ORCHESTRA | Schema Mappings | Fine(tuples) | Full | N | how-provenance | Y | External | Eager | Annotation Algebras |
| SciDB | UDFs, Matrix Operations | Fine (array cells) | partial | N | N | N | External | Eager / Lazy | Debugging |
| Tioga | UDFs | Fine (array cells) | partial | N | N | N | External | Lazy | Debugging |
| Lipstick | PigLatin Operators | Fine(tuples) | partial | Y | N | N | External | Eager | What-if analysis |
| Panda | UDFs, Matrix Operations | Fine(tuples) | partial | N | N | N | External | Eager / Lazy | Debugging |

Table A.3: Comparative Table of Database Provenance

A common observation on query provenance is that not all (scientific) data processing can be restricted to querying, and similarly that not all scientific data is in relational form. Science-specific databases and dataflow programming have been used as extended modes of database-oriented scientific computing. A commonality in all systems we review, namely SciDB, Tioga, Lipstick and Panda is their inclusion of arbitrary computations, termed User Defined Functions (UDF), as well as white-box steps. The impact of having both kinds of computations is that lineage accuracy cannot be guaranteed for black-box operations.

Provenance for extended computations, such as matrix operations over array-based data structures has been studied in SciDB [SBD$^+$09] and Tioga systems [WS97]. SciDB is a scientific database system tuned to store array structured data, a format common in Astronomy. An important feature of SciDB is to create chains of data transformations including UDFs as well as matrix operations, for which cell-level and accurate why provenance can be provided by inversion. SciDB employs a combination of eager and lazy strategies to efficiently handle provenance of voluminous array structured data. The Tioga system allows scientists to create image-processing pipelines, comprised entirely of UDF steps using array structured imagery data. Tioga is notable in its treatment of “inversion functions”, as first-class entities. Tioga maintains a registry of inversion functions and allows pipeline developers to associate each image-processing step with an inversion function. This information is then used to lazily compute provenance of (fragments of) result imagery. Provenance from computations over scientific databases has been primarily used for debugging transformations and datasets.

Recently a “dataflow” model of data processing has emerged, which combines the traditional black-box workflow approach with white-box query operations [SWKH10] [ADD$^+$11]. These approaches formally build on Nested Relational Calculus (NRC) to represent computations [BNTW95]. NRC provides a richer programming model and data structure than basic SQL queries over flat relations. NRC embodies queries over nested relations, user defined functions, looping and conditionals. Techniques for tracking why and how provenance in NRC computations have been recently studied. The Lipstick system is one example of such dataflow systems where each processor is a mini program written in the PigLatin language (a more user friendly procedural alternative to the SQL language, which is formally mappable to NRC) Lipstick combines white-box how provenance tracking with the tracking of for nested/layered compositions of NRC-based processors. Lipstick demonstrates a what if analysis capability, a unique application area of how-provenance that checks whether the deletion of an input record from the dataflow inputs have a changing-effect on a selected output. Panda [ICF$^+$12] is another dataflow system comprised of SPJ white-boxes and black box steps. Instead of eagerly collecting all provenance Panda has lineage mappings automatically generated via static analysis of queries. These mappings could be lazily evaluated whenever the lineage of a dataflow result is probed. Another advantage of mappings is that they can be composed.which allows lineage tracking to skip over intermediary computations for efficiency.

# Appendix B

# Abstraction Primitives Integrity Guarantees

## B.1 Dataflow Preservation (Completeness)

**Definition B.1.** *A workflow production $p : L \rightarrow R$ for workflow graphs (as per Definition 4.5) is dataflow preserving if for any path among two port nodes in L there exist a path among corresponding nodes in R. (Correspondence based on node-id attribute).*

**Proposition B.2.** *Eliminate production is dataflow preserving.*

***Proof*** The matched and retained nodes for the eliminate production in Figure4.14 are external port nodes ($n4$ and $n7$). So we need to check preservation of paths among these nodes. The rule includes a delete specification for all dataflow paths that link among these two nodes that pass via the activity to be eliminated ($n0$). On the other hand for each such path the rule introduces an $iddf$, henceforth it is dataflow preserving.

**Proposition B.3.** *Collapse-Up production is dataflow preserving.*

***Proof*** There are four groups of matched an retained nodes in Figure4.16. Specifically

- ports ($n16$) that are sources of dataflow into inputs ($n13$) of upstream activity ($n0$)
- ports ($n4$) that are targets of dataflow from outputs ($n23$) of upstream activity ($n0$)

- ports ($n20$) that are sources of dataflow into inputs ($n22$) of to-be-collapsed activity ($n1$)
- ports ($n18$) that are targets of dataflow from outputs ($n14$) of to-be-collapsed activity ($n1$)

There are 3 possible flow cases among these 4 groups, (a) from $n16$ to $n4$ (b) from $n20$ to $n18$ and (c) $n16$ to $n18$. Clearly, case (a) is addressed as the paths connecting $n16$ to $n4$ passing via $n0$, more specifically $n16 \rightarrow n13 \rightarrow n0 \rightarrow n23 \rightarrow n4$ are replaced by new dataflow paths that pass via newly introduced activity $n5$, more specifically $n16 \rightarrow n6 \rightarrow n5 \rightarrow n12 \rightarrow n4$. Cases (b) and (c) are addressed similarly.

**Proposition B.4.** *Collapse-Down production is dataflow preserving.*

***Proof*** Similar to Collapse Up, Collapse Down is based on activity composition. The matched and retained nodes for this production are (from Figure4.16):

- ports ($n20$) that are sources of dataflow into each input ($n13$) of activity to-be-collapsed ($n0$). Note that this path is not designated with reference to $n0$, but designated indirectly with reference to $n1$ in order to be able to qualify this rule for multiple copies to be created one per downstream activity i.e. $n1$.
- ports ($n15$) that are sources of dataflow into inputs ($n22$) of downstream activity ($n1$)
- ports ($n10$) that are targets of dataflow from outputs ($n7$) of downstream activity ($n1$)

Here there are two possible paths that need preserving (a) from $n20$ to $n10$, and (b) $n15$ to $n10$. Case (a) is addressed by replacing all possible path instances of the pattern $n20 \rightarrow n13 \rightarrow n1 \rightarrow n7 \rightarrow n10$ with the pattern $n20 \rightarrow n18 \rightarrow n5 \rightarrow n12 \rightarrow n10$. Case (b) is addressed as pattern $n15 \rightarrow n22 \rightarrow n1 \rightarrow n7 \rightarrow n10$ is replaced with $n15 \rightarrow n11 \rightarrow n5 \rightarrow n12 \rightarrow n10$. Note that this primitive preserves dataflows, only in some cases redundantly. For the case of collapsing onto multiple downstream activities a dataflow path in the original graph may map to multiple paths in the result graph.

## B.2 Dataflow Reflection (Soundness)

**Definition B.5.** *A workflow production $p : L \rightarrow R$ is dataflow reflecting (or sound) if for any path among two port nodes in R there exist a path among corresponding nodes in L. (Correspondence based on node-id attribute).*

With respect to this definition :

**Proposition B.6.** *Collapse Up production is not dataflow reflecting.*

***Proof*** The composition primitive may introduce false dependency relations among ports. Inspect the rule in Figure4.16. Among the nodes retained by the application of this rule; $n20$ represent those that are the sources of dataflow arriving at the inputs of the activity to be collapsed up. $n4$ represent those that are the targets of dataflow originating from the outputs of the upstream activity. In $L$ here is no path from $n20$ to $n4$ [1], on the other hand in $R$ the newly introduced nodes and links create a path pattern $n20 \rightarrow n11 \rightarrow n5 \rightarrow n12 \rightarrow n4$.

**Proposition B.7.** *Collapse-Down production is dataflow reflecting.*

***Proof*** As we inspect the rule in Figure 4.18 we see that there are no extra paths introduced by RHS among retained nodes 15, $n10$, and $n20$. Both RHS and LHS of the rule contains paths from 15 to $n10$, and from $n20$ to $n10$. Despite being based on composition lie the collapse-up primitive, the collapse down primitive is dataflow preserving because it discards all outputs of the activity to be collapsed $n0$.

**Proposition B.8.** *Eliminate production is dataflow reflecting.*

***Proof*** Inspecting the rule in Figure4.14, the retained ports are $n4$ and $n7, the$introduction of an indirect dataflow dependency between these ports is achieved in a sub-rule that executes only if its outer rule executes which matches and discards a dataflow path among these nodes. Hence this production is sound.

## B.3 Validity

The guarantee of validity depends on the underlying mechanism of obtaining the output graph from the input. In our graph transformation based approach where productions may add new nodes and edges to the graph, we achieve this simply, by the use of type graphs. Type graphs identify *allowed graph vocabulary (nodes and edges)* in the LHS and RHS of a production, so naturally excluding false/invalid representations. The multiplicity feature of type graphs further define *allowed structures*, such as the necessity of an output port node to belong to a single activity.

**Proposition B.9.** *Collapse Up, Collapse Down and Eliminate productions are valid with respect to the type graph $T_W$.*

[1]there cannot be if L is a directed acyclic valid workflow description graph

## B.4 Acyclicity

**Definition B.10.** *A workflow production $p : L \rightarrow R$ for workflow graphs is acyclicity preserving if both L and R designates an acyclic patterns, i.e. where no node in L or R is reachable from itself. Given p satisfies this property, the for any acyclic input graph G the application of p on G would result in an acyclic graph H.*

**Proposition B.11.** *Eliminate production is acyclicity preserving.*

***Proof*** follows from this simple argument. The eliminate production can be broken down into two stages, 1) the saturation of the workflow graph with all inferred indirect (transitive) data flow dependencies and 2) the elimination of activities with designated motif attributes. Clearly, stage one preserves acyclicity as it corresponds to the computation of transitive closure of dataflow relations in a DAG , and at stage 2 we're simply eliminating nodes and edges from an acyclic graph, which preserves its acyclicity.

**Proposition B.12.** *Collapse Up production is acyclicity preserving.*

***Proof*** Abstraction by grouping of nodes is known to result in cycles when the group is not a convex hull, i.e. there are directed paths that goes outside the group and then arrives back in to a node in the group [DZL11]. More specifically for the rule in Figure 4.16the grouping of two activities, one upstream $n0$, and the other to-be-collapsed up $n1$, the group comprised of $n0$ and $n1$

1. is not a convex hull, when there exists a path that starts at $n0$, visits some node outside the group matched in the rule by $n8$ and arrives at $n1$.

2. is guaranteed to be a convex hull, when there exists no such path as defined in (1).

In the Collapse-up production we guarantee a convex-hull group by encoding case (1) above as a negative application condition (NAC), thereby guaranteeing case (2).

**Proposition B.13.** *Collapse Down production is acyclicity preserving.*

For this production we no longer need to have a prevention (NAC) for cycles (See Figure 4.18). Recall from our introduction of this primitive in Chapter 4 that in collapse down we (may) create multiple composites (groups), when the activity to be collapsed down matched by $n0$, has multiple downstream activities matched by a universally quantified pattern of $n1$. In such cases we do not need to worry about dataflow paths that start $n0$ and visit some node outside the group, because have groups exclusive groups containing a copy of $n0$ and each matched successor $n1$.

## B.5 Bipartiteness

Bipartiteness of provenance corresponds to having the ***data*** $\xleftarrow{\textbf{used}}$ ***activity*** $\xleftarrow{\textbf{generatedBy}}$ ***data*** pattern in retrospective provenance. Note however our abstraction primitives operate over prospective provenance. A bipartite pattern in retrospective provenance is enabled by the following pattern in prospective provenance: ***port*** $\xRightarrow{\textbf{df}}$ ***port*** $\xRightarrow{\textbf{inputOf}}$ ***activity*** $\xRightarrow{\textbf{hasOut}}$ ***port*** $\xRightarrow{\textbf{df}}$ ***port***. This is the general pattern matched in our productions for activity elimination and collapse. Note that the illustrated path here is the big picture. This big pictures is broken down into smaller pictures (inputs of activity, incoming links of inputs, outputs of activity, outgoing links of outputs etc.). Each sub-picture are handled by separate universally quantified (optionally matched) sub-rules.

A workflow description that is comprised of such and (only such) patterns can be used as a view over workflow execution provenance to give a bipartite account of data derivation. The *ports* in prospective provenance foretell *data* artefacts and their roles in retrospective provenance, similarly *inputOf* relation foretells usage relation (*used*) and *hasOut* foretells generation relation (*generatedby*). Note that the *df* relation and the ports at its two ends, manifests as one data artefact being fulfilling roles identified both ports. Henceforth the *df* relation is idempotent for bipartiteness of provenance.

**Proposition B.14.** *Eliminate production does not result in workflow abstractions that can be used as views capable of giving bipartite accounts of data derivation.*

***Proof*** This is simply due to the *iddf* relation introduced by Eliminate in Figure 4.14. Unlike the *df* relation the ports at the two ends of an *iddf* relation will manifest as two distinct artefacts, we know for a fact, because one is the input of the activity eliminated an the other is the output. When using a workflow abstraction with an *iddf* relation in it, we would have to result to using lineage ( *wasInfluencedBy*) relation among two artefacts as the data processing activity linking the two has been abstracted out. This additional third relation (lineage) means the account of data derivation that we give is no longer bipartite.

**Proposition B.15.** *Collapse (Up &Down) productions result in workflow abstractions that can be used as views capable of giving bipartite accounts of data derivation.*

***Proof*** If we look at the rules for these productions in Figures 4.16 & 4.18 respectively we can see that the the entire rule (LHS+RHS) is comprised of the *df*, *hasInput*, and *outputOf* relations. Hence the result of these production are capable of giving bipartite accounts.

# Appendix C

# Default Abstraction Policies

```
<rewriteRules configName="collapseNoStrategy">
    <rule motif="http://purl.org/net/wf-motifs#DataPreparation"
        primitive="collapseUp" />
    <rule motif="http://purl.org/net/wf-motifs#DataMovement"
        primitive="collapseUp" />
    <rule motif="http://purl.org/net/wf-motifs#DataPreparation"
        primitive="collapseDown" />
    <rule motif="http://purl.org/net/wf-motifs#DataMovement"
        primitive="collapseDown" />
</rewriteRules>

<rewriteRules configName="collapseWithStrategy">
    <rule motif="http://purl.org/net/wf-motifs#OutputExtraction"
        primitive="collapseUp" />
    <rule motif="http://purl.org/net/wf-motifs#Flatten"
        primitive="collapseUp" />
    <rule motif="http://purl.org/net/wf-motifs#InputAugmentation"
        primitive="collapseDown" />
    <rule motif="http://purl.org/net/wf-motifs#Split"
        primitive="collapseDown" />
</rewriteRules>

<rewriteRules configName="elimShims">
    <rule motif="http://purl.org/net/wf-motifs#DataPreparation"
        primitive="eliminate" />
    <rule motif="http://purl.org/net/wf-motifs#DataMovement"
        primitive="eliminate" />
</rewriteRules>
```

# Appendix D

# Sample Run of Analysis Rules

```
Pinar-ALPERs-MacBook-Pro:datalog pinarpink\$ ./dlv -silent
-N=99999 ./prog-input-2/axioms_depth-2
./prog-input-2/axioms_formula-2
./prog-input-2/axioms_reaching-2
./test-data/Ch5_Example.edb

process(concatStr), process(flattenList), process(flattenList2),
processOutput(concatStr, outstr), processOutput(flattenList, outstr),
processOutput(flattenList2, outstr),
processInput(concatStr, str1), processInput(concatStr, str2),
processInput(concatStr, str3), processInput(concatStr, str4),
processInput(flattenList, inlist), processInput(flattenList2, inlist),
hasLhbRoot(concatStr, uid1), hasLhbRoot(flattenList, uidf1),
hasLhbRoot(flattenList2, uidf3),
dataLink(dl1, w1, alphabet, concatStr, str1),
dataLink(dl2, w1, symbols, concatStr, str2),
dataLink(dl3, w1, cons, concatStr, str3),
dataLink(dl4, w1, numbers, concatStr, str4),
dataLink(dl5, concatStr, outstr, flattenList, inlist),
dataLink(dl6, flattenList, outstr, flattenList2, inlist),
dataLink(dl7, flattenList2, outstr, w1, result),
lhbNode(uid1, cross, concatStr), lhbNode(uid21, str1, concatStr),
lhbNode(uid22, dot, concatStr), lhbNode(uid23, str3, concatStr),
lhbNode(uid31, str2, concatStr), lhbNode(uid32, str4, concatStr),
lhbNode(uidf1, cross, flattenList), lhbNode(uidf2, inlist, flattenList),
lhbNode(uidf3, cross, flattenList2), lhbNode(uidf4, inlist, flattenList2),
hasChild(uid1, uid21, 0), hasChild(uid1, uid22, 1),
hasChild(uid1, uid23, 2), hasChild(uid22, uid31, 0),
hasChild(uid22, uid32, 1), hasChild(uidf1, uidf2, 0),
hasChild(uidf3, uidf4, 0),
context(ctxA, w1, alphabet, 0), context(ctxS, w1, symbols, 0),
workflow(w1),
workflowInput(w1, alphabet), workflowInput(w1, symbols),
workflowInput(w1, cons), workflowInput(w1, numbers),
workflowOutput(w1, result),
definedDepth(wl,alphabet,1), definedDepth(wl,symbols,1),
definedDepth(wl,cons,0), definedDepth(wl,numbers,1),
definedDepth(concatStr,str1,0), definedDepth(concatStr,str2,0),
definedDepth(concatStr,str3,0), definedDepth(concatStr,str4,0),
definedDepth(concatStr,outstr,0), definedDepth(flattenList,outstr,0),
definedDepth(flattenList,inlist,1), definedDepth(flattenList2,outstr,0),
definedDepth(flattenList2,inlist,1),
deltaDepth(wl,alphabet,0), deltaDepth(wl,symbols,0),
deltaDepth(wl,cons,0), deltaDepth(wl,numbers,0),
deltaDepth(concatStr,str1,1), deltaDepth(concatStr,str2,1),
deltaDepth(concatStr,str3,0), deltaDepth(concatStr,str4,1),
deltaDepth(concatStr,outstr,2), deltaDepth(flattenList,outstr,1),
deltaDepth(flattenList,inlist,1), deltaDepth(flattenList2,outstr,0),
deltaDepth(flattenList2,inlist,0),
predictedDepth(wl,alphabet,1), predictedDepth(wl,symbols,1),
predictedDepth(wl,cons,0), predictedDepth(wl,numbers,1),
predictedDepth(wl,result,0), predictedDepth(concatStr,str1,1),
predictedDepth(concatStr,str2,1), predictedDepth(concatStr,str3,0),
predictedDepth(concatStr,str4,1), predictedDepth(concatStr,outstr,2),
predictedDepth(flattenList,outstr,1), predictedDepth(flattenList,inlist,2),
predictedDepth(flattenList2,outstr,0), predictedDepth(flattenList2,inlist,1),
sizeCumulative(uid1,2), sizeCumulative(uid21,1),
sizeCumulative(uid22,2), sizeCumulative(uid23,2),
sizeCumulative(uid31,2), sizeCumulative(uid32,2),
sizeCumulative(uidf1,1), sizeCumulative(uidf2,1),
sizeCumulative(uidf3,0), sizeCumulative(uidf4,0),
iterated(dl1,1), iterated(dl2,1), iterated(dl4,1), iterated(dl5,1),
smooth(dl3), smooth(dl6), smooth(dl7),
lhf(uid1,0), lhf(uid21,0), lhf(uid22,1), lhf(uid23,2), lhf(uid31,1),
lhf(uid32,1), lhf(uidf1,0), lhf(uidf2,0), lhf(uidf3,0), lhf(uidf4,0),
depthMapping(concatStr,str1,1), depthMapping(concatStr,str2,2),
depthMapping(concatStr,str3,0), depthMapping(concatStr,str4,2),
depthMapping(flattenList,inlist,1), depthMapping(flattenList2,inlist,0),
reaches(ctxA,wl,alphabet,1), reaches(ctxA,concatStr,str1,1),
reaches(ctxA,concatStr,outstr,1), reaches(ctxA,flattenList,outstr,1),
reaches(ctxA,flattenList,inlist,1), reaches(ctxA,flattenList2,inlist,1),
reaches(ctxS,wl,symbols,1), reaches(ctxS,concatStr,str2,1),
reaches(ctxS,concatStr,outstr,2), reaches(ctxS,flattenList,inlist,2),
contextTruncated(ctxA,flattenList2,inlist,1),
contextTruncated(ctxS,flattenList,inlist,1),
contextPreserved(ctxA,concatStr,str1,0),
contextPreserved(ctxA,flattenList,inlist,0),
contextPreserved(ctxS,concatStr,str2,0)
```