Active provenance for Data-Intensive workflows: engaging users and developers

Alessandro Spinuso
Koninklijk Nederlands Meteorologisch Instituut
De Bilt, The Netherlands
spinuso@knmi.nl

Malcolm Atkinson
University of Edinburgh
Edinburgh, United Kingdom
mpa@staffmail.ed.ac.uk

Federica Magnoni
Istituto Nazionale Geofisica e Vulcanologia
Rome, Italy
federica.magnoni@ingv.it

***Abstract*—We present a practical approach for provenance capturing in Data-Intensive workflow systems. It provides contextualisation by recording injected domain metadata with the provenance stream. It offers control over lineage precision, combining automation with specified adaptations. We address provenance tasks such as extraction of domain metadata, injection of custom annotations, accuracy and integration of records from multiple independent workflows running in distributed contexts. To allow such flexibility, we introduce the concepts of programmable *Provenance Types* and *Provenance Configuration*. *Provenance Types* handle domain contextualisation and allow developers to model lineage patterns by re-defining API methods, composing easy-to-use extensions. *Provenance Configuration*, instead, enables users of a Data-Intensive workflow execution o prepare it for provenance capture, by configuring the attribution of *Provenance Types* to components and by specifying grouping into semantic clusters. This enables better searches over the lineage records. *Provenance Types* and *Provenance Configuration* are demonstrated in a system being used by computational seismologists. It is based on an extended provenance model, S-PROV.**

***Index Terms*—Reproducibility, Workflow management software, Metadata, Data flow computing, Collaborative work**

I. Introduction

Scientific communities are building platforms where Data-Intensive scalable computing systems (DISC) [26] are crucial to conduct their research campaigns. Such systems often use workflows to express and combine independent processing tasks. The workflow languages used adopt abstractions to facilitate re-use of methods across application domains and computational environments. As a result, methods are often multi-disciplinary and co-developed by research-developers and data engineers. This requires a framework where professionals can cooperate while pursuing scientific challenges that present different levels of complexity concerning data, methods, organisation and distribution of the computations. Fostering dialog between the domains of expertise facilitates diagnosis, validation, reproducibility and management of both data and processes. Provenance recordings associated with the production of scientific results underpins these facilities. However, in Data-Intensive tasks, provenance information is sometimes too coarse or too detailed [29]. It may fail to capture information that would enable domain scientists to link the recorded traces with the high-level concepts and the data properties that are specific to their field. Moreover, lack in precision in describing data derivations could make traceability of results ineffective for a correct understanding of the method's behaviour [6]. This suggests the need for linage recording mechanisms that facilitate the contextualisation of the provenance with metadata linking to the purpose of the study. It also motivates improvements to the accuracy of the data-derivations in the scope of the combinatorial operators. These objectives are achieved with the introduction of a model for Data-Intensive operations, based on PROV [4] and a conceptual and technical framework that enables domain and user-driven contextualisation, precision tuning, selectivity and rapid exploitation. We address modern *Agile* systems [23] and in particular Data-Intensive workflow tools [19]. We encourage domain experts and engineers to collaborate in the analysis of the computational methods by presenting provenance as a *lingua franca*. The former can then focus on the concepts and terms established in their community, while the latter focus on how methods map to platforms. Our main contributions are the following.

Provenance representation for Data-Intensive workflow systems. The key elements to be recorded (*observables*) are described and linked in terms of their provenance relationships in a comprehensive model, that we call S-PROV, which makes use of and further specialises PROV constructs (Section III). The model includes concepts such as processors' *state*, data and metadata granularity. It describes the computational components and their distribution, abstracting the mapping to concrete executables [18] with special attention to streaming and stateful operations.

Integration of provenance capturing and analysis tools. We deliver a provenance framework integrated with the workflow system's abstractions that scales across different execution environments. It exposes rules and extensions supporting developers keeping dependencies consistent and precise. This improves the usability of the provenance records with respect to precision and domain-relevant metadata. We introduce the concepts of *Provenance Types* and *Provenance Configuration* (Sections IV, V) and show how their combination enables the contextualisation of the provenance information at runtime. This paper uses lineage visualisation and exploration tools developed in the context of this research [31]. It reports an

This work has been supported by the EU H2020 research and innovation programme under grant agreement No 777413.

approach and technologies that foster the *Active* interaction between users and a workflow's provenance mechanisms. Workflow developers and end users choose whether to control and tailor their provenance collection. The framework is available for data-streaming applications using `dispel4py` [19] mapped to several platforms. It is in use for demanding applications in seismology and climate-impact research, for the VERCE[1] [7] and CLIPC[2] projects, and now the DARE project[3]. The DARE architecture and vision, described in a contemporary paper [8], is the context for our recent achievements, illustrated with the seismic rapid assessment use case. (Section VI).

II. Related Work

In scientific computing a result is considered reliable when the scientific process that generated it is understood and agreed valid and it is (1) reproducible and (2) analysable for potential defects [10]. Reproducibility is one of the reasons why workflow management systems (WMS) gained momentum in science [5], [9], especially when they integrate solutions to capture prospective and retrospective provenance. The former describing the workflow variations in structure and parametrisation, and the latter showing the progressive effects on the data [13], also known as data-lineage. The provenance model ProvOne [2], based on the PROV specification, offers a framework to represent workflow related provenance by merging the two prospective and retrospective concepts. In this work we extend the model to allow a multi-layered interpretation of the lineage recordings. We address the challenge of making provenance pervasive, where technical details must be conveyed to the developers, while facilitating its exploitation by workflow users.

The benefits of combining domain and application specific metadata in the lineage recording are identified in recent works [15], [21]. More specifically, WMSs such as Wings [22] and SciCumulus [11], [12] show that lineage enriched with domain information increases the quality and relevance of the provenance records. The former demonstrates this guiding semantic-driven workflow composition, while the latter shows this enabling effective run-time monitoring. More recently, Deelman *et al*. [17] highlight the importance of keeping a human-in-the-loop and the crucial role of provenance for interactive analysis of the results produced by the new generation of HPC workflows. We show how users and developers contextualise the lineage with control over inclusion and activation. A preliminary version is actively used in an HPC context [7], where run-time validation and visualisation were essential requirements. To meet requirements we have added mechanisms to include in provenance traces, system- and user-specified properties about the data and the computation. We made these easy to specify using configurable provenance types, semantic attribution and selectivity controls. Configuration is decoupled from the workflow's computational logic so that types can be changed to switch between different lineage requirements, fostering cooperation between different experts by ensuring the information collected meets each of their needs and is correlated via the lineage data.

Recent work undertakes ways to capture more precisely dependencies of results and parameters. Dependencies become ambiguous when processing components merge or combine multiple input sources or results depend on accumulated state. By identifying these critical steps, workflow systems may facilitate Factorial Design techniques [6] to help developers disambiguate such dependencies. In our work we provide control over the precision of the traces and disambiguate when requested. This possibility is reflected in our provenance model, thanks to explicit representation of the process state, which makes possible to model stateful dataflow programs that perform complex multi-step computations. This is managed by the provenance type capturing a particular lineage pattern, thereby removing the need to change the workflow's logic.

Many WMSs adopt a database to store lineage data [24]. Other work emphasises queries over the stored data, while these are generated [30]. We present details about storage and access of lineage information in a separate paper. In our *Active* framework lineage is captured by each processing element and sent at a tuneable rate to a remote API [31] accessing a database as persistent storage, or saved on a local file store. The information is represented in a way that, while still compliant with the underlying model, services and tools can make it available to users at run time for rapid exploitation.

III. S-PROV: Resource Mapping and Stateful Operators

In this section we introduce a model (S-PROV) to represent the provenance of a Data-Intensive application, which is implemented by composing a set of streaming operators within an abstract workflow definition. Such applications can be executed at scale by different types of enactment engines seamlessly [19]. They implement a computational model where data streams transmit a sequence of data units and each operator, or processing element (PE), takes data from zero or more input streams and yields data on zero or more output streams. From the perspective of a provenance model, the input and output data of a PE are related in a *Derivation*, while the invocation of a PE to consume and produce data is interpreted as a provenance *Activity*. S-PROV imports and further extends these and other concepts from the PROV [4] and ProvONE [2] models. PROV is intended as a conceptual framework offering machine understandable descriptions of records that describe with contextual metadata people, institutions, entities, and activities involved in producing data. ProvONE offers an extension to more precisely characterise workflow processes.

ProvONE represents the structure of a *provone:Workflow* as a graph of interconnected entities of type *provone:Program*. To add specificity, in S-PROV we revise the description of the abstract workflow by introducing a new class, *Component* (the abstract workflow step), which extends the basic

[1]http://www.verce.eu
[2]http://www.clipc.eu
[3]http://www.project-dare.eu

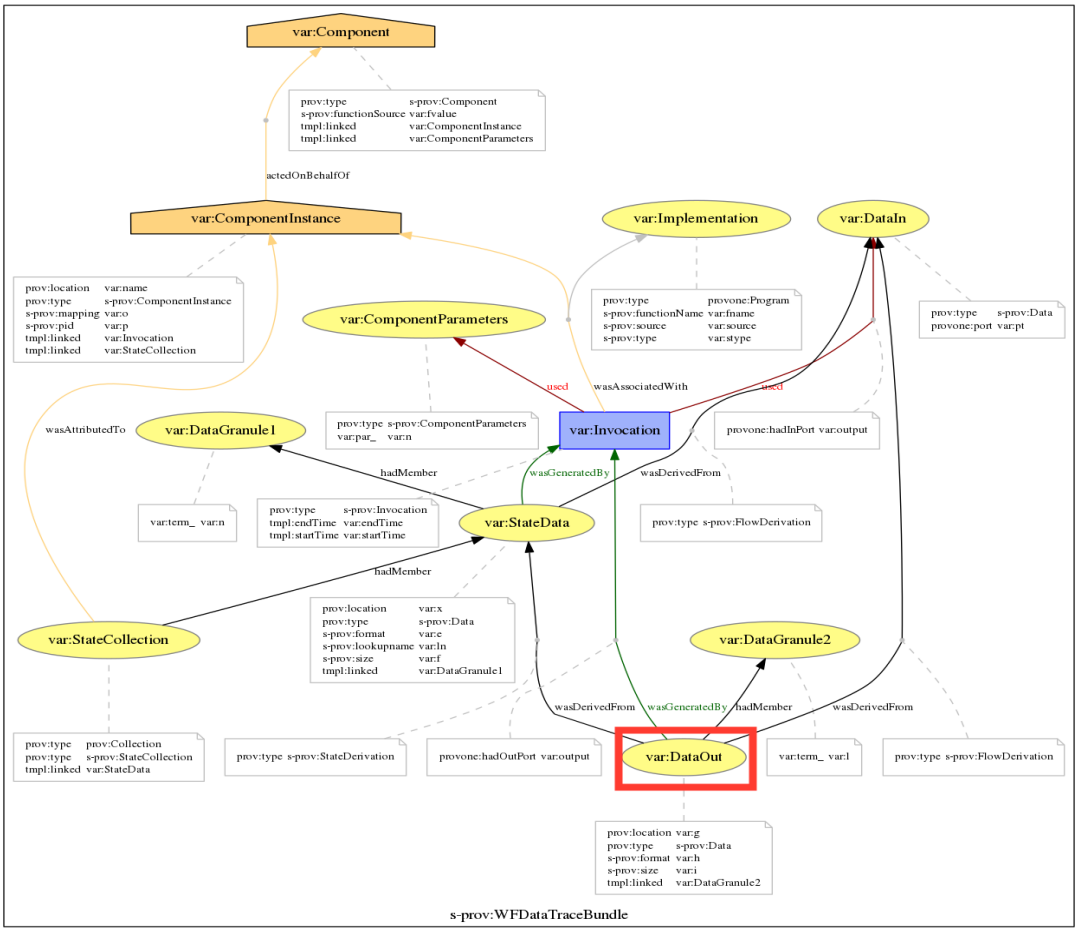

Fig. 1. S-PROV: The graph shows the provenance relationships captured by the S-PROV model between the entities, agents and activities involved in the generation of an output (*DataOut*). The *prov:type* property of each element specifies the name of the class. The two *wasDerivedFrom* relationships have a different *prov:type FlowDerivation* and *StateDerivation* depending on the stateless or stateful nature of the operation, as described in Table I

PROV class *prov:Agent*. A *Component* delegates the execution of a program to multiple instances of the program itself. We represent the instances in S-PROV by introducing the class *ComponentInstance* that extends the PROV class *prov:SoftwareAgent*. Making these semantics explicit enables us to represent detailed information about the execution of concurrent operators, which act on behalf of the same abstract *Component*. Information includes the location of the execution and the parametrisation (*ComponentParameters*) of each instance. The latter may change asynchronously at runtime, for instance as a consequence of a steering action [25]. ProvONE, through the concept of *provone:Workflow*, which is also a program, and the relationship *provone:hasSubProgram*, offers support for workflow encapsulation, which is an important and reusable feature of the model. Here the activity class of *provone:Execution* applies to workflows, as well as their sub-processes. We refine this concept in order to differentiate between the execution of a complete workflow and a sub-process. The former is described by the class *WFExecution*, the latter by the *Invocation* of an *ComponentInstance*. During an invocation the relationships between the input and output *Data* are established. The *StateCollection* contains references to the *Data* retained in the instance's internal state across multiple invocations. For instance, these can be represented by accumulated values and intermediate results. When stateful data contribute to new output, the data dependency is represented by a *StateDerivation*. This has the effect of obtaining a more precise lineage record, where also the component logic is made more explicit. Figure 1 shows S-PROV in its graphical representation. Finally, in S-PROV we combine system-level provenance with contextual information and metadata that are relevant to the users' interests and to the workflow's application domain, making provenance useful for the scientists, as well as for the workflow developer and system experts. The capture of such detail is partly automated, partly programmatically specified by the developer or offered as a set of configurable and reusable types – see Sections IV and V. S-PROV is available with an ontology[4] for evaluation and use.

[4]https://github.com/aspinuso/s-provenance/blob/master/resources/s-prov-o.owl

S-PROV Classes	Description
Component	The processing operator of the workflow in its abstract definition. Components are of type *prov:Agent*, they delegate to their instances the execution of a *provone:Program*, which may change at runtime.
ComponentInstance	This is a *prov:SoftwareAgent* that acts on behalf of the a *Component*. A *Component* delegates to multiple *ComponentInstances* as a result of the mapping of the workflow to the underlying resource for its execution. Its location is described by system process *pid* and *worker* node.
ComponentParameters	The set of parameters used by a *ComponentInstance*.
Invocation	The *prov:Activity* performed by a *ComponentInstance* over incoming data.
WFExecution	A *provone:Execution* describing a workflow run. From this activity associations to agents such as users and abstract components are established. It links to details about deployment and execution mode.
StateCollection	The internal state of a *ComponentInstance*. Contains references to *Data* items involved in stateful operations.
Data	Any self-contained collection of data produced or used by an *Invocation*.
DataGranule	A *DataGranule* is part of a *Data* collection and is described by domain-metadata.
FlowDerivation	A *prov:wasDerivedFrom* relationship between output and input data.
StateDerivation	A *prov:wasDerivedFrom* relationship between output data and a data product in the *StateCollection*.

TABLE I
OVERVIEW AND DESCRIPTION OF THE PROVENANCE ENTITIES OF THE S-PROV MODEL.

IV. PROGRAMMABLE PROVENANCE TYPES

Understanding and debugging DISC systems and the large-scale analytics they perform is a major challenge. It is helped by using lineage tracing mechanisms as an instrument to examine what has happened in standard terms [14]. Lineage metadata volume and scope are highly dependant on the requirements – why data-lineage is being recorded. This motivates our conceptual design and a technical realisation of a framework enabling the modular and customisable specification of the provenance produced by a Data-Intensive workflow application. We address use cases where users and developers want to tune the level of detail and precision of the traces and extract metadata from the workflow steps, adopting vocabularies that match their domain and current research. Details include a baseline of technical metadata about the single operations or steps, which are captured by the S-PROV model, but can also involve data quality test results, reference data values, according to specified controls.

Our approach considers a workflow processing element defined by a class, according to the Object-Oriented paradigm. The class defines the behaviour of its instances as their type, which specifies what an instance will do in terms of a set of methods. We introduce the *Provenance Type* and its specialisations, as the type that augments the basic behaviour of the processing element class with the capabilities that deliver provenance data. Different types capture specific lineage patterns, increasing specificity of dependencies between output and input data of each processor, and extract data properties according to established vocabularies or schemas. They enhance the workflow's process making it provenance-aware. This approach, as indicated in previous works [20], [27], tries to balance between automation, transparency and explicit intervention of the developer of a Data-Intensive tool, who can tune provenance-awareness through easy-to-use extensions. In order to reduce the coding efforts of the expert, we used these extensions to create a library of ready-made variants. Our conceptual framework gives to the provenance types the following characteristics.

Preserve native methods: types retain the original naming scheme for methods keeping their parameters and products unchanged.

Offer programmable extensions: extended signatures and optionally embeddable methods deliver control over lineage precision and granularity, for a provenance-aware representation of stateful operations and the implementation of reusable lineage patterns by research developers.

Handle the extraction of domain-metadata: types handle the automated extraction of metadata according to controlled vocabularies and community standards, as well as experimental and user-driven annotations.

Regulate selectivity, destination and rate of the lineage traces: types store traces at tuneable rates, connecting to different storage systems or remote lineage services. Lineage volume can be regulated selectively by specifying filters, *e.g.* on metadata-value ranges.

Trigger operations on data: types may embed metadata-driven messages within the traces. These are used by the workflow management mechanisms to trigger operations on the data concurrently to enactment. For instance, to transfer the data to remote locations or to cache locally for rapid re-use.

Finally, since the type approach focuses on the processing-element abstractions, it provides the foundation to automatically handle input and output traceability; it enables time-stamping and captures contextual information seamlessly, across different implementations and mappings to target execution engines. Although not imposed by the framework, we may distinguish between provenance **Pattern** and **Contextualisation Types** and use them in combination. The former capture lineage patterns by applying rules associated with the events triggered by the ingestion and production of data-streams within components. The latter are used to extract domain metadata from each new output produced, represented as a collection of *DataGranules* in S-PROV. Domain contextualisation can comply with a well-defined schema provided by an external ontology [22], but is not limited to that. New terms can be supported and injected into the provenance traces, serving the need of adding more experimental annotations.

Lineage Pattern Types	Description
SingleInvocationFlow	A workflow component that performs *stateless* operations; *e.g.* the processing element of a simple streaming pipeline where output data depends exclusively on the last received.
AccumulateFlow	A component whose output depends on a sequence of input data; *e.g.* computation of periodic average.
SlideFlow	A component whose output depends on computations over sliding-windows of input data; *e.g.* computation of rolling sums.
Nby1Flow	A component whose output depends on the data received on all its input ports in lock-step; *e.g.* combined analysis of multiple variables.
ASTGrouped (Accumulate State Trace Grouped)	This type manages accumulation operations in a stateful operator with grouping rules. It makes sure that data derivations are established between outputs and inputs related to same grouping index.
Contextualisation Types	**Description**
NetCDFType	Extracts metadata from data streams represented in *NetCDF* [1]. It guarantees consistency between the *ids* of the ingested data and the lineage entities by reusing the unique identifiers already contained in the format. This links lineage across multiple workflows.
SeismoType	Works on seismic data consisting of multiple time-series and represented in any of the formats supported by the *ObsPy* library [28]. Metadata match an experimental schema for seismology, SEIS-PROV [3].

TABLE II

PROVENANCE TYPES: A SELECTION OF EXISTING TYPES THAT SUPPORT COMMON LINEAGE PATTERNS AND METADATA EXTRACTION FOR SEISMOLOGICAL AND CLIMATE RESEARCH WORKFLOW APPLICATIONS

Provenance Configuration options	Description
Attribution Information	General information about the run, such as *username, runId, description, workflowName*, and its semantic characterisation *workflowType*.
Assign Provenance Types	One or more types can be namely assigned to each workflow component.
Assign Semantic Clusters	Components can be grouped into clusters (*s-prov:prov-cluster*), represented by a semantic class, to declare their participation to conceptual activities.
Selectivity-rules	A dictionary of metadata rules associated with a specific workflow component to selectively enable the recording of lineage information based on the data properties of its outputs.
Namespaces	Namespaces of the vocabularies or ontologies used in the provenance trace.
Provenance Storage mode	Users can choose if the provenance can be stored as files or sent to a remote service.

TABLE III

PROVENANCE CONFIGURATION: USERS CONFIGURE ANY OF THE OPTIONS IN THE TABLE TO CUSTOMISE THE LINEAGE CAPTURING OF A WORKFLOW EXECUTION.

A. Contextualisation and Precision API

Combining **Pattern Types** and **Contextualisation Types** gives the opportunity of capturing lineage patterns while handling the automated extraction of domain metadata to the traces, with no modification to the implementation of the processing element's class. Developers may create types that fit the components they implemented and record them, *e.g.* as components' properties in a catalogue. These could be suggested to users together with alternatives that capture lineage with less granularity, but that can still collect information and attributes of interest at a reduced overhead, making users aware of the implications. A *Provenance Type* exposes an abstract interface. In the following list we introduce a list of abstract methods whose implementation characterises the specific behaviour of a **Contextualisation Type**.

makeUniqueId: generates and returns the *id* assigned to every output of a processing element. An implementation of this method could adhere, for instance, to the *ids* generation policies of a particular data management plan.

extractDataSourceId: processes may ingest data produced by other workflows as their initial input, and the *id* of the input data could be reused by a provenance type when tracing the data derivation associated with a new output. This of course requires the input data to be represented by self-contained and structured formats carrying the data *id* as relevant metadata, *e.g.* the *NetCDF Attribute Convention*[5]. Depending on the format, the implementation of this method reads and returns the *id*, which is then reused by the PE, with the effect of linking the lineage across different workflow executions.

extractItemMetadata: This function is called automatically by a provenance type every time the data is written on an output port. It extracts metadata from the data written on a component's specific output port, according to a particular vocabulary. Metadata will be used to describe the *DataGranules* associated with the output.

addNamespacePrefix: making a provenance type aware of different namespaces allows developers to qualify the metadata terms extracted by the PE. The prefixes will be used consistently when exporting the lineage traces to semantic-web formats, such as RDF.

Besides domain contextualisation, our framework offers to developers additional methods to record accurate dependencies between output and input data. These are used to describe *FlowDerivations*, as well as *StateDerivations*, depending on the interaction of the PE's instance with its input channels and *StateCollection*, thereby implementing a reusable **Pattern Type**.

applyDerivationRule: This method is called automatically by a provenance type every time the data is written on an output port and every time an invocation ends. Its implemen-

[5]http://wiki.esipfed.org/index.php/Attribute_Convention_for_Data_Discovery_1-3

tation captures the data derivations between the output and the ingested data stream within stateless or stateful operations. It is used to record provenance that follows a specific and reusable lineage pattern. The events it reacts to are the following:

***write-event*:** occurs when a component "writes" data on any of its output ports. Before the actual writing, it triggers a rule that captures the provenance derivations associated with the new output.

***end-invocation-event*:** occurs at the end of an invocation, just before the process starts reading again. It also provides indications whether the invocation had previously produced any data, for instance, by setting a boolean value to a *void-invocation* variable. This event is used to trace intermediate data dependencies and stateful derivations across invocations.

Depending on the event and the pattern to be recorded, the rules may update the *StateCollection* (through the **updateProvState** method illustrated next), as well as combining other methods available that allow, for instance, to ignore the input (**ignorePastFlow**) or to reset the whole state, (**discardState**). For instance, to record the lineage of a process that computes the average calculation over a stream of input values we use the **AccumulationFlow** type, which is listed among others in Table II. Here data dependencies will be accumulated across *end-invocation-event*, until a *write-event* occurs. Thus, the output will be described by the complete set of derivations and the *StateCollection* will be reset to start tracing the new computation. In Section VI-A, we will discuss a more complex case of accumulation which takes into account grouping rules.

updateProvState: updates the *StateCollection* with a reference to a data entity, identified by a lookup tag. The lookup tag will allow developers to refer to the entity when this is used to derive new data.

write: this is the native write operation that activates the transfer of data between adjacent components of a workflow. Different systems may use different names, or simply use the *return* statement of a function. We call it *write* for clarity. It triggers the associated *write-event*, which is then captured by the **applyDerivationRule** method. Our framework extends this action with explicit provenance controls by superimposing a provenance type. The controls allow to add additional metadata associated with the output or to capture very specific lineage scenarios by indicating lookup tags to establish explicit data derivations between the output and the entities in the *StateCollection*.

Summarising, while the implementation of the **extractItemMetadata** is specific of the provenance type, the last two methods are used by developers and expert users to add metadata to the *Data* and *DataGranules* entities at runtime, allowing for further contextualisation of the lineage. Workflow practitioners may indicate details about formats, data location, as well as free-text messages, for instance, to report anomalies, run-time errors *etc.*. Metadata can be anonymous or qualified by the namespaces that are associated with the **addNamespacePrefix** method. Similarly, developers can include the PROV property

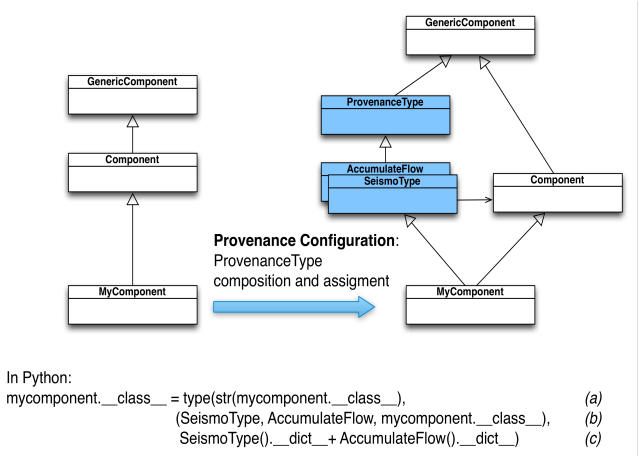

Fig. 2. Attribution of Provenance Types: The diagram shows the approach to the injection of a set of specific provenance types (*SeismoType, StatelessType*) for a workflow component. The figure also shows how metaprogramming is handled in native Python.

prov:type, in order to add further typing information to the data produced, *e.g.* a class of an external ontology. Table II shows a selection of the available pattern and contextualisation types. These types, intended to meet requirements in the climate and seismology domains, were co-developed with their experts. The types are available and documented in GitLab[6], together with the provenance type framework for `dispel4py`. We expect this library to be extended to support additional scenarios and data formats.

V. Provenance Configuration

Once the workflow is developed, users can separately specify the attribution of types, assign custom semantics to groups of components and declare selectivity controls. The options currently available are described in detail in Table III. In our implementation, the assignment of the provenance types is automatically handled by the framework which adopts metaprogramming[7] techniques. This design and implementation choice, explained in Figure 2, allows the attribution of provenance types by declaratively combining different categories of types. It keeps the independence from the underlying execution environment. It preserves native methods and handles provenance extraction seamlessly across development and production setups, where different engines will be used for the same calculation.

The availability of a configuration profile is of great use at different stages of a research investigation. Users start with identifying incrementally the useful types which are relevant for their findings. They may initially insert few metadata and use less precision in the the recording of the data derivations, for the sake of experimentation. Formal metadata and detailed precision can be applied when the research gets closer to

[6]https://gitlab.com/project-dare/dispel4py/blob/master/doc/dispel4py.provenance.rst

[7]https://en.wikipedia.org/wiki/Metaprogramming

its final results. That is, when the need for outreach to and reproducibility by other peers becomes increasingly likely and thereby provenance needs to be complete, precise and rigorous, as in the case for publication and curation in citable data repositories. When lineage is adopted in production systems, for monitoring and validation purposes, the flexible configuration approach helps data-managers limit the impact of provenance generation on the infrastructure's resources and performance. Especially in near real-time systems, *Selectivity-rules* based on metadata value ranges can narrow the focus of the lineage onto relevant situations, which could also trigger further operations on the outputs at runtime, *i.e.* feeding complementary processing tasks performed concurrently within independent execution context. However, managers can also mine from accumulated provenance records information for resource planning and to optimise operational practices. The decoupled configuration approach permits more generality and fosters collaboration between developers and domain experts when preparing and interpreting the lineage information. It encourages the implementation and the reuse of fundamental methods across disciplines, accommodating their provenance requirements.

VI. Test Case: Seismic Rapid Assessment

Computational seismology is presently facing the challenge of harnessing the power of highly accurate and computationally sophisticated seismic simulation tools and of managing increasing amounts of recorded and simulated data. The processing and usage of data requires tools to easily customise the available methods to keep up with the quickly evolving scientific community. Robust provenance-driven tools are needed to smartly organise storage of the data and to allow for data exploration, combination and reuse. This promotes reproducibility and error detection in scientific experiments. These needs become critically urgent after large seismic events, since reliable and rapid estimates of the earthquake impact are fundamental to coordinate emergency response. In this context, the rapid assessment of seismic ground motion (RA) is an application is the key requirement from computational seismology, embodying all the aforementioned needs.

After a large earthquake it is essential to rapidly simulate the propagation of seismic waveforms in surrounding areas in order to asses the earthquake's impact by quantitatively estimating specific ground motion parameters that are also significant for structural engineers. Moreover, comparison and integration of synthetic information with recorded ground motion data allows us to improve the characterisation of earthquake impact on ground behaviour.

The theoretical foundations and applicable procedures are well established, using the high-level steps represented in Figure 3. This scenario requires intermediate results to be properly described by usable metadata and traceable in the context of the generating process and workflow. This would foster their retrieval to allow their comparison and reuse and to diagnose, validate and fine-tune new experimental analyses.

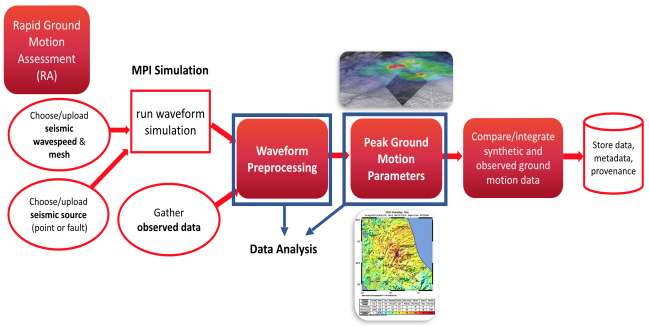

Fig. 3. The Rapid Assessment method analysing the impact of an earthquake composes re-usable tasks that run at different scales and require human supervision and intervention. The image highlights its data-analysis steps that exploit the *Active* provenance framework.

As a first step, the synthetic waveforms representing the ground motion are simulated. A seismologist selects the model and numerical code to solve the seismic-wave equation and supplies initial conditions. Then, the corresponding recorded seismograms for the earthquake are gathered from national and international servers.

At this point, in order to use both synthetics and data for the following analyses, they undergo typical seismological processing that has to be as similar as possible to make them consistent and comparable. This *Waveform pre-processing* stage, Figure 4, is fundamental to preparing seismological data. It is used in many other seismological applications such as seismic tomography and noise cross-correlation analyses. The results have associated provenance information about all the pre-processing steps (with related parameterisations) that they went through. This includes data properties after each step. These enable the detection of errors in analyses and the comparison of different ways of preparing the data. After pre-processing, the RA analysis requires the extraction of the ground-motion parameters from both the synthetic and recorded seismograms for subsequent comparison. In Figure 5 we present the *PGM Parameters* (Peak Ground Motion) workflow. The combined analysis is applied in parallel on both synthetic and observed data. For each considered seismic station, the horizontal components are processed to obtain Mean and Max norms, resulting in two outputs. Then, the workflow calculates the peak values for the two norms and groups them by station's name (*Match*). Eventually, these are compared and the results are stored in a file. Drawing on a strong scientific background, the *Waveform pre-processing* and *PGM Parameters* workflows are a valuable test-bed to demonstrating the benefits and flexibility offered by the *Active* provenance framework.

A. Seismic Analysis Workflows: Lineage precision and Metadata Extraction

We show in this section how to facilitate metadata customisation to track continuously evolving scientific methods. Moreover, we demonstrate how this combines with lineage

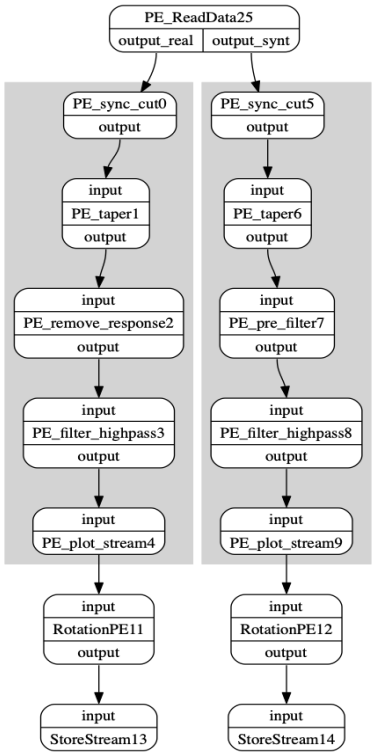

Fig. 4. Rapid Assessment: pre-processing pipeline for observed and synthetic seismological waveforms. The numbers after the processing elements' name are assigned by the `dispel4py` system to label the abstract workflow components.

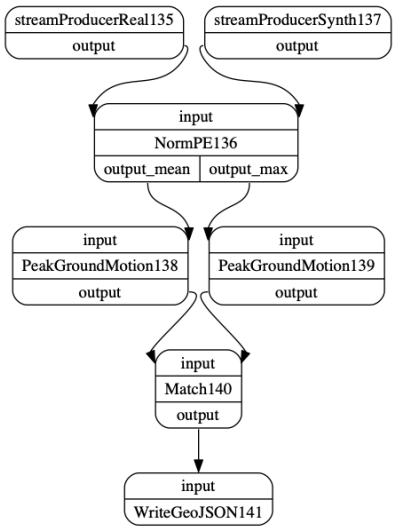

Fig. 5. Rapid Assessment: Workflow for the generation of Peak Ground Motion Parameters.

precision to reverse engineer a complex workflow's logic. We start from the Waveform pre-processing workflow in Figure 4. It is composed of simple stateless PEs receiving and producing waveform time-series according to a data format defined in the *Obspy* software package[8] [28]. The provenance type *SeismoType*, listed in Table II, can handle such a payload thanks to the implementation of its **extractItemMetadata** method, which uses the *Obspy* toolkit to access seismic data and metadata. By assigning such a provenance type to a PE, as shown in Listing 1, a user specifies that when the process is activated, it automatically populates the lineage with specified properties at runtime.

```
{ ... ,                                                          1
 's-prov:componentsType' :                                       2
 {'PE_ReadData': {'s-prov:type':['SeismoType'],                  3
  's-prov:prov-cluster':'seis:DataHandler'},                     4
                                                                 5
  'PE_taper': {'s-prov:type':['SeismoType'],                     6
  's-prov:prov-cluster':'seis:Processor'},                       7
                                                                 8
  'PE_remove_response': {'s-prov:type':['SeismoType'],           9
  's-prov:prov-cluster':'seis:Processor'},                       10
                                                                 11
  'PE_pre_filter': {'s-prov:type':['SeismoType']                 12
  's-prov:prov-cluster':'seis:Processor'},                       13
                                                                 14
  'StoreStream': {'s-prov:type':['SeismoType'],                  15
  's-prov:prov-cluster':'seis:DataHandler'}}}                    16
```

Listing 1. Rapid Assessment: extract of a provenance configuration json for *Waveform pre-processing* workflow. It shows the assignment of **Contextualisation Type** *SeismoPE* and the semantic clustering to relevant processing elements used by the workflow. Abstract components implemented by the same processing element will automatically assume the same provenance setup. *e.g. StoreStream13* and *StoreStream14*. Unless specified otherwise in the *s-prov:type* list, the default **Pattern Type** is the *SingleInvocationFlow*.

Alternatively, developers can specify additional metadata information to add to the lineage. In Listing 2 we show this for a simple plotting function implementing components *PE_plot_stream4* and *PE_plot_stream9* (Figure 4).

```
def plot_stream(stream, output_dir, tag):                        1
    stats = stream[0].stats                                      2
    filename = "%s.%s.%s.%s.png" %                               3
        (stats['network'], stats['station'],                     4
         stats['channel'], tag)                                  5
                                                                 6
    path = os.environ['STAGED_DATA'] + '/' + output_dir          7
    dest = os.path.join(path, filename)                          8
    stream.plot(outfile=dest)                                    9
                                                                 10
    prov = {'location': "file://" + socket.gethostname() +       11
                                "/" + dest,                      12
            'format': 'image/png',                               13
            'metadata': {'origin': tag}}                         14
                                                                 15
    return {'_d4p_prov': prov, '_d4p_data': stream}              16
```

Listing 2. Plotting function that ingests a seismic time-series stream, an output location and a tag describing its synthetic or observed origin. It produces an image and passes the stream to the next processing element. Lines 11-16 show the customised metadata that will be added to the lineage upon return. Here the volatile time-series is associated with its plot whose format and 'origin' are specified together with its current location for immediate access.

Depending on its position in the graph and its parametrisation, the function characterises the synthetic or observed "origin" of the produced image, information not otherwise included in the metadata captured by the *SeismoType*. In Figure 6 we provide a

[8]https://www.obspy.org/

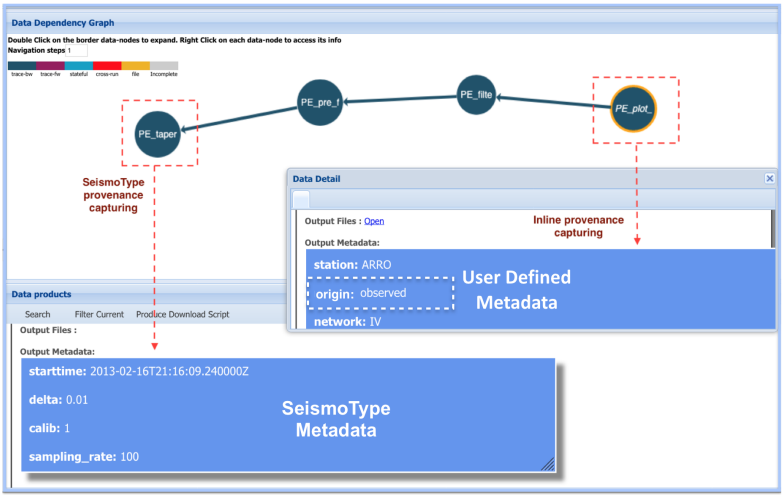

Fig. 6. Lineage Contextualisation: The image shows the derivations (*wasDerivedFrom*) relationships between data entities (circles) produced by the pre-processing workflow, from the *PE_plot* backwards to the *PE_taper* step. For visual compression we report in the circles' labels the name of the processing element generating the data. The yellow contour indicates that the data is associated with a materialised and accessible resource. In the figure, the data generated by the *PE_taper* (left) is described by the metadata extracted by the *SeismoType*, which is the type assigned to the processing element. Instead, the data labelled *PE_plot* (right) is described by metadata that are captured by combining both inline specifications and configuration.

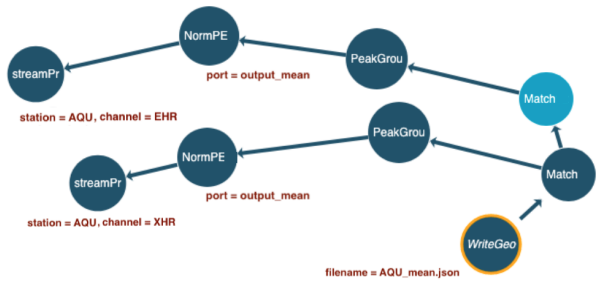

Fig. 7. Lineage Precision: The image shows the lineage of a file *AQU_mean.json* (yellow circle) produced by the *WriteGeoJSON* PE as a result of the the PGM workflow. The *wasDerivedFrom* relationships (arrows) show that the file was correctly derived from the mean norm values of the observed and synthetic data channels of the same station, respectively EHR and XHR. The light blue circle indicates a *StateDerivation*, revealing that the *Match* had stored the input data produced by a *PeakGroundMotion* processing element in its internal state before matching the data. This precision has been captured thanks to *ASTGrouped* pattern type assigned to the *Match* PE in the provenance configuration.

visualisation of the lineage recorded for the function's outputs. This is generated by the *S-ProvFlow* system[9] [31], which offers integrated tools and services for lineage analysis.

Another interesting case is the application of rules regulating the dependencies within *stateful* components that combine data from their inputs in various ways. We take as an explanatory example the *Match* component of the *PGM Parameters* workflow of Figure 5. This streaming operator matches incoming data based on the station code to extract ground-motion parameters between synthetic and observed data. Since many instances of the component are executed concurrently, a grouping policy ensures that one instance receives all the data with the same station code, and that different instances receive distinct subsets of station codes. The *grouping* rules are based on the data properties and foster deterministic matching and merging operations, realising a stream-oriented implementation of map-reduce [16]. This is a feature adopted by several systems, for instance *Apache Storm*[10], which is one of the execution engines supported by `dispel4py`. When capturing the lineage of a PE instance subject to grouping rules, we need to take into account that the data of different groups arrive without any particular order. These settings require developing the component to be a *stateful* operator, which involves updating and accessing intermediate stages of the acquired data for the right group. From the provenance perspective, we capture this lineage pattern with the *ASTGrouped* type (Table II) and assign it to the *Match* PE, as shown in Listing 3.

```
{ ... ,
 's-prov:componentsType' :
  {'streamProducerReal':{'s-prov:type':['SeismoType'],
   's-prov:prov-cluster':'seis:DataHandler'},

   'streamProducerSynth':{'s-prov:type':'[SeismoType'],
   's-prov:prov-cluster':'seis:DataHandler'},

   'Match':{'s-prov:type':['ASTGrouped'],
   's-prov:prov-cluster':'seis:DataHandler'}}}
```

Listing 3. Rapid Assessment: extract of a provenance configuration json for the *PGM Parameters* workflow. The *s-prov:type* list of the *Match* component indicates the choice for the *ASTGrouped* **Pattern Type**.

The **applyDerivationRule** method for the *ASTGrouped* updates the provenance state with a new entity whenever an *end-invocation-event* is triggered and no data was produced. This is reflected in the lineage with two derivations of the new entity, respectively with the current input and with the

[9]https://github.com/KNMI/s-provenance

[10]https://storm.apache.org/

one previously added to the state on the same grouping key. The key is computed by a hash function of the grouping term and its value. Finally, a *write-event* simply establishes the derivation of the output with the current input and with the entity in the provenance state related to the current grouping. As shown in Figure 7, by recording in the lineage the updates to a process state, we can precisely trace the data that contributed to the output, revealing complex logic without accessing the components' implementation. Considering the operator *stateless*, or capturing less detailed metadata would have produced incorrect, incomplete and thereby unusable provenance.

VII. Conclusions and Future Work

In this paper we presented an approach to provenance capturing in Data-Intensive workflows that balances between automation and human contribution. We introduced a conceptual design based on reusable and combinable *Provenance Types* that lead to the *Provenance Configuration* of a workflow for its provenance-aware execution. This is backed by a provenance model S-PROV, that accommodates complex lineage patterns. It represents details about the mapping of the abstract workflow to its distributed and concurrent enactment. Our framework aims at stimulating a versatile and *Active* participation of the domain experts and developers to improve the content and precision of the lineage information early in the research life-cycle, yielding improved utility. We have described its functional abstractions and discussed its usage and benefits through the effective adoption in an application for seismic rapid assessment. Here, the implementation of the framework for the `dispel4py` system demonstrates how lineage contextualisation is achieved by declaring simple statements directly in the processing element function, or through a separate configuration that assigns *Provenance Types* and semantics to the worklfow's components.

The incremental improvement of the relevance and usefulness of provenance increases confidence in the possibilities for its exploitation and therefore, promoting awareness of its importance. Through the *Active* participation of the expert, provenance is ready made for validation and results management use cases, exploring solutions to be applied to the next generation of WMSs [17]. Finally, we have adopted services and tools developed around our framework to demonstrate its effects (*S-ProvFlow*) [31]. In future work, we want to explore the combination of model and tools to tackle the challenges of lineage exploration for steering actions.

References

[1] NetCDF, Network Common Data Form. http://www.unidata.ucar.edu/software/netcdf/. Accessed: May 2019.

[2] ProvONE Data Model. https://purl.dataone.org/provone-v1-dev. Accessed: May 2019.

[3] SEIS-PROV: Provenance for Seismological Data. http://seismicdata.github.io/SEIS-PROV/. Accessed: June 2019.

[4] W3C PROV Data Model. http://www.w3.org/TR/prov-dm/. Accessed: June 2019.

[5] G. Alonso and C. Mohan. *Workflow Management: The Next Generation of Distributed Processing Tools*, pages 35–59. Springer US, Boston, MA, 1997.

[6] P. Alper, K. Belhajjame, and C. A. Goble. Static analysis of taverna workflows to predict provenance patterns. *Future Generation Computer Systems*, 75:310 – 329, 2017.

[7] M. Atkinson et al. VERCE delivers a productive e-science environment for seismology research. In *2015 IEEE 11th International Conference on e-Science*, pages 224–236, Aug 2015.

[8] M. Atkinson et al. Comprehensible control for researchers and developers facing data challenges. In *Proc. IEEE eScience 2019*, 2019.

[9] M. Atkinson, S. Gesing, J. Montagnat, and I. Taylor. Scientific workflows: Past, present and future, 2017.

[10] E. R. Boose et al. Ensuring reliable datasets for environmental models and forecasts. *Ecological Informatics*, 2(3 SPEC. ISS.):237–247, 2007.

[11] F. Costa et al. Enabling re-executions of parallel scientific workflows using runtime provenance data. In *Provenance and Annotation of Data and Processes*, volume 7525 of *Lecture Notes in Computer Science*, pages 229–232. Springer Berlin Heidelberg, 2012.

[12] F. Costa et al. Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach. *Proceedings of the Joint {EDBT/ICDT} 2013 Workshops*, pages 282–289, 2013.

[13] S. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1–6, 2008.

[14] S. De. *Newt: an architecture for lineage-based replay and debugging in DISC systems*. PhD thesis, UC San Diego, 2012.

[15] D. De Oliveira, V. Silva, and M. Mattoso. How Much Domain Data Should Be in Provenance Databases? *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance*, page 9, 2015.

[16] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. ACM.

[17] E. Deelman et al. The future of scientific workflows. *International Journal of High Performance Computing Applications*, 32(1):159–175, 2018.

[18] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[19] R. Filgueira, A. Krause, M. Atkinson, I. Klampanos, A. Spinuso, and S. Sanchez-Exposito. dispel4py: An agile framework for data-intensive escience. In *11th IEEE International Conference on e-Science*, pages 454–464. IEEE, 2015.

[20] I. Foster. The virtual data grid: a new model and architecture for data-intensive collaboration. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, SSDBM '03, page 11, Washington, DC, USA, 2003. IEEE Computer Society.

[21] L. M. R. Gadelha et al. MTCProv: a practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases*, 30(5):351–370, Oct 2012. Springer US.

[22] Y. Gil et al. Mind your metadata: Exploiting semantics for configuration, adaptation, and provenance in scientific workflows. *Lecture Notes in Computer Science*, 7032 LNCS(PART 2):65–80, 2011.

[23] R. Haberfellner and O. Weck. Agile systems engineering versus agile systems engineering. In *INCOSE International Symposium*, volume 15, pages 1449–1465. Wiley Online Library, 2005.

[24] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. A survey on provenance: What for? what form? what from? *The VLDB JournalThe International Journal on Very Large Data Bases*, 26(6):881–906, 2017.

[25] M. Mattoso et al. Dynamic steering of HPC scientific workflows: A survey. *Future Generation Computer Systems*, 46:100–113, 2015. Elsevier B.V.

[26] A. M. Middleton. Data-intensive technologies for cloud computing. In *Handbook of cloud computing*, pages 83–136. Springer, 2010.

[27] P. Missier, K. B. Jjame, J. Zhao, and C. Goble. Data lineage model for Taverna workflows with lightweight annotation requirements. *Ipaw*, 5272/2008:17–30, 2008.

[28] B. Moritz et al. ObsPy: A Python Toolbox for Seismology. *Seismological Research Letters*, 81(3):530–533, 2010.

[29] L. Murta et al. noWorkflow: Capturing and Analyzing Provenance of Scripts. In *Provenance and Annotation of Data and Processes*, volume 8628 of *Lecture Notes in Computer Science*, pages 71–83. Springer International Publishing, 2015.

[30] V. Silva et al. Raw data queries during data-intensive parallel workflow execution. *Future Generation Computer Systems*, 75:402–422, 2017.

[31] A. Spinuso. *Active provenance for data intensive research, PhD Thesis*. The University of Edinburgh, 2018.