
Design and Application of Provenance within Distributed Workflow Management Systems

Semantic Web Journal
2025, Vol. XX(X) 1-??
©COPYRIGHT INFO 2025
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/ToBeAssigned
journals.sagepub.com/home/trr

SAGE

Augustus Ellerm¹, Benjamin Adams², Mark Gahegan¹, Cornelis J. Drost, Ryan Chard³

Abstract

The growing adoption of computational workflows in scientific research requires robust mechanisms for recording and managing provenance information. Distributed and federated Workflow Management Systems complicate provenance capture: execution is divided across different sites, intermediate states are transient, and data are staged and restaged throughout the workflow execution. Building upon the RO-Crate ecosystem—which packages research outputs with their metadata to make them easier to share, reuse, and understand—we extended the Provenance Run Crate to support distributed WMSs contexts by designing the *Distributed Step Crate*, a per-step provenance fragment encoded within the distributed node itself. These fragments are collated within the *Distributed Provenance Crate*, resulting in a contiguous provenance record. We further propose an *Orchestration Crate* to contain multiple workflow provenance records for a combined view of provenance across complex, distributed workflows.

The *Distributed Provenance Crate* is demonstrated by its application to Globus Flows, a distributed workflow engine. The resulting provenance reproduces the step-level semantics of a reference, centralized Common Workflow Language *Provenance Run Crate* record, while extending this record with runtime, hardware, and access-control metadata. Finally, we discuss the conditions for the application of the *Distributed Step Crate* to production distributed WMSs. An open-source reference implementation and example RO-Crates accompany this paper. Together, these results illustrate how provenance can be effectively extended to distributed WMS contexts, enhancing transparency, reproducibility, and collaboration in scientific research.

¹Centre for eResearch, *University of Auckland*, New Zealand

²Department of Computer Science and Software Engineering, *University of Canterbury*, New Zealand

³Data Science and Learning, *Argonne National Laboratory*, Lemont, Illinois, USA

Corresponding author:

Augustus Ellerm, gus.ellerm@auckland.ac.nz

Introduction

The increasing adoption of computational workflows across scientific disciplines, aimed at automating data management and computation, has spurred the advancement of workflow provenance models and applications that capture semantic aspects of research processes. The W3C PROV standard defines provenance as “information about entities, activities, and people involved in producing a piece of data or thing” (1). Workflow provenance tools automatically capture this information to address the challenges of reproducibility, transparency, and portability within scientific workflows. As the size and complexity of data and computational workflows grow, so does the value of provenance systems that enable introspection and reuse of data and workflow components. The more than 300 Workflow Management Systems (WMSs) currently available reflect both their widespread adoption and the considerable heterogeneity of their application (2).

Many WMSs can capture detailed step-level provenance. They record the inputs, outputs, parameters, and runtime environment of each task. However, this is typically only possible when the entire workflow is executed within a single controlled environment (e.g., on a local machine, an HPC cluster, or a single cloud project). We refer to this pattern as a centralized deployment. By contrast, contemporary scientific workflows span multiple organizational boundaries and physical sites. Data may be acquired at an instrument facility, transferred to a supercomputing center, analyzed with a domain-specific virtual laboratory in the Cloud, and results stored within a university database. Distributed WMSs are designed to orchestrate these types of workflows, moving data between endpoints, scheduling computation, and coordinating credentials across services for users. However, few WMSs achieve both step-level provenance and this type of distributed orchestration. Table 1 illustrates the capabilities of four representative distributed WMSs. Some, such as Pegasus (3) and Nextflow (4), offer partial solutions; others, such as AiiDA (5), provide rich provenance in distributed settings within computational materials / chemistry, but are domain specific. Domain agnostic distributed WMSs typically lack robust provenance support for distributed contexts.

The assembly of complete and contiguous provenance records in federated and distributed WMSs is difficult. By complete, we mean that the provenance record for each workflow step captures its execution context, inputs, and outputs, wherever that step ran. By contiguous, we mean these per-step records are linked so that the causal lineage of data and results can be reconstructed across tasks, sites, and organizations. Encouragingly, recent improvements in tools such as Nextflow and those in the broader CWL ecosystem

Table 1. Comparison of four representative distributed WMSs based and their support for provenance recording.

WMS	Provenance Support
Pegasus (3)	Partial — Captures job-level provenance (software, parameters) but does not encode this within a formal provenance model.
Nextflow (4)	Partial — Natively provides logs and audit reports. Extensions such as <i>nf-prov</i> and <i>nf-provone</i> extend Nextflow’s provenance capabilities using the ProvONE model.
Globus Flows (6)	No — Only logs execution status; detailed provenance requires manual instrumentation or external tooling.
Aiida (5)	Yes — Rich provenance via a centralised AiiDA database for materials/chemistry workflows; not designed for cross-institution federation.

(e.g., Toil (7) and Arvados (8)) reflect a growing recognition of provenance as a core need in distributed workflows. However, these tools remain limited in maturity and interoperability. Provenance models for distributed and federated compute environments (e.g. ProvSearch (9) and HyperProvenance (10)) aim to enable interoperability, but wholly new provenance models often struggle to gain adoption across diverse research domains because they require new tooling and community engagement. Rather than introducing another bespoke provenance format, we build on RO-Crate, an established community-maintained standard for packaging machine-readable research objects and their metadata in a FAIR, JSON-LD format (11). RO-Crate is already used to package workflow executions, data products, and provenance within centralized WMS deployments, with the RO-Crate community recently introducing the Workflow Run RO-Crate (12) collection of profiles specifically to address workflow provenance.

In this paper, we address the challenge of distributed WMSs provenance by extending the Provenance Run Crate (13) so that provenance is captured within each participating node of a distributed workflow. Using this approach, provenance information is recorded at runtime (i.e. as each task executes) rather than being reconstructed by a central service. Capturing provenance in this way supports consistent replication, re-execution, and reuse of computational research (12, 14).

We propose the design of a sub-crate structure, the *Distributed Step Crate*, which is generated at runtime on a distributed node. This intermediate RO-Crate records discrete provenance information on a step-wise basis for each discrete task and ‘host’ in a workflow.

Multiple Distributed Step Crates are compiled into a Distributed Provenance Crate which describes the entirety of the distributed workflow’s execution. Finally, a parent RO-Crate structure is proposed, designed to represent a scientific project as a set of potentially disjoint workflow provenance records, and we discuss how generalized provenance representations can open the way towards executable meta-workflow containers, orchestrating multiple, specialized WMSs in distributed environments.

We evaluate the Distributed Provenance Crate in the context of Globus Flows (6), a service in the Globus ecosystem that coordinates the movement and computation of secure data across multiple institutional endpoints. Within this environment, we implement two classes of Globus Flows compute providers that generate a Distributed Step Crate at runtime for each task executed. These Crates are transferred to an orchestration service, which uses *Glacier* tools to manage the flow execution and assemble the Distributed Provenance Crate after workflow completion. Finally, Distributed Provenance Crates associated with the same project are collected into a single Orchestration Crate.

After providing background on WMSs and provenance in the next section, we detail the Distributed Step Crate and Distributed Provenance Crate profiles, derived from the Provenance Run Crate profile. Following this, the Distributed Provenance Crate is demonstrated with an application to the Globus Flows framework, illustrating a practical implementation in a real-world workflow management scenario. Finally, we conclude with reflections on the implications of our findings for provenance in WMSs and outline future research directions.

Background

WMSs play an increasingly important role in the scientific life cycle, facilitating the automation and management of complex computational and data-pipeline tasks. As highlighted by Goble et al. (15), the benefits of WMSs and workflows can be viewed from three different perspectives. 1) From an execution perspective, workflow platforms abstract away complex infrastructure challenges associated with managing data, securing access, and optimizing performance. 2) From a reuse and reproducibility perspective, these systems enable the packaging and modularisation of workflows, facilitating their execution across various environments. 3) Finally, from a reporting perspective, workflows act as comprehensive records of experimental designs and computational methodologies—capturing inputs, outputs, parameter configurations, and additional introspective metadata. Machine-actionable provenance

records support and enable these perspectives (12, 16–18). Clearly, semantic standards play a key role in delivering these benefits.

The bioinformatics domain has long been influential in the development and adoption of WMS technology, producing systems such as Galaxy (19), Taverna (20), and VisTrails (21), as well as workflow publication platforms such as WorkflowHub (22). As these tools have evolved and multiplied, there has been movement towards creating generalized WMSs to service many domains. Platforms such as Globus Flows (6) and Apache Airflow (23) provide domain-agnostic WMSs and offer robust methods of integrating distributed computing environments for workflow execution. In parallel to the development of these more generalizable WMSs, efforts such as the Common Workflow Language (24) aim to standardize workflow descriptions, enabling portability and interoperability between disparate WMSs.

Two key models for provenance representation are the Open Provenance Model (OPM) and PROV. Each treats provenance as a connected DAG, where nodes represent data and contextual entities, and edges the relationships between them. While both models are oriented to support interoperability, PROV provides a wider scope and extensibility with the well known PROV-O (and other) serialization. Similarly to WMSs, many application and domain specific provenance models have been developed over the years, including machine learning models (Prov-ML (25)), control-flow models (ProvONE+ (26)) and Biotechnology models (ISO 23494 (27)). There has also been work on generalizing models for multiple domains, such as CWLProv (28), built on the general workflow language CWL, and REPRODUCE-ME (29), which extends provenance modeling beyond traditional computational aspects to include human actors and external entities.

Within distributed/federated compute environments, tools such as ProvSearch (18) and HyperProvenance (10) have been proposed. However, it can be difficult for new models to be adopted in heterogeneous research communities. One strategy is to build on standards already widely used for describing digital research objects, such as RO-Crate, and toward that end, the RO-Crate community has created the Workflow Run RO-Crate collection of profiles (12). One of these profiles, the Provenance Run Crate profile (14) models both prospective and retrospective provenance information (Figure 1) and offers a standardized encoding that generalizes across WMSs, while supporting specialization through sub-profiles. In addition, it builds on a tool that is already widely adopted. This profile serve as the foundation for the approach discussed in this paper.

Provenance in a Distributed Setting

The RO-Crate **Provenance Run Crate** (13) (Figure 1) is designed to model step-level provenance within WMSs. It extends the **Workflow Run Crate**, which models the workflow as a black box with inputs and outputs. The Provenance Run Crate extends this representation with step-wise provenance modeling, enabling introspection of the structure and execution context of a workflow.

The Provenance Run Crate shown in Figure 1 is structured around two interrelated provenance graphs: a **prospective** (blue) graph, capturing the workflow plan, and a **retrospective** (red) graph, capturing the workflow execution (16). Within the prospective graph, each step is modeled as a **HowToStep** entity, describing the step’s software (**softwareApplication**), and the software’s inputs and outputs (**FormalParameter**).

The retrospective graph seeks to “open-up” the black box and extends the Workflow Run Crate with step-wise retrospective information. The model describes the execution of a WMS (**OrganizeAction**), which runs a workflow (**CreateAction**), which produces some result (**File** or **PropertyValue**). This models a workflow as a single step with associated results. Step-wise provenance is modeled through entities **ControlAction** and **CreateAction** (tool execution). The **ControlAction** entity models abstract behaviors, such as parallel execution or scattering in CWL, that a WMS may represent within its execution. The entity **CreateAction** (tool execution) represents the step itself, related to its software (**SoftwareApplication**) and results (**File** or **PropertyValue**).

Distributed Step Crate

When applying the Provenance Run Crate to a distributed compute environment, the first step is to identify the entities associated with individual tool executions within the profile. Each workflow **step** execution is represented by the entities **ControlAction** and **CreateAction** (tool execution). By extracting these entities and their forward dependencies, a subset of entities that model workflows steps within the Provenance Run Crate are isolated:

$$\begin{aligned} \text{ControlAction} &\xrightarrow{\text{instrument}} \text{HowToStep} \\ \text{ControlAction} &\xrightarrow{\text{object}} \text{CreateAction (tool execution)} \\ \text{CreateAction} &\xrightarrow{\text{instrument}} \text{SoftwareApplication} \\ \text{CreateAction} &\xrightarrow[\text{result}]{\text{object}} \text{File or PropertyValue} \end{aligned}$$

The resulting subgraph is illustrated in Figure 2.

Next, entities within the step execution subgraph are distinguished on the basis of scope, referring to their relation to either the distributed compute interface or

the orchestration interface. The orchestration interface is responsible for managing the workflow, while the distributed compute interface is responsible for the execution of individual steps.

The entities **ControlAction**, **HowToStep**, and **SoftwareApplication** are bound to the orchestration interface. These describe the behavior and metadata related to the management of individual steps (i.e., parallel behavior, position in flow, and software used). The entities **CreateAction** (tool execution) and **File** or **Property Value** are bound to the distributed compute interface, describing the provenance of the execution and results of the step itself.

These compute interface entities form the foundation of the Distributed Step Crate representing the provenance information of a single step in a distributed workflow. The Distributed Step Crate specification is shown in Figure 3. This specification models additional entities related to the **CreateAction** entity. These entities consist of **schema.org** types, unless explicitly stated otherwise.

The **CreateAction** entity is extended with **File**, **HowTo**, **Schedule**, **ActionAccessSpecification**, and **DistributedStep** types. These types provide a structured format for describing:

1. A file containing the script or tool executed on the distributed node,
2. The procedure of the tool execution,
3. Scheduling information regarding the distributed task,
4. The distributed access conditions to execute the step and
5. The environment of the computational node.

Six additional entities are associated with the model. First, the entities **HowToStep**, **MediaSubscription**, and **Organization** are defined:

- **CreateAction** $\xrightarrow{\text{step}}$ **HowToStep**
Embeds identity information such as the distributed jobs *identifier*, to uniquely identify each step within the distributed workflow.
- **CreateAction** $\xrightarrow{\text{requiresSubscription}}$ **MediaSubscription**
Models the authentication requirements that a distributed compute node may require. The **schema.org** field, **requiresSubscription**, may be of type **Boolean** or **MediaSubscription**, accommodating environments where no specification access control is required.

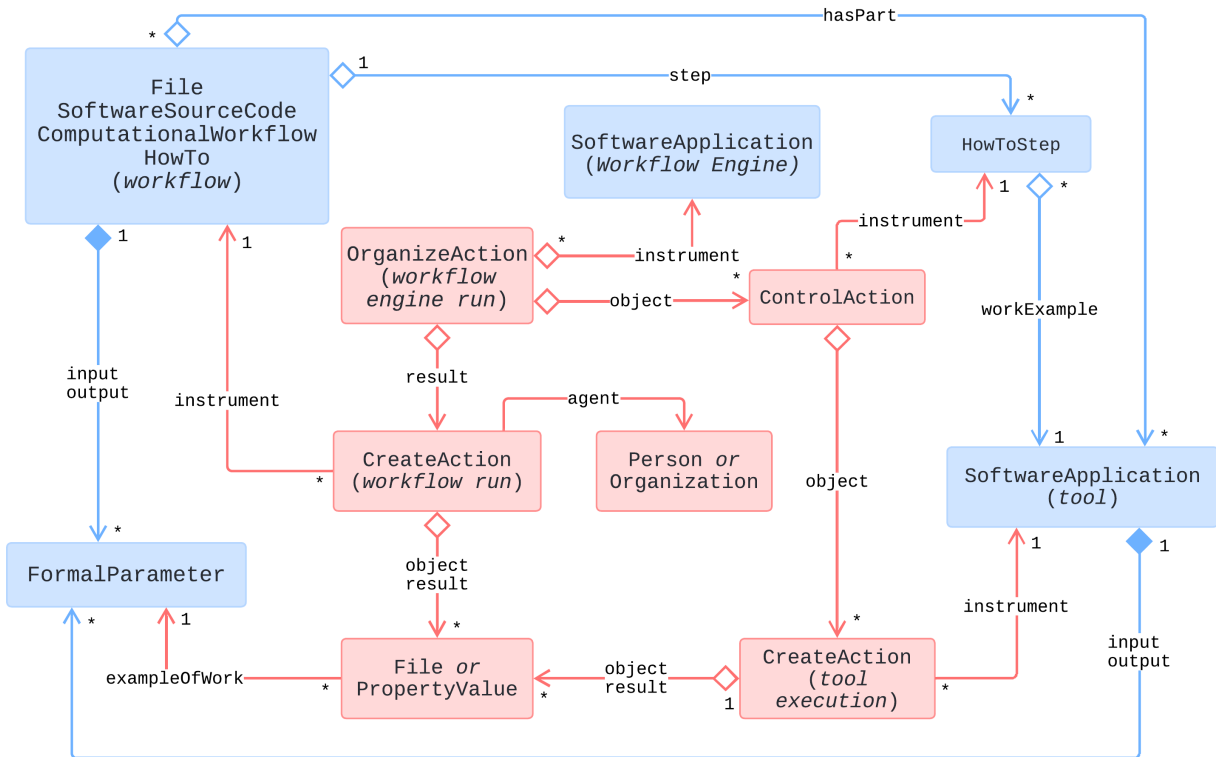


Figure 1. Provenance Run Crate Entity Graph describing prospective (blue) and retrospective (red) provenance; extracted from the Provenance Run Crate specification v0.5 (13).

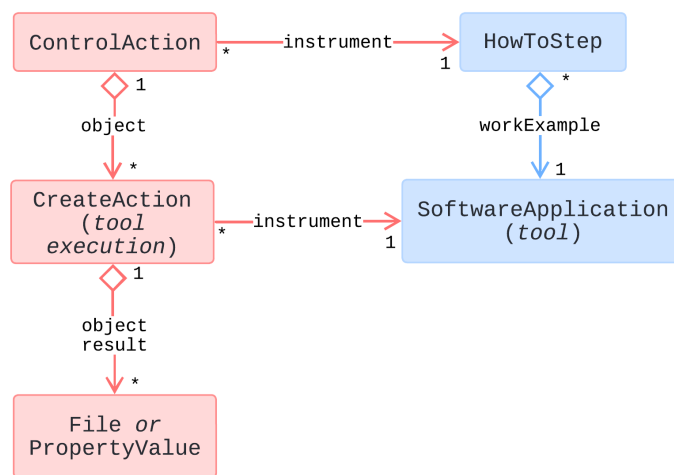


Figure 2. Step execution subgraph extracted from the Provenance Run Crate entity graph (Figure 1).

- **HowToStep** $\xrightarrow{\text{sourceOrganization}}$ **Organization**
Indicates that **Organization** is the source or provider of this specific distributed tool. For example, an organization authored, contributed, or is otherwise responsible for this distributed tool.
- **MediaSubscription** $\xrightarrow{\text{authenticator}}$ **Organization**
Indicates that an **Organization** is responsible for verifying, validating, or authorizing access to data that is behind a subscription (e.g., an organization such as Copernicus providing access to satellite data).

To accurately capture hardware specifications and performance metrics during the execution of distributed steps, the development of new types beyond the schema.org (30) framework is necessary. Although schema.org provides a type **SoftwareApplication** with parameters such as **processorRequirements** and **cpuRequirements**, these are intended to describe the constraints for software execution, a form of prospective information. Therefore, these existing types do not align semantically with the recording of retrospective environment information (31), particularly that of the performance and configuration of the hardware during the execution of the workflow.

Two new types, **HardwareRuntime** and **HardwareComponent**, are defined to document the performance and configuration of the hardware, respectively. **HardwareRuntime** is used to capture general hardware metadata from runtime, encoding a snapshot of the operational state during the execution of a distributed step. **HardwareComponent** details individual hardware components such as CPU, RAM, and GPU, documenting their performance and utilization. To integrate with existing

schema.org types, **HardwareRuntime** and **HardwareComponent**, build on established type hierarchies. Both follow the inheritance path of *Thing* > *CreativeWork* > *CustomType*, retaining the associated parameters.

Hardware types are integrated with the Distributed Step Crate through three relationships:

- **CreateAction** $\xrightarrow{\text{hasPart}}$ **HardwareRuntime**
Describes the configuration and containerization details of the hardware environment recorded during the execution of the distributed step.
- **HardwareRuntime** $\xrightarrow{\text{component}}$ **HardwareComponent**
Details individual components within the system (e.g. CPU, RAM, GPU).
- **HardwareComponent** $\xrightarrow{\text{performance}}$ **Observation**
Defined in schema.org, the **Observation** type enables records of a **HardwareComponent** to be described. The **Observation** type provides a structured way to detail both the measurement mechanism and actual measurement data of the associated component.

Together, these three entities enable the precise tracking and analysis of hardware performance within each Distributed Step Crate, enabling the documentation of system resources and their utilization (32, 33). The Distributed Step Crate profile provides a framework for recording provenance information directly from the source of distributed compute. It enables the encoding of input and output parameters and files, incorporates WMS-specific identifiers for cross-referencing against flow definitions, and includes authentication information (e.g. **OAuth-2 scopes**) to facilitate reuse and reproducibility.

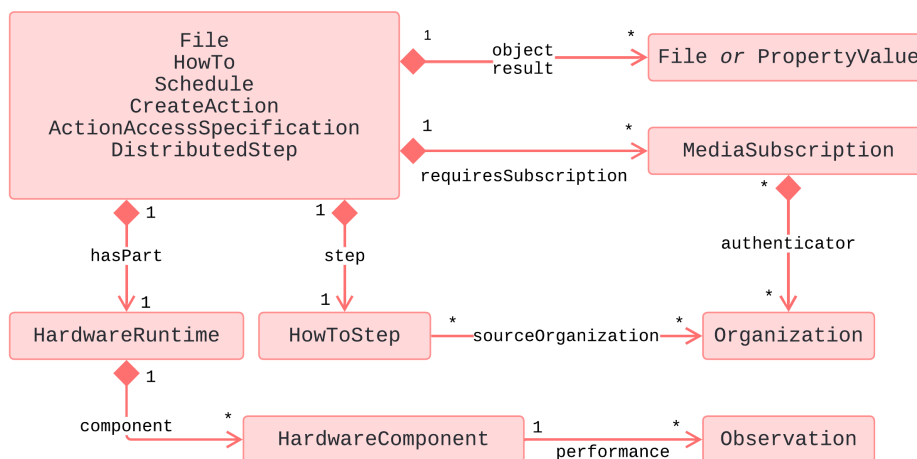


Figure 3. Distributed Step Crate Entity Graph.

Building the Distributed Provenance Crate

Given a set of Distributed Step Crates generated during the execution of a workflow, the next task is to instantiate the Distributed Provenance Crate, which describes the entire workflow execution. To do so, two assumptions regarding the WMS are made:

1. The WMS maintains a description of the workflow execution, including instantiated task identifiers for each distributed tool executed. This information is referred to as the Workflow Execution Description (WED).
2. Orchestrated distributed steps encode their globally identifiable task identifiers within the `HowToStep[identifier]` field of the Distributed Step Crate.

During the construction of the Distributed Provenance Crate, a cross-referencing process is used to align provenance fragments (Figure 4). Each Distributed Step Crate contains a unique task identifier (UUID) in its `HowToStep[identifier]` field. This identifier is cross-referenced with the WED to determine the workflow step to which the crate corresponds. This information is used to link each Distributed Step Crate to the appropriate `ControlAction` in the Provenance Run Crate.

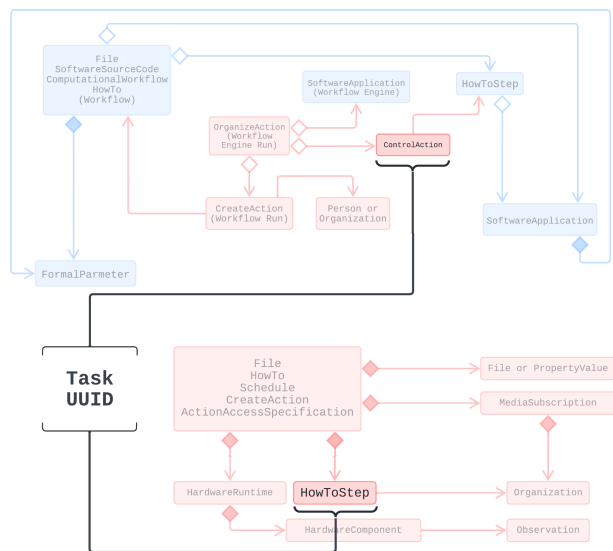


Figure 4. Distributed Step Crate alignment with parent Provenance Run Crate.

The Distributed Provenance Crate extends the Provenance Run Crate schema by encoding the additional information provided by the Distributed Step Crate, such as specific execution details and resource requirements. These extensions include:

- **Extension of CreateAction (tool execution) entity:** The `CreateAction` entity is extended with the `File`, `Schedule`, `ActionAccessSpecification`, and `DistributedStep` types.
- **Inclusion of new MediaSubscription and related Organization entities:** To capture authentication requirements and associated organizational information.
- **Inclusion of HardwareRuntime and related HardwareComponent and Observation entities:** To capture hardware runtime and component performance metrics.

The final specialized Distributed Provenance Crate entity graph is provided in Figure 5. The Distributed Step Crate and Distributed Provenance Crate schemas developed in this work are published in the LivePublication namespace on stable w3id IRIs (<https://w3id.org/livepublication/interface-schemas/>). Each module provides an RDF vocabulary, a JSON-LD context, and SHACL shape. The contexts are versioned and immutable and can be found [here](#) and resolve through [w3id.org](https://w3id.org/livepublication.org) to [livepublication.org](https://w3id.org/livepublication.org). We confirmed that these published contexts can be dereferenced and used with standard JSON-LD tools to expand our RO-Crate profiles into RDF graphs.

Orchestration Crate: Meta-workflow containerisation

With both the Distributed Provenance Crate and the Distributed Step Crate, it is possible to record the provenance of distributed workflow executions, enriched with information available from the participating nodes themselves. The focus now shifts from the provenance of individual computational workflows to the broader provenance of a scientific project *itself*.

Research projects increasingly rely on multiple workflows, and even workflow engines. The prevalence of meta-workflows (i.e., workflows which call on secondary workflows, possibly running on different infrastructures) highlights the need for provenance containers which are capable of representing multi-workflow provenance (34). Towards addressing this, the *Orchestration Crate* is introduced as a foundational structure for representing multi-flow provenance records.

To enable this representation, a generalized method is developed to containerize RO-Crates within parent crates. This approach requires each parent crate to reference the manifest files of its child crates, marking them with the type `RO-Crate`. Listing 1 illustrates a simple Orchestration Crate with a single child crate.

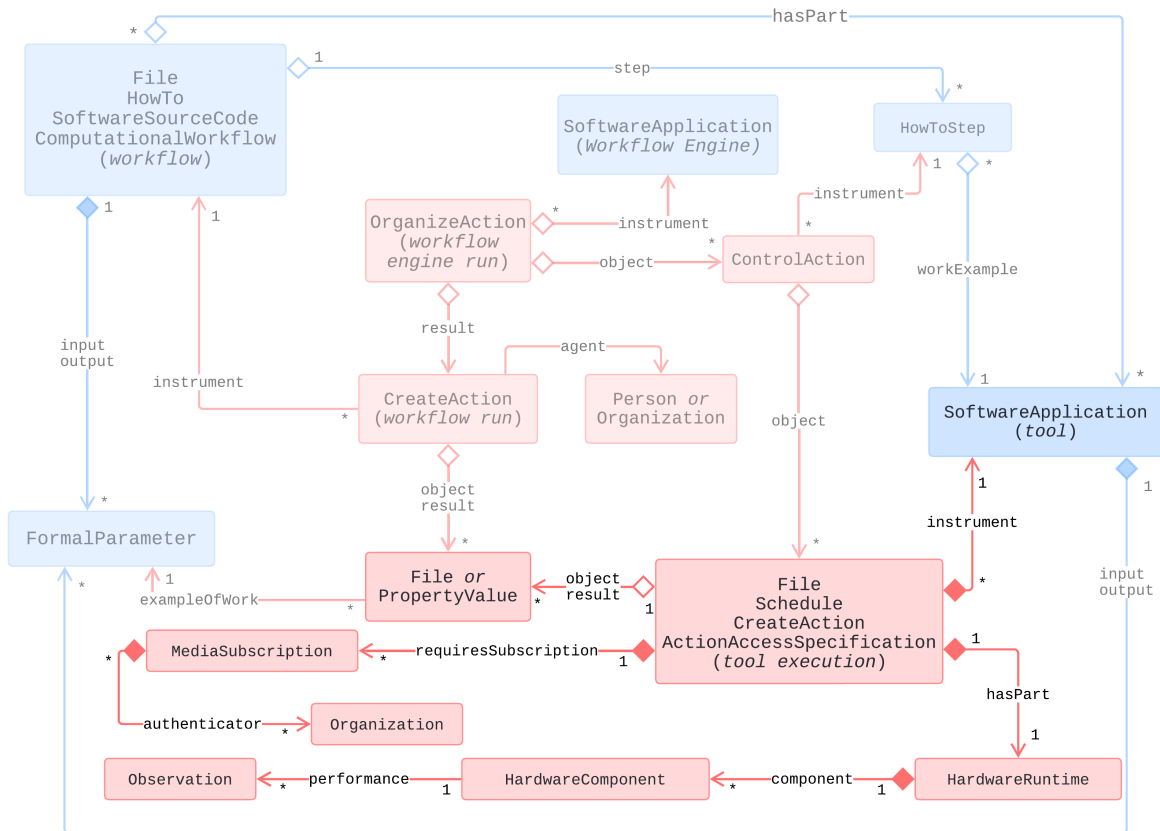


Figure 5. Provenance Run Crate profile with Distributed Step Crate extensions, forming the Distributed Provenance Crate. **Blue:** Prospective Provenance entity graph. **Red:** Retrospective Provenance entity graph.

Listing 1: Minimal RO-Crate manifest file specifying a child RO-Crate.

```

1  {
2  "@context":
3  ↪ "https://w3id.org/ro/crate/1.1/context",
4  "@graph": [
5  {
6    "@type": "CreativeWork",
7    "@id": "ro-crate-metadata.json",
8    "conformsTo": {"@id":
9    ↪ "https://w3id.org/ro/crate/1.1"},
10   "about": {"@id": "."}
11  },
12  {
13    "@id": ".",
14    "@type": "Dataset",
15    "name": "Orchestration Crate",
16    "hasPart": [
17    {
18      "@id":
19      ↪ "child-crate/ro-crate-metadata.json"
20    }
21  ]
22  },
23  ]
24  },

```

```

20  {
21    "@id":
22    ↪ "child-crate/ro-crate-metadata.json",
23    "@type": ["RO-Crate"],
24    "name": "Child Crate",
25    "dateCreated": "2024-04-01"
26  }
27  }

```

Building on this general method for containerizing RO-Crates, the Orchestration Crate is extended to accommodate different research needs and scenarios. These extensions enable the definition of Orchestration Crate profiles; each tailored to the provenance needs of a scientific project:

- **Historical Execution extension:** Recording historical executions of the same workflow, e.g., containing multiple Workflow Run Crates, each documenting a separate instance of the workflow's execution over time.

- **Meta-workflow extension:** Recording heterogeneous computational workflow provenance records for a single project.
- **Hybrid workflow extension:** Recording both computational and non-computational workflow descriptions—widening the scope of provenance recorded during a scientific project.

Case Study: Enabling provenance within Globus Flows

In this section, the Distributed Provenance Crate model is applied within a production-grade WMS, demonstrating its potential to improve the reproducibility and transparency of complex, distributed workflows. To implement it in such an environment, the WMS must support extensibility, particularly within its step execution: while many workflow platforms offer means to implement custom workflow steps, the Distributed Step Crate model requires that provenance is generated *within* each workflow step, as part of its execution, rather than some separate introspection task. To support this behavior, the base class of the executed workflow step must be extensible. Globus Flows meets this requirement and offers a flexible architecture in which orchestrated steps are treated as extensible components of the same class (35, 36).

Globus (37) is a research data management platform designed to connect researchers at different institutions and facilitate large-scale data transfers. Building on its data management foundations, Globus has expanded to include automation services (6) aimed at transitioning traditionally manual research processes to automated and reproducible workflows. These automation services support flows that utilize heterogeneous applications—from scientific instruments to data stores—and accommodate both short- and long-lived processes.

Globus Flows (6) enables users to create, publish, and share workflows composed of a series of actions defined as linked states in an Amazon States Language definition. It implements a flexible authorization model, in which workflow steps are registered as individual OAuth-2 resources, enabling users to delegate authorization in a piece-wise manner.* The security, sharing, and reproducibility of the Globus Flows framework provides a platform for collaborative workflows, particularly beneficial for longitudinal studies where robust and secure data handling is critical.

Extending Globus Flows with Compute

Globus Flows are structured around **Actions**—discrete units of work defined as states in a flow’s Amazon States Language representation. These actions may perform

data management, coordination, or computation tasks and are composable into multi-step workflows. To enable computational actions within a Globus Flow (e.g. performing data transformation rather than data transfer), two classes of compute action may be used. These classes are described in Table 2.

Table 2. Compute Action classes extended with Distributed Step Crate generation.

Compute Action Class	Description
Action Provider	Action Providers are REST API servers that conform to specific behaviors and API interfaces with Globus Services. They are registered centrally and made available to specific user groups for use within workflows. Action Providers allow system administrators to expose predefined, well-optimized tools.
Globus Compute	A Function as a Service (FaaS) action that enables computational steps with Globus Flows. A Globus Compute node executes a user defined Python function, optionally with data, and returns the result. Globus Compute provides a flexible, general-purpose interface for embedding user-defined computation within workflows.

Both classes are extended here with Distributed Step Crate interfaces and represent the two provenance-aware compute classes used to implement the Distributed Provenance Crate within Globus Flows.

Globus Flows with Compute Infrastructure Patterns

The experimental infrastructure for this case study comprises five components. Figure 6 illustrates how these components operate within a provenance-aware Globus Flow. Each component has a distinct role in the management of the workflow:

- **Orchestration Node:** Coordinates and manages the computational workflow and resulting provenance data.
- **Custom Action Provider Node:** An instance of an Action Provider, extended with Distributed Step Crate generation.
- **Globus Compute Node:** An instance of Globus Compute, extended with Distributed Step Crate generation.

*OAuth-2 allows users to grant a third-party application limited access to a service without sharing credentials.

- **Transfer Action Provider Node:** A Transfer Action Provider, responsible for moving data within the infrastructure between nodes.[†]
- **Data Store Node:** Provides persistent storage managed by a Globus Connect server for inputs datasets and output results generated during workflow execution.

In practice, this infrastructure enables Globus Flows to support computational workflows through its Custom Action Provider and Globus Compute class nodes. It is not necessary to use both action classes, i.e., a valid provenance-aware infrastructure may use only one of these action classes.

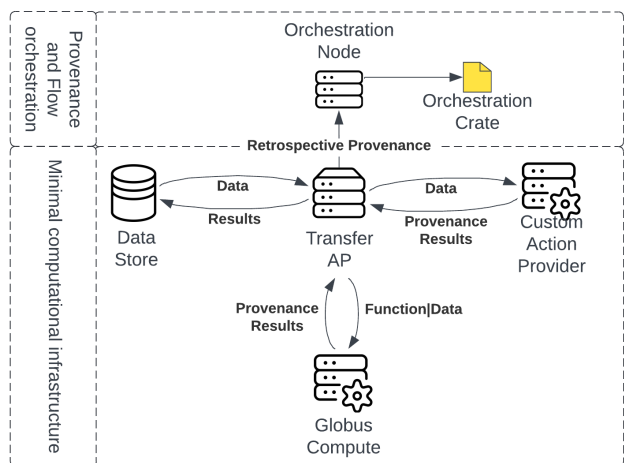


Figure 6. Provenance-Aware Globus Flows experimental infrastructure.

Orchestration Node: Gladier and Provenance

Within the architecture of distributed provenance-enabled WMSs, the Orchestration Node facilitates the execution of workflows and the management of resulting provenance fragments. Within this case study, the Orchestration Node employs Gladier, a Python toolkit, to interface with Globus. Gladier (38) is responsible for generating flow definitions that comprise a sequence of steps (Gladier **tools**) as specified by the user. Once this definition is generated, Gladier handles its registration with Globus, either registering a new flow or updating an existing one within the Globus registry.

For flows that incorporate computational tasks, Gladier also manages the registration of functions to the Globus Compute function registry, retrieving UUIDs that are then specified within the flow’s definition. Once registration is complete, the user provides the necessary inputs to the the flow. Its associated functions, compute actions, and/or Globus Connect servers are authenticated, and the flow is executed.

Gladier Extension To manage provenance within the system during workflow execution and maintain Gladier’s routine use for end users, we developed extensions to the Gladier toolkit, introducing the **ProvenanceClient** and **ProvenanceTool** classes. Typically, when using Gladier, users define **tools** that correspond to each step in their workflow—such as data transfer or computational tasks—and a client that manages flow execution and registration with Globus. The **ProvenanceTool** class allows users to specify computational tools that generate Distributed Step Crates at runtime. These tools are then encapsulated within a *ProvenanceClient*, streamlining the management of provenance fragments at runtime. Listing 2 provides an example client definition.

Listing 2: Example Gladier provenance-aware flow definition.

```

1 class ProvenanceFlow(ProvenanceBaseClient):
2     gladier_tools = [
3         "gladier_tools.globus."
4         ↪ Transfer:ToCompute",
5         data_preprocess,
6         data_postprocess,
7         "gladier_tools.globus."
8         ↪ Transfer:FromCompute"
9     ]

```

When a provenance flow is instantiated using the Globus/Gladier provenance framework, additional transfer steps are integrated into the flow definition. These steps are dynamically linked to their respective provenance-aware computational steps, ensuring that each computational step is automatically paired with a provenance transfer step. There are three primary advantages to integrating the provenance transfer steps into the flow definition:

- **Transparency:** The process of provenance recording, along with related performance details such as transfer timings, is made visible and traceable.
- **Configuration:** Users can easily modify provenance-specific configuration details.
- **Authentication:** Consolidates all OAuth-2 resource authentications into the flow setup phase, avoiding ad-hoc authentication requests during flow execution.

The Gladier Toolkit enabling provenance is available at <https://github.com/LivePublication/live-publication-gladier>. For an example of a

[†]In application, the Transfer AP does not come in contact with transferred data, rather, it facilitates the configuration of the transfer and initiates a direct transfer between nodes. For simplicity, it is represented as an integrated component here.

generated workflow description see the appendix listing [A2](#).

Orchestration Node: Provenance Embeddings The Orchestration Node is responsible for embedding all non-Distributed Step Crate bound provenance entities of the Distributed Provenance Crate.[‡] These entities comprise both prospective and retrospective types.

Prior to workflow execution, the Orchestration Node uses the Gladier-generated flow definition to construct the **prospective** provenance graph. This includes:

- The main entity `File`, `SoftwareSourceCode`, `ComputationalWorkflow`, `HowTo` (workflow), representing the Gladier generated workflow definition.
- `HowToStep` entities for each step within the workflow.
- `SoftwareApplication` entities for each tool referenced within the flow.
- A `SoftwareApplication` entity describing the executing workflow engine—i.e. `Globus`.

The entity graph captures both the structure and the intent of the workflow before it executes. However, due to limitations in Gladier’s abstraction of `tool` interfaces, full modeling of `FormalParameter` entities, used to describe joint inputs and outputs between steps, is currently not supported.

During workflow execution, the **retrospective** provenance graph is populated with runtime metadata such as timestamps associated with component execution and user authentication details (e.g., a user’s Globus UUID). A python toolkit that encodes this information within the Orchestration Node can be found here: https://github.com/LivePublication/LP_SDK.

Distributed Step Crate Generation

The final aspect to enabling the Distributed Provenance Crate within Globus Flows is generating Distributed Step Crates for each compute action class: *Action Provider* and *Globus Compute*.

Action Provider To enable the generation of Distributed Step Crates for Action Providers, customizations are made to a bootstrap library.[§] This library provides Action Provider templates, configured for application within Globus Flows. A custom version of this library is extended by developing a new Python library `lp_ap_tools`. This library defines a Python decorator (`LP_artefact`) that wraps the execution of a method’s Docker container within the Action Provider and is available here: <https://pypi.org/project/lp-ap-tools/0.1.5/>.

Python decorators extend the behavior of a function without altering its core logic. The decorator `LP_artefact` abstracts the provenance encoding processes from the Action Provider template, enabling system administrators to apply provenance tracking with minimal changes to their existing templates.

When a user submits a job to an Action Provider, pre-execution information and metadata are drawn from three data sources: (i) the Action Provider’s metadata, (ii) Globus Services, and (iii) the server’s operating system. This information is used, during the execution of a user’s step, to encode the Distributed Step Crate entities described in Figure 5. The main entity within the Distributed Step Crate is the pre-defined tool (metadata defined by the administrator) which the Action Provider is exposing. The results of the step are assumed to be written to disk and are captured during execution. Globus Services provision access control by evaluating a set of `Globus Scopes` passed to the service at runtime. These scopes are used for modeling `ActionAccessSpecification`, and finally, hardware metrics such as execution time and resource utilization are captured and encoded within the `HardwareRuntime` entity.

Position metadata (i.e. the `HowToStep[position]` field) is determined by passing the Action Provider additional metadata through its POST parameters. Ideally, an Action Provider could query the current task with Globus and retrieve the job UUID, however, the current Action Provider APIs do not provide this capability. The custom action provider template can be found here: https://github.com/LivePublication/LP_GlobusAP_Template.

Globus Compute Enabling Distributed Step Crate generation within Globus Compute is complex due to its distributed, multi-user architecture. Globus Compute operates as a distributed system composed of *workers* and *managers*, designed to handle many users simultaneously (via multi-tenant compute nodes (39)) and distribute tasks across multiple nodes (36). It executes jobs by retrieving the serialized function of a user from the Globus Compute function repository (40) and executes this function on some data.

For application here, Globus Compute is assumed to be configured as a Globus Compute Multi-User Endpoint. In this configuration, the endpoint functions as a process manager: it starts a dedicated compute endpoint agent for each authorized user request. Each of these user-specific endpoints operates independently and handles the execution of user-defined functions retrieved

[‡]That is, the inverse of the entities highlighted within Figure 5.

[§]<https://action-provider-tools.readthedocs.io/en/latest/>

from the Globus Compute Function repository. To enable Distributed Step Crate generation, modifications are applied at the level of these individual user compute endpoints, enabling Distributed Step Crate generation for each function executed per user.

Each worker spawned within the Globus Compute endpoint is extended with an interface that encodes the entities depicted in Figure 3. The main entity is the deserialised script. Results, such as files and property values, are again assumed to be written to disk and collected from the workers containerized environment after execution. The position of the step in the flow is captured through the flow context available to the Globus Compute endpoint and encoded in the *position* attribute of the *HowToStep* entity. Additionally, subscription information necessary for accessing the Compute node is derived from existing metadata configured by a system administrator, including organizational details and required Globus Scopes.

To gather hardware and performance metrics during each job, a script combining Python’s built-in modules and system commands retrieves data such as processor details, memory usage, and operating system specifics. Appendix Table A1 details the tools used to gather hardware details, and Listing 3 provides example CPU and observation entities encoded during Globus Compute endpoint’s execution.

Listing 3: Example CPU HardwareComponent and associated Observation.

```

1  {
2    @id: #da95275e-bd19-463c-9302-9da7a37db2f7,
3    @type: [HardwareComponent, CPU],
4    Architecture: x86_64,
5    CPU(s): 4,
6    Model name: AMD EPYC-Milan Processor,
7    description: CPU properties from lscpu,
8    name: CPU,
9    performance:
10   ↪ #69240cb8-4d5c-4334-97bb-559461c7c8c2
11 }
12 {
13   @id: #69240cb8-4d5c-4334-97bb-559461c7c8c2,
14   @type: Observation,
15   maxValue: {
16     cpu_percent: 31.0,
17     timestamp: 2024-05-20T21:55:18Z
18   },
19   minValue: {
20     cpu_percent: 0.0,
21     timestamp: 2024-05-20T21:55:17Z
22   },
23   measurementTechnique: Psutil.cpu_percent
24   ↪ polled at 0.05,
25   observationAbout:
26   ↪ #da95275e-bd19-463c-9302-9da7a37db2f7,

```

```

24   image: images/cpu_usage.png
25 }

```

The custom Globus Compute template can be found here: https://github.com/LivePublication/live-publication-globus-compute/tree/dist_step_crate_gen.

Provenance-Aware Globus Flows Evaluation

In this section, the application of the Distributed Provenance Crate to Globus Flows is evaluated with respect to its semantic correctness.

This implementation is evaluated through a worked example. The objective is to demonstrate that a provenance record generated from a distributed Globus Flow execution can embed the same logical structure and semantic relationships as one generated from a centralized CWL workflow.

The comparison Provenance Run Crate (derived from the Provenance Run Crate specification (12)) describes the execution of a simple, centralized, two-step CWL workflow. Executing this workflow runs its computational steps and, as a byproduct, produces the comparative **Provenance Run Crate**, which serves here as a validation target for the provenance records generated via Globus Flows.[¶]

This reference workflow is re-implemented in Globus Flows with a Globus Compute-based distributed infrastructure. The resulting provenance records are encoded using the Distributed Step Crate extensions and embedded into a single Distributed Provenance Crate for comparison. The resulting provenance record, and its constituent Distributed Step Crates, are hosted as a live, interactive RO-Crate here: <https://gusellerm.github.io/distributed-provenance-example/> (Figure 7).

The distributed provenance record is validated using a simple custom validator implemented within the **LP_SDK** library. This tool performs structural and semantic validation against the provenance model. The two records are conceptually equivalent: both encode the same workflow provenance information describing the execution of the workflow as a whole (as specified by the Workflow Run RO-Crate specification) and both enrich this model with step-wise provenance. Within the Globus Flows record, each step is described using a Distributed Step Crate, capturing equivalent execution semantics while

[¶]Find an interactive format of the reference Provenance Run crate here: https://www.researchobject.org/workflow-run-crate/profiles/provenance_run_crate/example3/ro-crate-preview.html.

Example Provenance-Aware Globus Flows execution

Download all the metadata for Example Provenance-Aware Globus Flows execution in JSON-LD format
[Check this crate](#)

Example Provenance-Aware Globus Flows execution

@id	/
name [?]	Example Provenance-Aware Globus Flows execution
@type	Dataset
description [?]	This RO-Crate contains a Globus Flows workflow execution that demonstrates the use of RO-Crate for provenance tracking in computational workflows.
datePublished [?]	2025-06-29T04:08:06+00:00
license [?]	https://creativecommons.org/licenses/by/4.0/
conformsTo [?]	<ul style="list-style-type: none"> Process Run Crate Workflow Run Crate Provenance Run Crate Workflow RO-Crate Distributed Provenance Crate
hasPart [?]	<ul style="list-style-type: none"> WEP.json Workflow Execution Description Workflow Input Description CPU usage RevText input RevText output Distributed Step Crate CPU usage sort_text input sort_text output Distributed Step Crate
mainEntity [?]	WEP.json
Items that reference this one	
about [?]	ro-crate-metadata.json

Figure 7. Distributed Provenance Crate generated with a Provenance-Aware Globus Flow workflow. An interactive representation is available here: <https://guselerm.github.io/distributed-provenance-example>.

incorporating additional information (i.e., performance and access control metadata). Additional information on this validation is available here: <https://guselerm.github.io/distributed-provenance-example/>.

This example demonstrates the efficacy of the provenance model presented within a distributed production WMS. Although Globus Flows workflow execution differs architecturally from the CWL example, the resulting provenance information is conceptually analogous while also extending the model to accommodate distributed execution, hardware context, and runtime metadata.

Discussion

Our evaluation shows that Distributed Step Crates compiled into a Distributed Provenance Crate can reproduce the step-level semantics of a centralized Provenance Run Crate while enriching this information with runtime environment details (hardware/runtime/auth). This shifts provenance capture closer to the source of compute, which is necessary in federated settings where intermediate states and performance context are otherwise lost. The contiguity of the provenance record is achieved under three preconditions: (1) the availability

of a global step identity at runtime, (2) recoverable input–output bindings between steps, and (3) the number of Distributed Step cage generating nodes within the workflow. Next, we discuss where these assumptions hold, where they fail, and what engineering patterns enable these conditions to be met within distributed WMSs.

The Distributed Provenance Crate schema described in this paper requires access to a global context of step UUIDs, used to enable the mapping between `ControlAction` entities and their associated Distributed Step Crates (Figure 4). Access to this context is not guaranteed; while all distributed WMSs govern the execution of steps across infrastructure, they do not necessarily make available the internal context and identities assigned to each step. Take, for example, the custom Action Provider developed as part of the Globus Flows integration. These servers lack native support for accessing the global workflow context during runtime, making it difficult to assign the `HowToStep[identifier]` field. This was addressed by passing additional position metadata to the Action Provider at runtime (see [Distributed Step Crate Generation](#)). When instrumenting the Distributed Provenance Crate within a WMS, accessible execution context, available at runtime within each provenance-generating node, simplifies Distributed Step Crate encoding, and reduces custom instrumentation.

Complete provenance records are dependent on the representation of *dataflow edges*. Dataflow edges represent the explicit input/output relationships between workflow steps—not only step position information. The Provenance Run Crate specification describes these bindings implicitly through the `File or Property-Value` entities *exampleOfWork* property which aggregates like semantic outputs and inputs (i.e., `FormalParameters`), identifying them as the same conceptual data. These relationships can be made explicit using the `ParameterConnection` entity that relates an output of a step with the input of another. As an extension, the Distributed Provenance Crate similarly provides capabilities for representing dataflow edges. However, within distributed environments, the edges between steps are difficult to identify as multiple versions of the “same” data are created, transferred, and referenced under different names and locals.

Distributed WMSs routinely select between these multiple existing versions of the same data to optimize transfer times, stage data into and out of execution nodes/sites, and register produced files in catalogs or object stores with assigned UUIDs.^{||} This complexity makes the inference of conceptual identity more

^{||}E.g., Pegasus’s [replica catalogues](#) and Arvados’s [Keep storage](#).

difficult—path equality is no longer a reliable indicator of identity.

Gladier generated Globus Flow workflow definitions do not describe input-output bindings between steps. Each step executes on isolated endpoints, and transfer actions move files across stages, which creates ambiguity in their conceptual identities. In practice, this means that input/output edges cannot be reliably reconstructed given a set of Distributed Step Crates and their corresponding Gladier-generated workflow description. Take, in contrast, CWL’s explicit `in` and `out` fields which map a step’s conceptual outputs with inputs (see Appendix Listing A1). Generally, this is a recurring challenge for provenance in distributed environments. Contiguous and complete records depend upon *conceptual* data-flow descriptions, which are not necessary for the operation of these platforms, and are therefore often absent from their native workflow descriptions. Distributed WMSs prioritize reliable governance and execution of distributed tasks; Unless the author or an adapter provides additional edge metadata, gaps will remain within the provenance chain.

The completeness of provenance within a Distributed Provenance Crate depends on the number of orchestrated steps that produce Distributed Step Crates. Here, we distinguish between extensible steps (those which can be instrumented with Distributed Step Crates) and non-extensible steps (e.g. external services called steps within a workflow). Within our example implementation, *compute provider* nodes are extensible steps, while *transfer actions* are not. This heterogeneity results in a mixture of fine- and coarse-grained provenance. Within distributed environments, it is unlikely that the WMS will govern every service orchestrated within the execution of some workflow. One of the strengths of distributed workflows *is* the ability to integrate third-party services and manage the execution of resources owned by collaborators (e.g., partner laboratories, national facilities or independent cloud resources) (41). Therefore, it is not expected or required that every step within a distributed workflow be provenance-generating, or generate the same provenance granularity.**

The Distributed Provenance Crate accommodates this provenance heterogeneity by discretizing provenance into per-step Distributed Provenance Crates. This provides two benefits; First, Distributed Step Crates can be specialized as a small family of profiles for different service classes. This allows each node to “report what it can”. Second, because Distributed Step Crates records only retrospective traces, the prospective representations of these steps (e.g., software, planned inputs/outputs) remains uniform across both extensible and non-extensible steps. The result is an opt-in provenance model: instrumented steps report detailed

retrospective records, non-instrumented steps contribute prospective metadata, and the aggregate still supports end-to-end lineage even as granularity varies by step.

Globus Flows in its current state lacks built-in parallel workflow management behaviors, which results in blocking provenance transfer tasks during execution. As the number of orchestrated steps increases, the time to transfer provenance grows linearly. Additionally, each transfer incurs overhead from the WMSs coordination methods (e.g., exponential back-off, polling mechanisms, and communication delay). Trivial “no-op” actions within Globus Flows have been shown to incur non-trivial overhead (an average of 2.88 seconds) (6). For these reasons, measuring the overheads of the Distributed Provenance Crate method using the developed Globus Flows implementation does not provide useful feedback on the general overheads of the provenance method itself.

However, per-step Distributed Step Crate generation will introduce notable overhead in parallel/HPC workflows driven by workflow granularity, I/O and data-transfer costs, and WMS orchestration methods. Minimizing this overhead may involve optimizing crate generation and supporting multi-step Distributed Step Crates when several steps run on the same node during the same workflow. This would likely require extending the Distributed Step Crate schema with a `ControlAction` entity, and providing nodes with sufficient global workflow context—something many distributed WMSs do not expose without additional instrumentation.

Based on the analysis and experience of applying the Distributed Provenance Crate to a production distributed WMS, we identify a set of four practical requirements for WMS compatibility:

- **Extensible step classes (or execution wrapper):** The WMS should expose a shared, instrumentable step type/wrapper such that Distributed Step Crate interface can be defined once and applied to *all* steps of that class.
- **Accessible workflow context at runtime:** Assuming a centralized WMS service^{††}, provide an external API to read stateful workflow context during execution from *within* workflow step environments. This includes instantiated step UUIDs and context information such as attempt number.

**This also applies to contexts with ethics restrictions and privacy concerns.

^{††}Most distributed WMSs rely on a central service to manage workflow orchestration. Decentralized orchestration models are out of scope, but usually rely on context propagation between each decentralized step.

- **Explicit dataflow edges:** Model input/output bindings in the workflow description to enable mapping `FormalParameter` (and optional `ParameterConnection`); where edges are absent or ambiguous, provide a logical/content identifier (e.g., content hash, dataset accession, or store-level UUID) with each `File` alongside its URI to preserve conceptual identity across stages.
- **Step environment recording:** Lightweight capture of hardware/runtime metrics and access conditions to populate Distributed Step Crate extensions.

Having analyzed the constraints and trade-offs of our Globus Flows implementation, we now step back to reflect on the process of extending community standards, and the re-execution, reuse, and interoperability afforded by the Distributed Provenance Crate.

Reflections on Extending a Community Standard

Extending the Provenance Run Crate was straightforward due to composable entities and JSON-LD vocabularies. The design of the Distributed Step Crate did not require the development of new formats or break compatibility with the existing standardized RO-Crate tooling. Using community standards, the Distributed Provenance Crate is more readily available for reuse and application.

Generalized schemas, such as the Provenance Run Crate, embed assumptions regarding the “form” of their subject. For example, entities such as the `ControlAction` entity which model complex step management behaviors (e.g. parallel execution), are not applicable to Globus Flows’ workflow descriptions, but are assumed within the schema. When developing a general schema, there is an inherent risk of over-specification, and in doing so becoming difficult to apply to non-conformant systems. The flexibility of the model to represent diverse systems and advice regarding the modification of the model are important factors when approaching the application of these schemas to diverse WMSs.

Our vocabulary work followed standard Semantic Web practice: reuse first (RO-Crate and schema.org), and mint sparingly (a small set of new terms for hardware and runtime). Publishing and validating new terms remains awkward and developer centric. Lightweight tooling—a lightweight profile/vocabulary authoring workflow producing versioned JSON-LD contexts and SHACL/JSON schemas—would reduce friction for non-technical users to both identify terms for reuse and develop specialized vocabularies for their projects.

Conclusion

In this paper, we address the growing challenge of provenance recording within distributed and federated WMSs. As our eScience infrastructure matures and we want to maintain strong links between scientific instruments, computation, and publication, support for workflows that leverage these distributed resources to self-report in the form of provenance records becomes increasingly important. The role of WMSs as managers of complex scientific computational workflows must include introspection and transparency mechanisms that allow researchers to query and share representations of their computational processes with colleagues.

We have shown how provenance models, such as Workflow Run Crate (12), can be modified for specific workflow execution environments. We introduced a new RO-Crate specification, the Distributed Step Crate, designed to facilitate the recording of provenance information at the source of compute. Additionally, we extend the Provenance Run Crate to create the Distributed Provenance Crate, integrating this information into a holistic provenance record. We also proposed the Orchestration Crate, a data model capable of containing multiple RO-Crates, which extends provenance recording from a single workflow execution to multiple workflow executions.

Finally, we applied this theoretical provenance model to the Globus Flows (6) framework through a novel integration with Globus Compute, Globus Action Providers (36) and the Gladier SDK (38). These integration’s enable the production of Distributed Step Crates at the source of compute and the creation of the Orchestration Crate. Users can execute provenance-aware flows using *ProvenanceTool* and *ProvenanceClient* extensions in Gladier, lowering the barrier of entry to generate comprehensive provenance records.

Future research will focus on:

- **FormalParameter modeling:** Modifying Gladier to explicitly model inputs/outputs and their relationships as *FormalParameters* between steps.
- **Non-blocking provenance transfers:** Extending the developed provenance-aware Globus Flows application with non-blocking provenance transfers.
- **Refinement of Distributed Step Crate:** Developing specializations of the Distributed Step Crate for both different classes of workflow steps and optimization techniques (i.e., batching of provenance within a single node).
- **Orchestration Crate profiles:** Development and application of Orchestration Crate patterns for use as meta-provenance containers.

We also plan to investigate the integration of this provenance model within frameworks for live and dynamic scientific publishing ([42](#)), and the application of the Orchestration Crate as an interface for orchestrating collections of workflows across multiple WMSs.

References

1. Belhajjame, K., R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes. PROV-DM: The PROV data model. *W3C*.
2. Amstutz, P., M. Mikheev, M. R. Crusoe, N. Tijanić, S. Lampa, and Others. Existing workflow systems. Common workflow language Wiki, GitHub. <https://s.apache.org/existing-workflow-systems>, 2024. Accessed: 2024-4-15.
3. Deelman, E., K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger. Pegasus, a workflow management system for science automation. *Future Gener. Comput. Syst.*, Vol. 46, 2015, pp. 17–35.
4. Di Tommaso, P., M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame. Nextflow enables reproducible computational workflows. *Nat. Biotechnol.*, Vol. 35, No. 4, 2017, pp. 316–319.
5. Huber, S. P., S. Zoupanos, M. Uhrin, L. Talirz, L. Kahle, R. Häuselmann, D. Gresch, T. Müller, A. V. Yakutovich, C. W. Andersen, F. F. Ramirez, C. S. Adorf, F. Gargiulo, S. Kumbhar, E. Passaro, C. Johnston, A. Merksys, A. Cepellotti, N. Mounet, N. Marzari, B. Kozinsky, and G. Pizzi. AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance. *Sci Data*, Vol. 7, No. 1, 2020, p. 300.
6. Chard, R., J. Pruyne, K. McKee, J. Bryan, B. Raumann, R. Ananthakrishnan, K. Chard, and I. T. Foster. Globus automation services: Research process automation across the space–time continuum. *Future Gener. Comput. Syst.*, Vol. 142, 2023, pp. 393–409.
7. Vivian, J., A. A. Rao, F. A. Nothaft, C. Ketchum, J. Armstrong, A. Novak, J. Pfeil, J. Narkizian, A. D. Deran, A. Musselman-Brown, H. Schmidt, P. Amstutz, B. Craft, M. Goldman, K. Rosenbloom, M. Cline, B. O'Connor, M. Hanna, C. Birger, W. J. Kent, D. A. Patterson, A. D. Joseph, J. Zhu, S. Zaraneke, G. Getz, D. Haussler, and B. Paten. Toil enables reproducible, open source, big biomedical data analyses. *Nat. Biotechnol.*, Vol. 35, No. 4, 2017, pp. 314–316.
8. Arvados homepage. <https://arvados.org/>. Accessed: 2025-6-22.
9. Costa, F., D. De Oliveira, and M. Mattoso. Towards an Adaptive and Distributed Architecture for Managing Workflow Provenance Data. In *2014 IEEE 10th International Conference on e-Science*, Vol. 2. IEEE, 2014, pp. 79–82.
10. Williams, A. and D. K. Tosh. Scientific Workflow Provenance Architecture for Heterogeneous HPC Environments. In *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2021, pp. 0921–0927.
11. Soiland-Reyes, S., P. Sefton, M. Crosas, L. J. Castro, F. Coppens, J. M. Fernández, D. Garijo, B. Grüning, M. La Rosa, S. Leo, E. Ó Carragáin, M. Portier, A. Trisovic, RO-Crate Community, P. Groth, and C. Goble. Packaging research artefacts with RO-Crate. *Data sci.*, Vol. 5, No. 2, 2022, pp. 97–138.
12. Leo, S., M. R. Crusoe, L. Rodríguez-Navas, R. Sirvent, A. Kanitz, P. De Geest, R. Wittner, L. Pireddu, D. Garijo, J. M. Fernández, I. Colonnelli, M. Gallo, T. Ohta, H. Suetake, S. Capella-Gutierrez, R. de Wit, B. P. Kinoshita, and S. Soiland-Reyes. Recording provenance of workflow runs with RO-Crate. *PLoS One*, Vol. 19, No. 9, 2024, p. e0309210.
13. Workflow Run RO-Crate working group. Provenance Run Crate. <https://w3id.org/ro/wfrun/provenance/0.5>. Accessed: 2024-4-27.
14. Research Object Community. Provenance run Crate. https://www.researchobject.org/workflow-run-crate/profiles/provenance_run_crate/. Accessed: 2025-10-13.
15. Goble, C., S. Cohen-Boulakia, S. Soiland-Reyes, D. Garijo, Y. Gil, M. R. Crusoe, K. Peters, and D. Schober. FAIR Computational Workflows. *Data Intelligence*, Vol. 2, No. 1-2, 2020, pp. 108–121.
16. Freire, J., D. Koop, E. Santos, and C. T. Silva. Provenance for Computational Tasks: A Survey. *Comput. Sci. Eng.*, Vol. 10, No. 3, 2008, pp. 11–21.
17. Davidson, S., S. Cohen-Boulakia, A. Eyal, B. Ludäscher, T. McPhillips, S. Bowers, M. Anand, and J. Freire. Provenance in Scientific Workflow Systems. *IEEE Data Eng. Bull.*, Vol. 30, 2007, pp. 44–50.
18. Costa, F., D. De Oliveira, and M. Mattoso. Towards an Adaptive and Distributed Architecture for Managing Workflow Provenance Data. In *2014 IEEE 10th International Conference on e-Science*, Vol. 2. IEEE, 2014, pp. 79–82.
19. Goecks, J., A. Nekrutenko, J. Taylor, and Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.*, Vol. 11, No. 8, 2010, p. R86.
20. Oinn, T., M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, Vol. 20, No. 17, 2004, pp. 3045–3054.
21. Callahan, S. P., J. Freire, E. Santos, and V. T. Huy. VisTrails: Visualization meets data management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*. unknown, 2006, pp. 745–747.
22. Silva, R. F. d., L. Pottier, T. Coleman, E. Deelman, and H. Casanova. WorkflowHub: Community framework for enabling scientific workflow research and development.

- In *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2020.
23. Haines, S. Workflow Orchestration with Apache Airflow. In *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications* (S. Haines, ed.). Apress, Berkeley, CA, 2022, pp. 255–295.
 24. Chapman, B., J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic. *Common Workflow Language, v1.0*. Tech. rep., Common Workflow Language working group, 2016.
 25. Souza, R., L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Vital Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. S. Netto. Workflow provenance in the lifecycle of scientific machine learning. *Concurr. Comput.*, Vol. 34, No. 14.
 26. Butt, A. S. and P. Fitch. ProvONE+: A Provenance Model for Scientific Workflows. In *Web Information Systems Engineering – WISE 2020*. Springer International Publishing, 2020, pp. 431–444.
 27. International Organization for Standardization. Provenance information model for biological material and data, 2023.
 28. Khan, F. Z., S. Soiland-Reyes, and others. CWLProv-Interoperable Retrospective Provenance capture and its challenges. *International*.
 29. Samuel, S. and B. König-Ries. REPRODUCE-ME: Ontology-Based Data Access for Reproducibility of Microscopy Experiments. In *The Semantic Web: ESWC 2017 Satellite Events*. Springer International Publishing, 2017, pp. 17–20.
 30. Guha, R. V., D. Brickley, and S. Macbeth. Schema.org: evolution of structured data on the web. *Commun. ACM*, Vol. 59, No. 2, 2016, pp. 44–51.
 31. Şimşek, U., K. Angele, E. Kärle, O. Panasiuk, and D. Fensel. Domain-Specific Customization of Schema.org Based on SHACL. In *The Semantic Web – ISWC 2020* (J. Z. Pan, V. Tamma, C. d’Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, and L. Kagal, eds.). Springer International Publishing, Cham, 2020, pp. 585–600.
 32. Truong, H.-L., P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak, and K. Rozkwitalski. K-WfGrid distributed monitoring and performance analysis services for workflows in the grid. In *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science’06)*. IEEE, 2006.
 33. Gu, Y. and C. Q. Wu. Performance analysis and optimization of distributed workflows in heterogeneous network environments. *IEEE Trans. Comput.*, Vol. 65, No. 4, 2016, pp. 1266–1282.
 34. Glatard, T., M.-E. Rousseau, S. Camarasu-Pop, R. Adalat, N. Beck, S. Das, R. F. da Silva, N. Khalili-Mahani, V. Korkhov, P.-O. Quirion, P. Rioux, S. D. Olabarriaga, P. Bellec, and A. C. Evans. Software architectures to integrate workflow engines in science gateways. *Future Gener. Comput. Syst.*, Vol. 75, 2017, pp. 239–255.
 35. Globus. Action Provider Tools: Tools to help developers build Action Providers for use in Globus Flows. <https://github.com/globus/action-provider-tools>. Accessed: 2025-7-1.
 36. Chard, R., Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard. funcX: A Federated Function Serving Fabric for Science. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing. HPDC ’20*, Association for Computing Machinery, New York, NY, USA, 2020, pp. 65–76.
 37. Globus. <https://www.globus.org/>. Accessed: 2025-7-1.
 38. Vescovi, R., R. Chard, N. D. Saint, B. Blaiszik, J. Pruyne, T. Bicer, A. Lavens, Z. Liu, M. E. Papka, S. Narayanan, N. Schwarz, K. Chard, and I. T. Foster. Linking scientific instruments and computation: Patterns, technologies, and experiences. *Patterns (N Y)*, Vol. 3, No. 10, 2022, p. 100606.
 39. Ananthakrishnan, R., Y. Babuji, M. Baughman, J. Bryan, K. Chard, R. Chard, B. Clifford, I. Foster, D. S. Katz, K. Hunter Kesling, C. Janidlo, R. Mello, and L. Wang. Enabling remote management of FaaS endpoints with globus compute multi-user endpoints. In *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*. ACM, New York, NY, USA, 2024.
 40. Bauer, A., H. Pan, R. Chard, Y. Babuji, J. Bryan, D. Tiwari, I. Foster, and K. Chard. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. *Future Gener. Comput. Syst.*, Vol. 153, 2024, pp. 558–574.
 41. Wolstencroft, K., R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res.*, Vol. 41, No. Web Server issue, 2013, pp. W557–61.
 42. Ellerm, A., M. Gahegan, and B. Adams. LivePublication: The Science Workflow Creates and Updates the Publication. In *2023 IEEE 19th International Conference on e-Science (e-Science)*. IEEE, 2023, pp. 1–10.

Appendix A Supplementary Material

Table A1. A list of functions and tools used to compile the hardware and environment details of compute providers.

Hardware Scraping Tool	Description
Platform.processor	CPU type and architecture
Platform.node	Systems hostname
Platform.platform	Operating system and version information
Os.cp_count	Number of CPU cores available
Plutil.virtual_memory()	System memory usage and availability
Lscpu	CPU architecture and capabilities
Lshw-C display	Display and Graphics Card
Nvidia-smi	NVIDIA GPU information
Sys.version	Version of python installed in host
Df	Available disk space on mounts

Listing A1: Example CWL workflow step definition illustrating how inputs and outputs are explicitly mapped between steps.

```

1  {
2    "in": [
3      {
4        "source": "#main/input",
5        "id": "#main/rev/input"
6      }
7    ],
8    "out": [
9      "#main/rev/output"
10   ],
11   "run": "#revtool.cwl",
12   "id": "#main/rev"
13 },
14 {
15   "in": [
16     {
17       "source": "#main/rev/output",
18       "id": "#main/sorted/input"
19     },
20     {
21       "source": "#main/reverse_sort",
22       "id": "#main/sorted/reverse"
23     }
24   ],
25   "out": [
26     "#main/sorted/output"
27   ],
28   "run": "#sorttool.cwl",
29   "id": "#main/sorted"
30 }

```

Listing A2: Example provenance-aware, Gladier-generated workflow description.

```

1  {
2    "States": {
3      "TransferToCompute": {
4        "Comment": "Transfer a file or directory in Globus",

```

```

5     "Type": "Action",
6     "ActionUrl": "https://transfer.actions.globus.org/ transfer/",
7     "Parameters": {
8         "source_endpoint.$": "$.input.to_compute_transfer_source_endpoint_id",
9         "destination_endpoint.$": "$.input.to_compute_transfer_destination_endpoint_id",
10        "transfer_items": [
11            {
12                "source_path.$": "$.input.to_compute_transfer_source_path",
13                "destination_path.$": "$.input.to_compute_transfer_destination_path",
14                "recursive.$": "$.input.to_compute_transfer_recursive"
15            }
16        ]
17    },
18    "ResultPath": "$.Transfer",
19    "WaitTime": 600,
20    "Next": "RevTxt"
21},
22"RevTxt": {
23    "Comment": null,
24    "Type": "Action",
25    "ActionUrl": "https://compute.actions.globus.org",
26    "ExceptionOnActionFailure": false,
27    "Parameters": {
28        "tasks": [
29            {
30                "endpoint.$": "$.input.compute_endpoint",
31                "function.$": "$.input.rev_txt_function_id",
32                "payload.$": "$.input.RevTxt"
33            }
34        ]
35    },
36    "ResultPath": "$.RevTxt",
37    "WaitTime": 300,
38    "Next": "Transfer_provenance_rev_txt"
39},
40"Transfer_provenance_rev_txt": {
41    "Comment": "Transfer a file or directory in Globus",
42    "Type": "Action",
43    "ActionUrl": "https://transfer.actions.globus.org/ transfer/",
44    "Parameters": {
45        "source_endpoint.$": "$.input.prov_compute_GCS_id",
46        "destination_endpoint.$": "$.input.orchestration_server_endpoint_id",
47        "transfer_items": [
48            {
49                "recursive": true,
50                "source_path.=": "`$.RevTxt.details.results[0].task_id` + '.crate'",
51                "destination_path.=": "`$.input._provenance_crate_destination_directory` + '/' +
52                ↪ `$.RevTxt.details.results[0].task_id`"
53            }
54        ]
55    },
56    "ResultPath": "$.Transfer_provenance_rev_txt",
57    "WaitTime": 600,
58    "Next": "TransferRT_ST"
59},
60"TransferRT_ST": {
61    "Comment": "Transfer a file or directory in Globus",
62    "Type": "Action",
63    "ActionUrl": "https://transfer.actions.globus.org/ transfer/",
64    "Parameters": {
65        "source_endpoint.$": "$.input.rt_st_transfer_source_endpoint_id",
66        "destination_endpoint.$": "$.input.rt_st_transfer_destination_endpoint_id",
67        "transfer_items": [

```

```

67     {
68         "source_path.$": "$.input.rt_st_transfer_source_path",
69         "destination_path.$": "$.input.rt_st_transfer_destination_path",
70         "recursive.$": "$.input.rt_st_transfer_recursive"
71     }
72 ]
73 },
74 "ResultPath": "$.Transfer",
75 "WaitTime": 600,
76 "Next": "SortTxt"
77 },
78 "SortTxt": {
79     "Comment": null,
80     "Type": "Action",
81     "ActionUrl": "https://compute.actions.globus.org",
82     "ExceptionOnActionFailure": false,
83     "Parameters": {
84         "tasks": [
85             {
86                 "endpoint.$": "$.input.compute_endpoint",
87                 "function.$": "$.input.sort_txt_function_id",
88                 "payload.$": "$.input.SortTxt"
89             }
90         ]
91     },
92     "ResultPath": "$.SortTxt",
93     "WaitTime": 300,
94     "Next": "Transfer_provenance_sort_txt"
95 },
96 "Transfer_provenance_sort_txt": {
97     "Comment": "Transfer a file or directory in Globus",
98     "Type": "Action",
99     "ActionUrl": "https://transfer.actions.globus.org/ transfer/",
100    "Parameters": {
101        "source_endpoint.$": "$.input.prov_compute_GCS_id",
102        "destination_endpoint.$": "$.input.orchestration_server_endpoint_id",
103        "transfer_items": [
104            {
105                "recursive": true,
106                "source_path.=": "`$.SortTxt.details.results[0].task_id` + '.crate'",
107                "destination_path.=": "`$.input._provenance_crate_destination_directory` + '/' +
↳ `$.SortTxt.details.results[0].task_id`"
108            }
109        ]
110    },
111    "ResultPath": "$.Transfer_provenance_sort_txt",
112    "WaitTime": 600,
113    "Next": "TransferFromCompute"
114 },
115 "TransferFromCompute": {
116     "Comment": "Transfer a file or directory in Globus",
117     "Type": "Action",
118     "ActionUrl": "https://transfer.actions.globus.org/ transfer/",
119     "Parameters": {
120         "source_endpoint.$": "$.input.from_compute_transfer_source_endpoint_id",
121         "destination_endpoint.$": "$.input.from_compute_transfer_destination_endpoint_id",
122         "transfer_items": [
123             {
124                 "source_path.$": "$.input.from_compute_transfer_source_path",
125                 "destination_path.$": "$.input.from_compute_transfer_destination_path",
126                 "recursive.$": "$.input.from_compute_transfer_recursive"
127             }
128         ]

```

```
129     },
130     "ResultPath": "$.Transfer",
131     "WaitTime": 600,
132     "End": true
133   }
134 },
135 "Comment": "Workflow matching the provenance rocrate example",
136 "StartAt": "TransferToCompute"
137 }
```