A Terminology for Scientific Workflow Systems

Frédéric Suter[a,*], Tainã Coleman[b,*], İlkay Altintaş[b], Rosa M. Badia[c], Bartosz Balis[d], Kyle Chard[e], Iacopo Colonnelli[f], Ewa Deelman[g], Paolo Di Tommaso[h], Thomas Fahringer[i], Carole Goble[j], Shantenu Jha[k], Daniel S. Katz[l], Johannes Köster[m], Ulf Leser[n], Kshitij Mehta[a], Hilary Oliver[o], J.-Luc Peterson[p], Giovanni Pizzi[q], Loïc Pottier[p], Raül Sirvent[c], Eric Suchyta[a], Douglas Thain[r], Sean R. Wilkinson[a], Justin M. Wozniak[s], Rafael Ferreira da Silva[a]

[a] *Oak Ridge National Laboratory, TN, USA*
[b] *University of California, San Diego, CA, USA*
[c] *Barcelona Supercomputing Center, Barcelona, Spain*
[d] *AGH University of Krakow, Krakow, Poland*
[e] *University of Chicago, Chicago, IL, USA*
[f] *University of Torino, Torino, Italy*
[g] *Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA*
[h] *Seqera Labs, Barcelona, Spain*
[i] *University of Innsbruck, Institute of Computer Science, Innsbruck, Austria*
[j] *University of Manchester, Manchester, United Kingdom*
[k] *Rutgers University-New Brunswick; Princeton Plasma Physics Laboratory; Princeton University, NJ, USA*
[l] *NCSA & School of Computing and Data Science & iSchool, University of Illinois Urbana-Champaign, IL, USA*
[m] *University of Duisburg-Essen, Essen, Germany*
[n] *Institute for Computer Science, Humboldt-Universität zu Berlin, Berlin, Germany*
[o] *National Institute of Water and Atmospheric Research, Wellington, New Zealand*
[p] *Lawrence Livermore National Laboratory, Livermore, CA, USA*
[q] *PSI Center for Scientific Computing, Theory and Data, Villigen, Switzerland*
[r] *University of Notre Dame, Notre Dame, IN, USA*
[s] *Argonne National Laboratory, Lemont, IL, USA*

Abstract

The term "scientific workflow" has evolved over the last two decades to encompass a broad range of compositions of interdependent compute tasks and data movements. It has also become an umbrella term for processing in modern scientific applications. Today, many scientific applications can be considered as workflows made of multiple dependent steps, and hundreds of workflow systems have been developed to manage and run these scientific workflows. However, no turnkey solution has emerged from the field to address the diversity of scientific processes and the infrastructure on which they are supposed to be implemented. Instead, new research problems requiring the execution of scientific workflows with some novel feature often lead to the development of an entirely new workflow system. A direct consequence of this situation is that many existing workflow management systems (WMSs) share some salient features, offer similar functionalities, and can manage the same categories of workflows but at the same time also have some distinct capabilities that can be important for specific applications. This situation makes researchers who develop workflows face the complex question of selecting a WMS. This selection can be driven by technical considerations, to find the system that is the most appropriate for their application and for the computing and storage resources available to them, or other factors such as reputation, adoption, strong community support, or long-term sustainability. To address this problem, a group of WMS developers and practitioners joined their efforts to produce a community-based terminology of WMSs. This paper summarizes their findings and introduces this new terminology to characterize WMSs. This terminology is composed of fives axes: workflow structure and characteristics, composition, orchestration, data management, and metadata capture. Each axis comprises several concepts that capture the prominent features of WMSs. Based on this terminology, this paper also presents a classification of 23 existing WMSs according to the proposed axes and terms.

Keywords: Scientific workflows, workflow management systems, community-based terminology

⋆This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doepublic-access-plan).

*Corresponding authors
Email addresses: suterf@ornl.gov (Frédéric Suter), t1coleman@ucsd.edu (Tainã Coleman), ialtintas@ucsd.edu (İlkay Altintaş), rosa.m.badia@bsc.es (Rosa M. Badia), balis@agh.edu.pl (Bartosz Balis), chard@uchicago.edu (Kyle Chard),

1. Introduction

The concept of *workflows*, i.e., the execution of orchestrated and repeatable patterns of activity, dates back to the early 1900s when the engineering and manufacturing community introduced one of the earliest examples of procedural workflow: the Ford assembly line adopted by automobile manufacturers to this date. Workflows has been used to model, analyze, and improve business processes, using tools such as flow charts, functional flow block diagrams, or control flow diagrams [1]. The database community has also used workflows to address the challenges of managing large datasets [2]. The capacity to describe and orchestrate such complex applications popularized workflows across multiple scientific domains. The term *scientific workflow* itself was introduced in 1996 [3, 4] to differentiate this specific type of workflow from the business and automation pipelines that inspired them. As scientific workflow may designate processes that go beyond science to cover more broadly defined research activities, we opted for the use of the term *workflow* in the remainder of this article with the following all-encompassing definition:

Definition. A workflow is a structured sequence of computational tasks or activities that achieve a research or analytical objective. Workflows define the flow of work, including the order of steps, the data and control dependencies between them, and the rules governing their execution. Modern workflows extend beyond traditional directed acyclic graphs to encompass dynamic, adaptive, and interactive processes that may include cycles, branches, and human-in-the-loop components. They span diverse domains, including scientific research, engineering, humanities, and business, and bridge heterogeneous computing environments from edge devices to high-performance computing (HPC) facilities and cloud infrastructure.

Over the past decades, workflows have become the predominant format for describing complex, multi-step, multi-domain scientific applications [5]. To manage the composition, planning, orchestration, and automation of the efficient execution of such workflows on powerful and often distributed compute infrastructures, a wide variety of workflow management systems (WMSs) have been proposed [6]. However, domain researchers who develop workflows and want to rely on a WMS to execute them often face the complex question of selecting a particular WMS. This selection can be driven by technical considerations, such as finding the most appropriate system for their application and for the computing and storage resources available to them, or factors such as reputation, adoption, community support, or long-term sustainability. The main reasons for this challenge are that no single ideal turnkey solution has emerged from the field to address the diversity of scientific processes and the heterogeneity of possible execution environments (both in terms of hardware and software). Instead, new research problems or new computer technologies related to the execution of workflows often lead to the development of an entirely new workflow management system.

A direct consequence of this situation is that many existing WMSs share some salient features, offer similar functionalities, and can manage the same categories of workflows, but often also have distinct features tailored for specific types of problems. This has been highlighted by different efforts to create taxonomies and characterizations of workflows and WMSs [7–15]. These efforts can help to provide workflow developers with some guidance when trying to select the appropriate tool to develop and execute their workflows, but they are also notoriously incomplete and quickly outdated in a fast-moving field. Consequently, decisions for specific systems are very often based on social aspects as much or more than on technical ones (e.g., previous experience in the community, word-of-mouth, comments in web forums, personal evaluation of a few known systems). Inspired by the work of the in situ processing community for data visualization and analysis systems [16], we propose in this article to go beyond a traditional taxonomy and develop a consistent terminology to describe WMSs. While it shares similarities with and builds on existing taxonomies, the driving principle of this effort was to determine terms that consensually describe the high-level features of workflows and WMSs, rather than categorizing systems based on implementation details.

To this end, we gathered a group of workflow system developers and workflow practitioners, all members of the Workflows Community Initiative (WCI) [17], and followed a process similar to that in [16] to create a strong terminology for WMSs. This paper synthesizes the discussions initiated during the different editions of the Workflows Community Summit [18–23], which led to the writing of this paper. The main contribution of this paper is the identification of five axes to characterize WMSs (Figure 1). Each axis comprises a series of concepts that capture the most salient features of WMSs. Based on the proposed terminology, our group analyzed 23 actively developed WMSs that are part of the WCI to determine which combination of terms can define each of them.

The remainder of this paper is organized as follows. Section 2 defines the proposed five axes to describe a workflow system. Section 3 reviews the 23 selected WMSs and classifies them according to the proposed axes and terms. Section 4 describes the process followed by members of the WCI to produce this terminology and Section 5 discusses previous efforts to establish taxonomies of WMSs. Finally, Section 6 summarizes our work.

iacopo.colonnelli@unito.it (Iacopo Colonnelli), deelman@isi.edu (Ewa Deelman), paolo@seqera.io (Paolo Di Tommaso), Thomas.Fahringer@uibk.ac.at (Thomas Fahringer), Carole.Goble@manchester.ac.uk (Carole Goble), shantenujha@acm.org (Shantenu Jha), d.katz@ieee.org (Daniel S. Katz), johannes.koester@uni-due.de (Johannes Köster), leser@informatik.hu-berlin.de (Ulf Leser), mehtakv@ornl.gov (Kshitij Mehta), hilary.oliver@niwa.co.nz (Hilary Oliver), peterson76@llnl.gov (J.-Luc Peterson), giovanni.pizzi@psi.ch (Giovanni Pizzi), pottier1@llnl.gov (Loïc Pottier), Raul.Sirvent@bsc.es (Raül Sirvent), suchytaed@ornl.gov (Eric Suchyta), dthain@nd.edu (Douglas Thain), wilkinsonsr@ornl.gov (Sean R. Wilkinson), woz@anl.gov (Justin M. Wozniak), silvarf@ornl.gov (Rafael Ferreira da Silva)

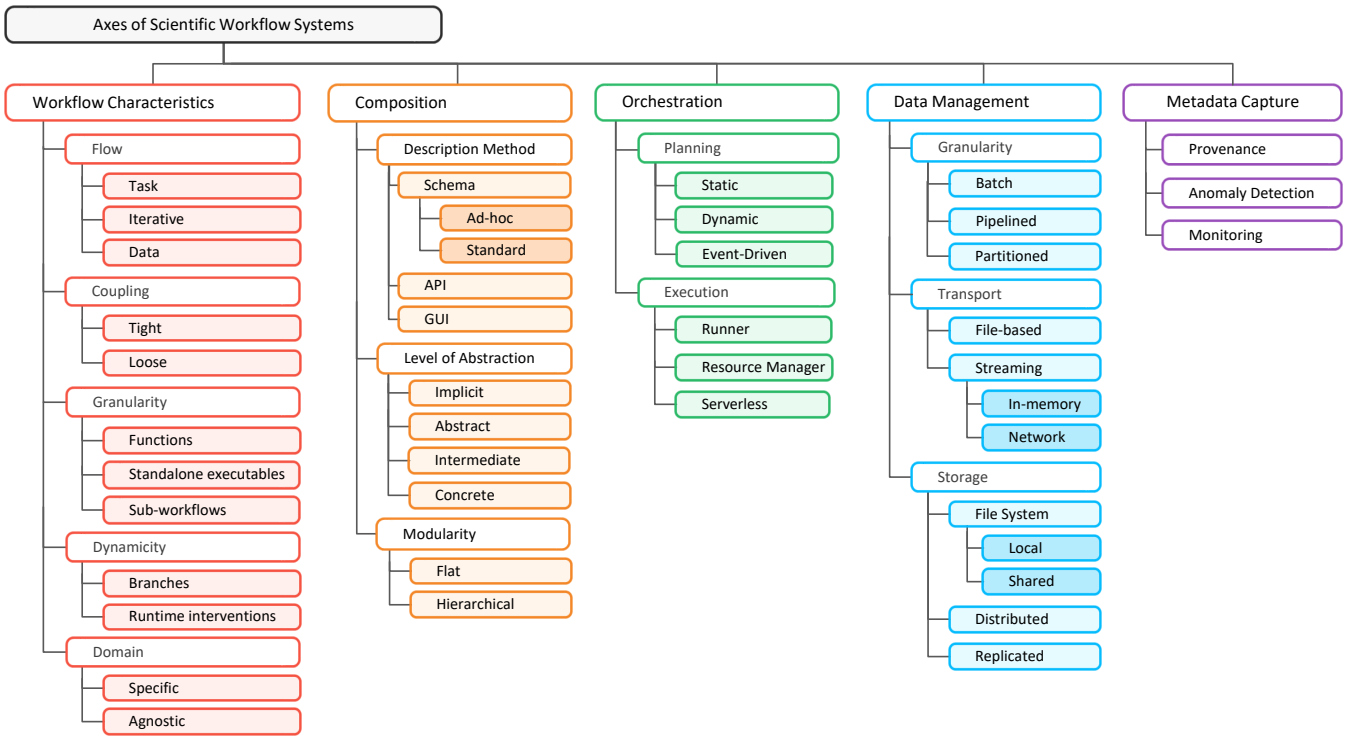

Figure 1: Five axes that categorize workflows and workflow management systems, with each axis further delineated into corresponding terms and sub-terms to provide a structured and detailed terminology.

2. Axes of Scientific Workflow Systems

WMSs often consist of subsystems that handle specific aspects of workflow management, such as resource allocation, task scheduling, or, data management. A WMS coordinates these subsystems to ensure efficient and robust execution. Additionally, characterizing a given workflow system requires considering the characteristics of the workflows it can support, as it often influences the design of the system. The primary goal of defining this terminology is to help scientists navigate the wide range of available tools [6] and better express their computational needs. To achieve this, we identified five key axes to describe a workflow system:

- **Workflow Characteristics:** This axis examines fundamental organizational aspects that impact how workflows operate and adapt. Specifically, it examines how execution is driven (by tasks or data), the level of complexity of individual components, the nature of dependencies between these components, and the ability to modify execution paths at runtime. These structural elements significantly influence how WMSs optimize resource use and performance.
- **Composition:** This axis addresses how workflows are defined, organized, and configured by WMSs. It explores the methods used to describe workflows, the level of detail required in these descriptions, and how complex workflows can be shaped from simpler components. This axis helps to understand how accessible and flexible different WMSs are for users with varying technical backgrounds.
- **Orchestration:** This axis covers the implementation and execution management approaches for workflow components. It analyzes different methods for launching and coordinating tasks, from direct execution to more sophisticated approaches that leverage distributed resources, event-based triggers, or cloud services. These orchestration strategies determine how efficiently workflows use available computing infrastructure.
- **Data management:** This axis focuses on how data is handled throughout the workflow lifecycle. It characterizes methods for moving data between workflow components, approaches to storing data at different stages, and techniques for optimizing data access patterns. These data management strategies significantly affect workflow performance, especially for data-intensive applications.
- **Metadata capture:** This axis explores additional contextual information collected during workflow execution. It covers methods for tracking workflow execution state, documenting provenance, monitoring performance, and detecting anomalies. These capabilities ensure that workflows can be reliably executed, optimized, debugged, and reproduced.

Figure 1 provides an overview of the terms used for each axis, and this section describes these terms in more detail. With this terminology, we can describe a WMS based on a selection of specific terms for each of the five identified axes. As subterms within an axis are not mutually exclusive, a WMS may be classified by a combination of sub-terms.

2.1. Workflow Characteristics

The first axis is more focused on the type of workflows a workflow system can manage than on the characteristics of a system itself, in other words, on *what* the workflow does. The large number of existing WMSs [6] indicates that there is no "one-size-fits-all" solution despite standardization and interoperability efforts [24]. In fact, the design and implementation of workflows is significantly influenced by structural aspects that are crucial to their efficiency, scalability, and adaptability. In this section, we characterize broad classes of workflows according to these defining features.

A prominent feature of a workflow is its **flow**, which has a direct impact on how WMSs optimize workflow execution. When workflow components receive inputs, process them, generate outputs, and then terminate, the workflow structure is defined by the composition of these **tasks**. WMSs are then responsible for orchestrating their execution, respecting their flow and control dependencies. They will also implement optimization strategies to improve workflow performance, such as minimizing the makespan or communication of the workflow. The different tasks that make up a workflow can also be executed multiple times in an **iterative** way. At each iteration, tasks are executed, terminated, and then wait to be invoked again. The structure and execution of the workflow can also be driven by the **data** flowing through the workflow components. These components are data operators that remain alive while there is data to process. In that case, WMSs aim to maximize the throughput of the workflow.

The structure of workflows also differs by the **granularity** of their individual tasks. Some workflows can compose some **function** calls to perform complex processing tasks. To some extent, a script or a program can be seen as a workflow and a runtime system as a workflow system. The most common definition of a workflow is a composition of **standalone executables**, which aggregate multiple functions calls to perform complex computations on a set of inputs and produce a set of outputs. With the increase in scale and complexity of computational problems, it is now common to express workflows as a hierarchical and modular composition of **sub-workflows**.

Another defining feature of workflows is the **coupling** of the tasks that compose them. This term defines the dependencies and interactions between the different tasks. The **tight** coupling of some tasks indicates that these tasks must be executed concurrently, being co-located on the same computing resources or running on different sets of processors. This is often caused by periodic data exchanges between tasks while they run. Conversely, a **loose** coupling of tasks does not impose any constraint on the concurrent execution of tasks, giving more flexibility to the WMS when scheduling the workflow.

The **dynamicity** of a workflow indicates its ability to modify its structure during its execution. Dynamic workflows can comprise several **conditional branches** that are activated or not depending on the realization of a predefined condition or triggered by an external event. Such conditions can be related to changes observed in the processed datasets (e.g., a variable reaching a certain threshold, the convergence of an iterative process is reached), to changes in the status or availability of compute, network, or storage resources, or to time-related events (e.g., it is too late to process a given execution path). Such conditional branches allow workflows and WMSs to efficiently react to changes and foster more robust, efficient, and flexible executions. A second type of dynamic behavior found in workflows is when a **runtime intervention** is needed. In that case, the workflow system gives the control back to the user who started the workflow or to an automated external decision process. Such interventions at runtime can modify the initial execution plan of a workflow in different ways (e.g., rerun certain tasks or an entire sub-workflow, modify task configuration, cut a given path or start exploring a new path, or trigger the early termination of the entire workflow).

Finally, it is possible to distinguish WMSs with respect to the **domain** they serve. Some systems are deeply rooted in a scientific community and thus mainly target domain-**specific** workflows, while others are more application-**agnostic**.

2.2. Composition

Composition refers mainly to how a workflow system allows its users to describe the different components of the workflow, their configuration and input parameters, and the data and control dependencies between these components. This axis also covers the coupling between the description of the workflow itself and that of the targeted hardware and software infrastructure on which to execute the workflow.

We identified three subcategories of **description methods** to compose a workflow. The first, **schema**, refers to the case where the workflow is described in a text file, using a specific format (e.g., XML, JSON, YAML, or a domain-specific language) and syntax. We further decompose this category to distinguish that the syntax used by a WMS is **ad-hoc**, meaning that it can only be understood by this particular WMS, or part of a common **standard** shared by multiple WMSs, such as the Common Workflow Language (CWL) [24], the Interoperable Workow Intermediate Representation (IWIR) [25], or Wf-Format [26]. Note that supporting a description standard may not always be possible, for instance, when a WMS implements significant features that cannot be easily expressed in the standard. The second subcategory includes WMSs that expose an **API** to describe workflows. This API builds on or extends one or more popular programming languages (e.g., Python, C++) or a text templating engine (e.g., jinja) to leverage loops and conditional statements and allow users to describe their workflows in a more compact and flexible way. The third subcategory corresponds to WMSs that rely on a **graphical user interface (GUI)**.

Workflow composition can also be defined by the **level of abstraction** of the description provided by the user. A **high-level abstract** composition will only focus on describing the logical structure of the task graph, a generic description of the data flowing through the workflow, and the amount of resources required by each component. This abstract description is generally independent of a specific instance of the workflow (i.e., that specifies all input parameters and component configuration parameters) and of a specific target computing and storage infras-

tructure. The advantages of an abstract workflow composition are that it favors the reusability and portability of the workflow. However, it requires more effort from users or the workflow system to execute a specific instance on a specific infrastructure.

Some systems have an **intermediate-level abstract** composition. They allow for a high-level workflow description while requiring some execution details from the users. Systems with intermediate-level abstraction provide users with a balance between automation and manual fine-tuning, which can be advantageous when the application requires a higher level of execution control. This comes at the cost of lower portability when compared to fully abstract systems and possible performance trade-offs (e.g., the responsibility of allocation optimization falls on the users in these systems).

Conversely, a **concrete** composition is more closely related to an instance and an infrastructure. All parameters are specified in the description, and the workflow can be deployed and run directly from it. Note that some WMSs allow to factor infrastructure related information as a separate description, allowing users to port a workflow from one infrastructure to another without changing the high- or intermediate-level abstract composition of the workflow itself.

When an API is used to describe a workflow, the composition is **implicit** as the workflow's structure is derived from the composition of the different function calls made by the user, or from metadata attached to a dataset to process, indicating for instance which files are needed and in what way.

Finally, we also distinguish composition methods with regard to their **modularity**. With the evolution of scientific applications from relatively simple workflows (e.g., data processing pipelines or fan-out/fan-in execution patterns for ensemble runs) to more complex workflows composed of interconnected sub-workflows (i.e., workflow of workflows), the composition methods exposed by WMSs are also evolving from a **flat** description of a set of components to a more **hierarchical** description that enables modular and scalable design. This shift allows for better management of large-scale applications, where individual sub-workflows can be developed, tested, and optimized independently before integration. It also allows researchers to create new complex data analysis workflows by composing existing workflows developed in their community. However, such hierarchical composition introduces new challenges, such as dealing with an increased orchestration complexity, handling dependencies across nested workflows, and efficiently managing resource allocation. To address these, WMSs provide features such as parameterized workflow components and reusable templates that facilitate modular workflow design while maintaining scalability.

2.3. Orchestration

Orchestration refers to the method(s) employed by a workflow system to deploy, schedule, and execute the computational components of a workflow. In this section, we focus on the general functional features of WMSs rather than the specific technical details of their implementations. For instance, we leave optimization techniques, such as advanced, performance-oriented scheduling and resource allocation techniques and algorithms, out of the scope of this axis. However, we still consider it important to classify WMSs into three broad categories related to execution **planning**. Some systems impose a **static** planning of the workflow execution, i.e., all the decisions about when and where each task composing the workflow is executed must be taken before the execution starts. Conversely, some systems can make or adapt scheduling and resource allocation decisions during the execution of the workflow, hence implementing a **dynamic** planning strategy. (Note that certain systems implementing static planning may emulate dynamic planning through hierarchical workflows.) The third category encompasses WMSs that do not plan the workflow execution in advance but rather let the execution react to specific events and/or conditions that occur at runtime. In such **event-driven** execution, when a trigger condition or event is met, the workflow system automatically initiates subsequent, usually predefined, actions such as starting new tasks, notifying users, and adjusting the resource allocation. This type of automation minimizes manual intervention, making the orchestration less error-prone.

We identified three categories for the actual **execution** of the tasks that compose a workflow. WMSs might use one or more orchestration methods (see Table 2) to execute a workflow. The **runner** orchestration method refers to WMSs that are fully responsible for the acquisition of computing and storage resources and the management of the individual tasks that compose a workflow. It connects the high-level workflow definition (i.e., its composition, see Section 2.2) to the available resources. A runner system ensures that tasks execute in the correct order, respecting their pre-defined control and flow dependencies. It oversees the life cycle of a task from the time it is dispatched and monitors it until it is completed according to its specifications.

Other WMSs delegate resource allocation and part of the management of the execution of individual tasks to a **resource manager**. This orchestration method is typically used in HPC systems where the allocation of compute nodes is handled by a batch scheduler, or cloud systems, where container orchestration systems are used. The interactions between the workflow system and the underlying resource managers encompass ordering queue of jobs to execute in an ensemble, controlling the release of limited quantities of tasks or data to not overwhelm the underlying execution system, or implementing a *pilot job* [27] mechanism to reduce the queuing overhead caused by scheduling and executing tasks independently by grouping them within the pilot allocation.

The last orchestration method relies on a **serverless** execution of tasks. This refers to a cloud-based model in which the responsibility for infrastructure management, allocation scaling, and job execution is entirely delegated to a cloud service provider. A key distinction of this model is that the user or WMS must first define one or more functions along with all of their software dependencies, and then the WMS may execute those functions to carry out the workflow. The cloud platform takes care of the provisioning and server management, abstracting the underlying computing and storage infrastructure entirely. In some cases, it can be the most cost effective orchestration method as users are usually only charged based on

the actual usage of computing resources rather than maintaining servers always on, even when idle.

2.4. Data Management

The data management axis characterizes the way WMSs transport, store, and manage the lifecycle of one of the key components of scientific workflows: data. Before detailing the different categories and terms related to data management, we make an important distinction between two types of data, as the way a workflow system manages each of them may differ. **Input/output** data respectively refer to the data needed at the beginning of the workflow and to the final outcomes of its execution, while **intermediate** data denotes every piece of data that did not exist before the beginning of the workflow and will not be kept after the end of this execution.

A first way to distinguish WMSs according to how they manage data is to consider the **granularity** at which these systems handle data management operations. A common approach followed by many WMSs is to consider the data operations of a workflow component at the granularity of a **batch**: all the needed input data are consumed before performing computations and all the output data is produced, and made available to subsequent components in the workflow, at the end of these computations.

Another approach is to consider a **pipelined** granularity in which workflow components periodically produce and/or consume individual records during their entire lifecycle. This is typically used to manage in situ processing workflows [28], where analysis and visualization components are loosely coupled to a main data producer (i.e., a numerical simulation). In such workflows, data is consumed as it is produced, in opposition to a *post-hoc* approach in which analyses or visualization happens once the full dataset has been generated.

A third intermediate granularity is to consider data as **partitioned**, i.e., divided in groups of individual records, and to transfer these partitions across the workflow. This approach is particularly useful when individual records are small. Considering them individually would be very latency-sensitive and could negatively impact performance.

A second way to differentiate WMSs is by how they **transport** data from one workflow component to another. Again, a common approach is to rely on **file-based** transport, in which a workflow component that produces intermediate data will write them into a file(s) on a storage system. In contrast, a workflow component that consumes intermediate data will read it from file(s). An alternate approach is to directly **stream** intermediate data between components. Depending on the respective allocations of the producing and consuming components, it is possible to further refine these two broad approaches.

For WMSs that rely on the file-based transport approach, we can further distinguish them according to the **storage** they use. When workflow components are co-located on the same compute node, the workflow system can leverage the existence of a **local file system**, while when components are allocated to different nodes of the same compute cluster or to different clusters of the same computing facility, it will have to rely on a **shared file system**. Commonly used in collaborative or high-performance computing environments, shared file systems correspond to a centralized model where data is accessible by multiple systems or nodes simultaneously. They bring several advantages when executing workflows, such as simple and collaborative access to a unified storage space or good cost efficiency. They also come with different challenges, such as data consistency, performance bottlenecks, scalability, or security, that a WMS will have to face, and may address. In the extreme case where the execution of a workflow is distributed over multiple computing facilities, this approach can leverage a **distributed storage space**. This involves managing and storing data across multiple local and/or remote systems, enabling scalability, load balancing, resilience, and flexibility. Although it can resolve some issues of shared file systems, data consistency and security challenges persist. Furthermore, the management of such systems can be very complex, and data accesses may suffer from high latencies. An alternative approach in that case would be that the data-producing workflow components running in a given facility create one or several additional transfer tasks to send data to each their its data-consuming successors that run in another facility. Another common practice in distributed and shared systems targeted by WMSs is the use of **replicated storage**, which focuses on creating redundant copies of data to improve reliability, availability, and resilience. Such as the aforementioned storage solutions, replicated storage struggles with data consistency and complex data management, not to mention the increased storage costs and the write overhead created every time data needs to be updated.

For the stream-based transport approach, when the producer and consumer are co-located on the same node, data transport can be carried out **in-memory** through a shared address space. Otherwise, it implies a **network** communication between the nodes that respectively hosts the data producer and consumer.

2.5. Metadata Capture

The last axis of the terminology refers to the different categories of contextual information, or metadata, captured by WMSs during a workflow execution. Metadata constitute a critical layer of information that describes, tracks, and contextualizes workflow aspects such as inputs/outputs, parameters, and dependencies. Through the extra information, the workflow engine can decide on the execution order based on the dependency information, parallelize tasks, and schedule resources according to the needs of the task. Therefore, metadata enables efficient orchestration, automation, long-term data management, and resource optimization. By capturing descriptive execution logs and storing full context results, metadata can improve troubleshooting, debugging, and responsiveness. Overall, it can ensure scientific integrity, reproducibility, and reliability throughout the workflow's lifecycle.

Workflows are typically large and complex applications designed for execution in distributed systems. Given their role in critical research and high-impact projects, the ability to reproduce results enables others to validate the findings, build upon previous work, and promote collaboration to further scientific discovery.

A specific type of metadata is **provenance** data which can be further decomposed into prospective and retrospective provenance data. Prospective provenance corresponds to maintaining detailed information about the workflow design and structure, the configuration of the workflow system and the underlying computing and storage infrastructure, and the specific algorithms to be used and their parametrization. Prospective provenance is essential to facilitate reproducibility, especially for complex applications such as workflows [29]. Retrospective provenance data corresponds to what actually happened to the data processed by a workflow and captures everything related to a specific execution. It is usually extracted from execution logs to keep track of the data lineage (i.e., generation, transformation, and usage) and timestamps and runtime details. Retrospective provenance is particularly useful for detecting any deviation from the expected execution plan and is often used for debugging purposes.

Another type of metadata captured during workflow executions is **monitoring** of data, which comes from processes that oversee the workflow execution in real time. The data generated by monitoring provides critical insight into performance, resource utilization, and potential bottlenecks. WMSs can leverage it to dynamically reconsider an initial execution plan by modifying resource allocations or scheduling decisions. The monitoring data can also be analyzed by researchers after a workflow execution to optimize the description of the workflow itself to improve its efficiency.

The final category on this axis is related to **anomaly detection** [30]. We consider that a workflow management system supports anomaly detection if it captures metadata that can be used to implement fault tolerance mechanisms. These mechanisms vary in sophistication: Some systems terminate execution and display an error message, while others complete the execution but log warnings about potentially incorrect data resulting from unexpected behavior. There are even systems that can distinguish between anomalies that can be handled automatically (e.g., task retries or by an optional branch from a task-failed trigger) and anomalies that the workflow is not designed to handle and thus require user intervention. In the latter case, the scheduler remains alive on a timeout in a "stalled" state, awaiting operator intervention.

3. Surveying Existing Workflow Systems

This section considers **23 WMSs** that are part of the Workflows Community Initiative (WCI). This selection is motivated by the fact that the WCI focuses on *actively developed* WMSs with a *large user base*. We also ensured that the selection made was not limited to a specific research community, a narrow set of origin countries, or a certain category of supported workflows to avoid biases in the definition of our terminology. Although this list represents only a small fraction of the vast number of existing WMSs [6] and is thus far from being exhaustive, we believe that it is still representative of the diversity of the available systems. Moreover, this initial list of analyzed systems is not definitive nor intended to be limited to WMSs affiliated to the WCI. We plan to make this terminology available on the WCI website and broadly advertise its existence so that the list of WMSs mapped to the terminology continues to grow.

For each WMS, we analyze their published work and incorporate feedback from community efforts over the past four years. Table 1 summarizes the type of workflows each system is able to execute, while Table 2 highlights the primary characteristics of each system according to the axes and terms summarized in Figure 1 and detailed in Section 2. Table 2 also includes a column named `extensions`, which lists additional functionalities that WMSs can support beyond their default configurations. These extensions may include optional plugins, third-party integrations, or interoperability with cloud-based storage and computing resources.

Evolution of Workflow Characteristics. The evolution of WMSs in the past two decades reflects significant changes in computational approaches. Initially predominantly task-driven, workflows have expanded to embrace data-driven processing pipelines with the rise of big data. Modern workflows now integrate both paradigms, particularly as AI becomes embedded in research, enabling complex analytical pipelines that respond dynamically to data while preserving the structured execution needed for reproducibility. The growing complexity of applications called for greater composability and modern WMSs now support hierarchical sub-workflows and iterative processes, which allows researchers to independently develop and optimize components before integration. These systems have also evolved to support more dynamic execution through conditional branches, runtime interventions, and adaptive processing. However, while technical capabilities continue to expand, the scientific domains supported by WMSs are often determined more by social dynamics than by technical limitations.

The Social Dynamics of Workflow System Selection. While the technical characteristics described in our terminology provide a foundation for evaluating WMSs, the actual selection process in practice is often significantly based on social factors. Our community observations reveal that researchers frequently choose WMSs based not solely on technical merits, but on established social patterns and connections. When confronted with multiple technically viable options, scientists typically gravitate toward systems already in use by their immediate collaborators, departmental colleagues, or disciplinary communities. This preference for socially validated tools creates adoption groups within research domains and institutions. The perceived credibility of a workflow system is substantially enhanced when it appears in trusted publications or receives endorsements from respected colleagues. In addition, institutional knowledge transfer plays a crucial role, as existing expertise and support infrastructures significantly lower the barrier to adoption. These social dynamics create self-reinforcing adoption patterns that can sometimes override purely technical considerations, highlighting that workflow system selection exists within a complex socio-technical ecosystem where community practices, established knowledge bases, and trusted relationships often determine final choices. Nevertheless, this understanding emphasizes why developing a common terminology is particularly valuable, i.e., it provides a framework for dis-

Name		Flow	Granularity	Coupling	Dynamicity	Domain
AiiDA	[31]	Task Iterative	Sub-workflows Executables Functions	Loose	Branches Runtime intervention	Agnostic
AirFlow	[32]	Task	Executables	Loose	Branches	Agnostic
Apollo	[33]	Task Data Iterative	Functions Sub-workflows	Loose	Branches	Agnostic
COMPSs	[34]	Task Iterative	Functions Sub-workflows Executables	Loose	Branches	Agnostic
Cylc	[35]	Task Iterative	Executables Sub-workflows	Loose	Branches Runtime intervention	Agnostic
Dask	[36]	Data	Executables	Tight	-	Agnostic
EFFIS	[37]	Data	Executables	Tight Loose	-	Specific
FireWorks	[38]	Task	Sub-workflows	Tight	Branches	Agnostic
Galaxy	[39]	Data	Executables Sub-workflows	Loose	Branches Runtime intervention	Agnostic
Globus Compute	[40]	Data	Functions Executables	Loose	-	Agnostic
HyperFlow	[41]	Data	Functions Executables	Loose	-	Agnostic
Makeflow	[42]	Data	Sub-workflows	Loose	-	Agnostic
Merlin	[43]	Task Iterative	Sub-workflows	Loose	-	Agnostic
MLFlow	[44]	Task Iterative	Executables	Loose	-	Specific
Nextflow	[45]	Data	Sub-workflows	Loose	Branches	Agnostic
Parsl	[46]	Data	Sub-workflows	Loose	Branches	Agnostic
Pegasus	[47]	Data	Sub-workflows Executables	Loose	Branches	Agnostic
Radical	[48]	Task Iterative	Functions	Tight	-	Agnostic
Snakemake	[49]	Task Iterative	Sub-workflows Executables Functions	Loose Tight	Branches	Agnostic
StreamFlow	[50]	Task Data Iterative	Sub-workflows Executables	Loose	Branches	Agnostic
Swift/T	[51]	Task Data	Functions	Tight	Branches Recursion	Agnostic
TaskVine	[52]	Task Iterative	Functions Executables	Loose	-	Agnostic
Toil	[53]	Data	Sub-workflows	Loose	Branches	Agnostic

Table 1: Classification of workflow management systems based on structure and characteristics.This classification represents the state at the time of publication, to the best of the authors knowledge. As many of the presented WMSs constantly evolve, we suggest the reader to explore their respective documentation to get an up-to-date view of their capabilities and characteristics.

cussing technical aspects objectively while acknowledging the legitimate influence of social factors on technology adoption.

Emerging Patterns in Modern Workflow Systems. Several significant trends are reshaping the landscape of WMSs. The traditional schema-based approach to workflow composition is giving way to API-driven interfaces, reflecting broader programming paradigm shifts and resulting in less abstract, more programmatic workflow descriptions. This transition enables finer control over workflow execution while sometimes sacrificing portability across environments. Simultaneously, WMSs are increasingly addressing the need for dynamic execution capabilities, responding to growing demands from scientific applications that require adaptive runtime behaviors and conditional processing paths. Data management approaches are also evolv-

Name	Composition			Orchestration		Data Management		Metadata	Extensions
	Description	Abstraction	Modularity	Planning	Execution	Transport	Storage	Capture	
AiiDA	API	Intermediate	Hierarchical	Dynamic	Runner	File-based	Shared	Anomaly Provenance	Plugins Caching Fault tolerance HPC execution
AirFlow	API	Intermediate	Flat	Static	Runner	Stream	Shared	Monitoring	Dynamic pipelines
Apollo	Ad-hoc Schema	Abstract	Hierarchical	Dynamic	Resource Manager Serverless	Stream	Distributed	Monitoring	Container/serverless Multi-cloud Edge/cloud Multi-objective scheduling
COMPSs	API	Intermediate	Flat Hierarchical	Dynamic	Resource Manager Serverless	Stream File-based	Local Shared Distributed	Anomaly Monitoring Provenance	Adaptive resource allocation HPC scalable Replicated storage
Cylc	Ad-hoc Schema API/templating	Concrete	Flat Hierarchical	Static Event-driven	Runner Resource-manager	File-based	Shared	Anomaly Provenance Monitoring	HPC Execution Plugins Config templating
Dask	API	Concrete	Flat	Dynamic	Runner	Stream	Shared Distributed	Anomaly Monitoring Metadata	Python Libraries Cluster Management GPU Accel.
EFFIS	API	Intermediate	Flat	Dynamic	Resource Manager	Stream File-based	Shared Distributed Replicated	Anomaly Monitoring	
FireWorks	API Ad-hoc Schema	Intermediate	Hierarchical	Dynamic	Resource Manager	File-based	Shared Replicated	Anomaly Monitoring Provenance	Multi-platform execution
Galaxy	GUI Ad-hoc Schema	Concrete	Flat	Event-Driven	Runner	Stream	Shared	Anomaly Monitoring Provenance	External Tools Execution API
Globus Compute	API	Abstract	Hierarchical	Dynamic	Resource Manager Serverless	Stream File-based	Shared	Anomaly Monitoring	Distributed storage
HyperFlow	Ad-hoc Schema	Intermediate	Flat	Static Dynamic	Runner	Stream	Shared Distributed	Provenance	Replicated storage Cloud Integration Scalability
Makeflow	Standard (Make)	Abstract	Hierarchical	Static	Runner	File-based	Shared Replicated	Anomaly Monitoring Provenance	Distributed storage HPC execution
Merlin	Ad-hoc Schema	Intermediate	Hierarchical	Static	Runner	File-based	Shared Distributed Replicated	Anomaly Monitoring Provenance	Cloud-native Support
MLFlow	API	Intermediate	Flat	Static	Runner	File-based Stream	Shared Distributed	Monitoring	
NextFlow	Ad-hoc Schema	Abstract	Hierarchical	Dynamic	Runner	Stream File-based	Shared Distributed	Anomaly Monitoring Provenance	Replicated storage Container/Cloud Support HPC execution
Parsl	API	Abstract	Hierarchical	Dynamic	Runner	Stream File-based	Shared Distributed	Anomaly Monitoring	Replicated storage Dynamic Parallelization Cloud/Grid Support
Pegasus	Ad-hoc Schema API	Abstract	Hierarchical	Static	Runner	File-based	Shared Distributed	Anomaly Monitoring Provenance	Replicated storage Multi-level Scheduling
Radical	API	Abstract	Hierarchical	Static	Resource Manager	File-based	Shared Distributed	Anomaly Monitoring Provenance Metadata	Replicated storage Scalable
Snakemake	Ad-hoc Schema	Abstract	Flat Hierarchical	Static Dynamic Event-driven	Runner	File-based	Shared Distributed	Anomaly Monitoring Provenance	Plugins Scripting integration Software deployment integration Interactive reporting
StreamFlow	Standard (CWL)	Abstract	Hierarchical	Dynamic	Runner Resource Manager	File-based	Distributed	Anomaly Provenance	Replicated storage Cloud Integration
Swift/T	Ad-hoc Schema	High-level	Flat	Dynamic	Resource Manager	Stream File-based	Shared	Anomaly Monitoring	Local Storage [54] AI/ML Control [55] Parallel Tasks [56]
Taskvine	API	Intermediate	Flat	Dynamic	Resource Manager	File-Based	Shared Distributed Replicated	Anomaly Monitoring Provenance	Serverless Autoscaling HPC Execution Recoverable storage
Toil	Standard (CWL/WDL)	Abstract	Hierarchical	Static	Runner	Stream File-based	Shared Distributed	Anomaly Monitoring Provenance	Replicated storage Multi-Cloud Support

Table 2: Categorization of various workflow systems with respect to their composition, orchestration, data management, and information capture. In addtion, the last column highlights exemplary extensions provided beyond this common terminology. This classification represents the state at the time of publication, to the best of the authors knowledge. As many of the presented WMSs constantly evolve, we suggest the reader to explore their respective documentation to get an up-to-date view of their capabilities and characteristics.

ing in response to the explosive growth in data volumes and velocity; While file-based transport remains common, streaming approaches are gaining traction for near real-time processing needs. When file handling is required, modern WMSs must navigate complex storage hierarchies and scale horizontally across distributed storage locations to maintain performance. These trends collectively point toward more sophisticated and flexible systems that can adapt to diverse scientific computing requirements while managing increasingly complex data ecosystems.

From Extensions to Building Blocks. As WMSs mature, developers increasingly extend their native capabilities through additional components that address specific needs. This expansion has led to growing system complexity, challenging developers to maintain modular architecture and avoid unwieldy monolithic designs. Rather than each system independently implementing similar functionalities, a promising approach for the workflow community involves identifying and developing shared building blocks, reusable components that provide common services across different WMSs [48, 57, 58]. This community-based approach to the development of modular and interoperable components [58] could significantly reduce duplication of efforts while improving sustainability and adoption. Such standardized building blocks would address fundamental workflow needs like resource management, data movement, provenance tracking, and fault tolerance, allowing individual systems to focus on their unique strengths and domain-specific optimizations. The emergence of these community-maintained components represents a potential path toward consolidation in a currently fragmented ecosystem of over 300 WMSs, promoting interoperability while preserving the specialized capabilities that particular scientific domains require.

Workflow Registries in the Scientific Workflow Ecosystem. In addition to the WMSs themselves, the scientific community has developed various workflow registries that serve as centralized locations to share, discover, and reuse workflow definitions in the workflow ecosystem. These registries complement WMSs by facilitating knowledge exchange and promoting best practices across research domains. The nf-core [59] repository provides community-maintained curated Nextflow workflows for bioinformatics with continuous integration to ensure reproducibility. Similarly, the SnakeMake workflow catalog [60] offers domain-specific collections. WorkflowHub [61] provides a unified registry for all computational workflows that links to community repositories, making workflows findable, accessible, interoperable, and reusable (FAIR) according to the FAIR principles for workflows [62]. Unlike single-language workflow registries such as nf-core, the AiiDA plugin registry [63], and Galaxy Toolshed [64] that are associated with specific workflow platforms, WorkflowHub accepts workflows from any scientific domain, in any format and in any workflow language. Repositories such as Dockstore [65] improve reproducibility by combining containers, descriptor languages, and test parameter files to simplify software reuse and dependency management. Dockstore has facilitated large-scale biomedical research collaborations by using cloud technologies to increase the FAIRness of computational resources. WfInstances [26] is a key component of the WfCommons project that archives real-world workflow instances collected from workflow executions using various runtime systems. The repository ecosystem represents an important extension of the workflow landscape, bridging technical capabilities with community practices, and helping scientists navigate the complex decision space of workflow selection and reuse while promoting the recognition of workflows as artifacts.

4. Process to Define the Terminology of Workflow Systems

The terminology for scientific workflow systems presented in this paper emerged from a systematic, community-driven approach initiated in 2021 through the Workflows Community Initiative (WCI) [17]. This collaborative effort united workflow system developers, domain scientists, and workflow practitioners in diverse scientific disciplines and computing facilities. Through a series of Workflows Community Summit events [18–23], participants engaged in structured discussions about key aspects of scientific workflow systems, including essential features, challenges in interoperability, data management approaches for execution models and reproducibility requirements. These discussions were documented in technical reports that captured the evolving understanding of WMSs and established the foundations for a unified terminology, drawing inspiration from similar efforts in the in situ processing community [16].

The development of the terminology progressed through several phases, beginning with an analysis of summit reports and the existing literature on workflow taxonomies [7–15] to identify common patterns and classification schemes. A core working group then conceptualized the framework around five distinct axes to comprehensively cover the key aspects of workflows and WMSs, followed by the creation of a draft document defining these axes and their associated terms. This draft included an initial characterization of 23 representative WMSs and was circulated to workflow system developers and key stakeholders for critical feedback. Through multiple iterations of refinement based on community input, the working group adjusted definitions, added missing terms, and ensured that the terminology accurately represented the domain's complexity while remaining both comprehensible and practical. The terminology was validated by applying it to classify the WMSs listed in Table 2, confirming its applicability while revealing its ability to highlight commonalities and distinctions among diverse systems.

Throughout this process, the working group adhered to the principles of comprehensiveness, accessibility, neutrality, openness, and practicality. The terminology needed to cover the full spectrum of workflow system features without favoring particular implementation approaches, while remaining understandable to both experts and domain scientists. It was designed to be descriptive rather than prescriptive, avoiding implications that certain approaches were inherently superior and flexible enough to accommodate future innovations through the addition of new terms within the established axes. The resulting terminology,

as detailed in Section 2 and applied in Section 3, represents the collective expertise of a broad community of workflow researchers and practitioners, providing a common language for discussing and comparing WMSs that facilitates both scientific communication and informed decision-making when selecting workflow technologies for specific research needs.

5. Related Work

Over the past two decades, the scientific workflow community has proposed several taxonomies to structure the design space of WMSs. Early taxonomies introduced classification schemes based on architectural and infrastructural features, including workflow representation models (e.g., DAGs versus non-DAGs), scheduling strategies (i.e., centralized, decentralized, or hierarchical), fault tolerance mechanisms (e.g., task retries, checkpointing, alternate resource usage), and data movement techniques (e.g., file staging, replication, streaming) [7]. These taxonomies highlighted trade-offs between performance, fault resilience, and scalability across grid environments. Later frameworks organized WMS features according to the workflow lifecycle, encompassing composition interfaces (e.g., graphical editors, scripting APIs, domain-specific languages), resource mapping mechanisms (i.e., manual binding versus automated planners), execution engines (i.e., static versus dynamic schedulers), and provenance capture strategies (i.e., retrospective and prospective metadata logging) [9]. These classifications helped emphasize usability and reproducibility as central design goals. More recent comparative analyses have expanded the evaluation criteria to cover support for heterogeneous execution models (including iterative, streaming, and conditionally adaptive workflows), deployment flexibility across HPC, cloud, and hybrid environments, and mechanisms for handling large-scale, data-intensive workloads with performance-aware orchestration and optimized I/O strategies [8, 12, 13], or focusing on specific features such as fault tolerance [14] or provenance [15]. Such studies increasingly incorporate practical interoperability, expressiveness, and usability assessments to guide system selection in data-intensive scientific domains.

These taxonomies have provided valuable frameworks for evaluating and selecting WMSs, but they often emphasize either infrastructure-level capabilities or comparisons based on the workflow lifecycle. This paper contributes a complementary approach by proposing a terminology instead of defining yet another hierarchical taxonomy. The objectives are to offer a vocabulary that captures the essential properties of WMSs in a flexible and non-prescriptive manner, support consistent descriptions across heterogeneous systems, and help researchers express requirements and understand systems' capabilities more precisely. Thus, what distinguishes this work is its focus on standardizing language rather than classification alone. By moving away from rigid taxonomies and toward shared terms, we expect to enable clearer communication across domains and stakeholder groups, and support the design, comparison, and integration of next-generation WMSs. Grounded in broad community consensus, the proposed set of terms overlaps with the existing taxonomies, which shows its capacity to capture the main features of classical WMSs. However, it also reflects the evolution of workflow practices, with new terms including dynamic execution behaviors, modular reuse, and serverless orchestration models.

6. Conclusion

In this paper, we have introduced a new terminology for scientific workflow systems. This terminology comprises five axes along which a workflow system can be characterized. Each axis is then refined via multiple associated terms. The development of this terminology is a community-based effort rooted in and supported by the Workflows Community Initiative (WCI). It summarizes the collective thinking of WMS developers and members of the leadership and steering committees of the Workflows Community Initiative and reflects the achieved consensus around an initial set of terms. The main motivation for this work is to serve as a starting point for a uniformly understood vocabulary that would help workflow practitioners navigate the vast market of WMSs. To this end, we used this terminology to characterize a selection of existing WMSs, identify similarities and differences, and highlight some broad trends. This approach brought in many different perspectives and ensured that diverse perspectives were taken into account. It also provides this terminology with solid foundations and the backing of a significant number of workflow system developers and workflow practitioners. This will allow us to expose and explain the terminology to the respective user communities of the analyzed frameworks and foster its broader adoption. We also plan to gather and analyze user feedback and monitor the adoption of the terminology to conduct an empirical validation of the benefits of the proposed terminology. A concrete metric of success for the adoption of the terminology will be to be referred to in scientific articles, not by citing this paper but by using the terminology to describe a WMS or a workflow and position contributions using a uniformly understood vocabulary accepted by a broad community.

This terminology should not be considered static. As new systems are developed and new trends emerge from the community, new terms and axes may be introduced. A new working group of the WCI will be formed, which will include this paper's co-authors to ensure that the terminology evolves and keeps reflecting the state of the field. This group will also be in charge of extending the list of characterized WMSs beyond those that are part of the Workflows Community Initiative.

Acknowledgments

The authors express their deepest appreciation for the insightful review and comments from Khalid Belhajjame of the University Paris-Dauphine (France); Luiz Gadelha of the German Cancer Research Center (DKFZ, Germany); Johan Gustafsson of Australian BioCommons and Sehrish Kanwal of the Centre for Cancer Research at the University of Melbourne

(Australia); and Mahnoor Zulfiqar and Stuart Owen of the University of Manchester (United Kingdom).

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. BSC authors acknowledge projects CEX2021-001148-S and PID2023-147979NB-C21 from the MCIN/AEI and MICIU/AEI /10.13039/501100011033 and by FEDER, UE, and by the Departament de Recerca i Universitats de la Generalitat de Catalunya, research group MPiEDist (2021 SGR 00412). Ewa Deelman is funded by the U.S. Department of Energy under grant No. DE-SC0024387 and by the U.S. National Science Foundation under grant No. 2138286. This work was performed under the auspices of the US Department of Energy (DOE) by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work has been supported by the LDRD at Lawrence Livermore National Laboratory (24-SI-005), LLNL Release number: LLNL-JRNL-2007841. Giovanni Pizzi acknowledges financial support from the NCCR MARVEL, a National Centre of Competence in Research, funded by the Swiss National Science Foundation (grant number 205602), by the Open Research Data Program of the ETH Board (project "PREMISE": Open and Reproducible Materials Science Research) and by the SwissTwins project, funded by the Swiss State Secretariat for Education, Research and Innovation (SERI). Bartosz Balis is funded by the European Union through the Horizon Europe CLOUDSTARS project (101086248). Douglas Thain acknowledges support from National Science Foundation Grant OCI-2411436. Thain, Chard, Jha, and da Silva acknowledge support from National Science Foundation grant TIP-2346119.

References

[1] S. Williams, Business Process Modelling Improves Administrative Control, Automation (1967) 44–50.

[2] J. Wiener, Y. Ioannidis, A Moose and a Fox Can Aid Scientists with Data Management Problems, in: Proceedings of the Fourth International Workshop on Database Programming Languages, 1994, pp. 376–398. doi:10.1007/978-1-4471-3564-7_21.

[3] J. Wainer, M. Weske, G. Vossen, C. B. Medeiros, Scientific Workflow Systems, in: Proceedings of the NSF Workshop on Workflow and Process Automation Information Systems, 1996.

[4] A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, A. Wolf, Report from the NSF Workshop on Workflow and Process Automation in Information Systems, ACM SIGMOD Record 25 (1996) 55–67. doi:10.1145/245882.245903.

[5] R. M. Badia Sala, E. Ayguadé Parra, J. J. Labarta Mancho, Workflows for science: A challenge when facing the convergence of hpc and big data, Supercomputing frontiers and innovations 4 (1) (2017) 27–47.

[6] P. Amstutz, M. Mikheev, M. Crusoe, N. Tijanić, S. Lampa, et al., Existing Workflow systems, [Online] https://s.apache.org/existing-workflow-systems, updated 2025-03-17, accessed 2025-04-23 (2024).

[7] J. Yu, R. Buyya, A Taxonomy of Scientific Workflow Systems for Grid Computing, SIGMOD Rec. 34 (3) (2005) 44–49. doi:10.1145/1084805.1084814.

[8] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A Characterization of Workflow Management Systems for Extreme-Scale Applications, Future Generation Computer Systems 75 (2017) 228–238. doi:10.1016/j.future.2017.02.026.

[9] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-Science: An overview of Workflow System Features and Capabilities, Future Generation Computer Systems 25 (5) (2009) 528–540. doi:10.1016/j.future.2008.06.012.

[10] J. Liu, E. Pacitti, P. Valduriez, M. Mattoso, A Survey of Data-Intensive Scientific Workflow Management, Journal of Grid Computing 13 (4) (2015) 457–493. doi:10.1007/s10723-015-9329-8.

[11] E. M. Bahsi, E. Ceyhan, T. Kosar, Conditional Workflow Management: A Survey and Analysis, Scientific Programming 15 (4) (2007) 680291. doi:10.1155/2007/680291.

[12] A. D. Kiran, M. C. Ay, J. Allmer, Criteria for the evaluation of workflow management systems for scientific data analysis, Journal of Bioinformatics Systems Biology 6 (2) (2023) 16–31.

[13] Z. Ahmad, A. I. Jehangiri, M. A. Ala'anzy, M. Othman, R. Latip, S. K. U. Zaman, A. I. Umar, Scientific Workflows Management and Scheduling in Cloud Computing: Taxonomy, Prospects, and Challenges, IEEE Access 9 (2021) 53491–53508. doi:10.1109/ACCESS.2021.3070785.

[14] D. Poola, M. A. Salehi, K. Ramamohanarao, R. Buyya, Chapter 15 - A Taxonomy and Survey of Fault-Tolerant Workflow Management Systems in Cloud and Distributed Computing Environments, in: I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, B. Maxim (Eds.), Software Architecture for Big Data and the Cloud, Morgan Kaufmann, Boston, 2017, pp. 285–320. doi:https://doi.org/10.1016/B978-0-12-805467-3.00015-6.

[15] S. M. Serra da Cruz, M. L. M. Campos, M. Mattoso, Towards a Taxonomy of Provenance in Scientific Workflow Management Systems, in: 2009 Congress on Services - I, 2009, pp. 259–266. doi:10.1109/SERVICES-I.2009.18.

[16] H. Childs, S. Ahern, J. Ahrens, A. Bauer, J. Bennett, et al., A Terminology for in situ Visualization and Analysis Systems, International Journal of High-Performance Computing and Applications 34 (6) (2020) 676–691. doi:10.1177/1094342020935991.

[17] Workflows Community Initiative, [Online, last accessed 03/2025] https://workflows.community (2025).

[18] R. Ferreira da Silva, H. Casanova, K. Chard, I. Altintas, R. M. Badia, B. Balis, T. a. Coleman, F. Coppens, F. Di Natale, B. Enders, T. Fahringer, R. Filgueira, G. Fursin, D. Garijo, C. Goble, D. Howell, S. Jha, D. S. Katz, D. Laney, U. Leser, M. Malawski, K. Mehta, L. Pottier, J. Ozik, J. L. Peterson, L. Ramakrishnan, S. Soiland-Reyes, D. Thain, M. Wolf, A community roadmap for scientific workflows research and development, in: Proceedings of the IEEE Workshop on Workflows in Support of Large-Scale Science, 2021, pp. 81–90. doi:10.1109/WORKS54523.2021.00016.

[19] R. Ferreira da Silva, H. Casanova, K. Chard, T. Coleman, D. Laney, D. Ahn, S. Jha, D. Howell, S. Soiland-Reys, I. Altintas, et al., Workflows community summit: Advancing the state-of-the-art of scientific workflows management systems research and development, arXiv preprint arXiv:2106.05177 (2021).

[20] R. Ferreira da Silva, H. Casanova, K. Chard, D. Laney, D. Ahn, S. Jha, C. Goble, L. Ramakrishnan, L. Peterson, B. Enders, et al., Workflows community summit: Bringing the scientific workflows community together, arXiv preprint arXiv:2103.09181 (2021).

[21] R. Ferreira Da Silva, K. Chard, H. Casanova, D. Laney, D. Ahn, S. Jha, W. E. Allcock, G. Bauer, D. Duplyakin, B. Enders, et al., Workflows Community Summit: Tightening the Integration Between Computing Facilities and Scientific Workflows, arXiv preprint arXiv:2201.07435 (2022).

[22] R. Ferreira da Silva, R. M. Badia, V. Bala, D. Bard, T. Bremer, I. Buckley, S. Caino-Lores, K. Chard, C. Goble, S. Jha, D. S. Katz, D. Laney, M. Parashar, F. Suter, N. Tyler, T. Uram, I. Altintas, et al., Workflows Community Summit 2022: A Roadmap Revolution, Tech. Rep. ORNL/TM-2023/2885, Oak Ridge National Laboratory (Mar. 2023). doi:10.5281/zenodo.7750670.

[23] R. Ferreira Da Silva, D. Bard, K. Chard, S. De witt, I. Foster, T. Gibbs, C. Goble, W. Godoy, J. Gustafsson, U.-U. Haus, et al., Workflows Community Summit 2024: Future Trends and Challenges in Scientific Workflows, Tech. rep., Oak Ridge National Laboratory (ORNL) (10 2024). doi:10.2172/2474744.

[24] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanić, H. Ménager, S. Soiland-Reyes, B. Gavrilović, C. Goble, et al., Methods included: standardizing computational reuse and portability with the

common workflow language, Communications of the ACM 65 (6) (2022) 54–63.
[25] K. Plankensteiner, J. Montagnat, R. Prodan, IWIR: a language enabling portability across grid workflow systems, in: Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science, 2011, p. 97–106. doi:10.1145/2110497.2110509.
[26] T. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, R. Ferreira da Silva, WfCommons: A Framework for Enabling Scientific Workflow Research and Development, Future Generation Computer Systems 128 (2022) 16–27. doi:10.1016/j.future.2021.09.043.
[27] M. Turilli, M. Santcroos, S. Jha, A Comprehensive Perspective on Pilot-Job Systems, ACM Computing Surveys 51 (2) (Apr. 2018). doi:10.1145/3177851.
URL https://doi.org/10.1145/3177851
[28] S. Caino-Lores, M. Cuendet, J. Marquez, E. Kots, T. Estrada, E. Deelman, H. Weinstein, M. Taufer, Runtime Steering of Molecular Dynamics Simulations Through In Situ Analysis and Annotation of Collective Variables, in: Proceedings of the Platform for Advanced Scientific Computing Conference, ACM, 2023. doi:10.1145/3592979.3593420.
[29] S. Leo, M. R. Crusoe, L. Rodríguez-Navas, R. Sirvent, A. Kanitz, P. De Geest, R. Wittner, L. Pireddu, D. Garijo, J. M. Fernández, et al., Recording provenance of workflow runs with RO-Crate, PLoS ONE 19 (9) (2024) e0309210.
[30] K. Raghavan, G. Papadimitriou, H. Jin, A. Mandal, M. Kiran, P. Balaprakash, E. Deelman, Advancing Anomaly Detection in Computational Workflows with Active Learning, Future Generation Computer Systems 166 (2025) 107608. doi:10.1016/j.future.2024.107608.
[31] S. P. Huber, 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. Merkys, A. Cepellotti, N. Mounet, N. Marzari, B. Kozinsky, G. Pizzi, AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance, Scientific Data 7 (1) (2020) 300. doi:10.1038/s41597-020-00638-4.
[32] P. Singh, Airflow, Apress, Berkeley, CA, 2019, pp. 67–84. doi:10.1007/978-1-4842-4961-1_4.
[33] F. Smirnov, C. Engelhardt, J. Mittelberger, B. Pourmohseni, T. Fahringer, Apollo: Towards an Efficient Distributed Orchestration of Serverless Function Compositions in the Cloud-Edge Continuum, in: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing, 2021, pp. 1–10.
[34] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Alvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, R. M. Badia, Servicess: An Interoperable Programming Framework for the Cloud, Journal of grid computing 12 (2014) 67–91.
[35] H. Oliver, M. Shin, D. Matthews, O. Sanders, S. Bartholomew, A. Clark, B. Fitzpatrick, R. van Haren, R. Hut, N. Drost, Workflow automation for cycling systems, Computing in Science & Engineering 21 (4) (2019) 7–21. doi:10.1109/MCSE.2019.2906593.
[36] M. Rocklin, et al., Dask: Parallel computation with blocked algorithms and task scheduling., in: SciPy, 2015, pp. 126–132.
[37] E. Suchyta, S. Klasky, N. Podhorszki, M. Wolf, A. Adesoji, C. Chang, J. Choi, P. Davis, J. Dominski, S. Ethier, I. Foster, K. Germaschewski, B. Geveci, C. Harris, K. Huck, Q. Liu, J. Logan, K. Mehta, G. Merlo, S. Moore, T. Munson, M. Parashar, D. Pugmire, M. Shephard, C. Smith, P. Subedi, L. Wan, R. Wang, S. Zhang, The Exascale Framework for High Fidelity coupled Simulations (EFFIS): Enabling Whole Device Modeling in Fusion Science, International Journal of High Performance Computing and Applications 36 (1) (2022) 106–128.
[38] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, K. Persson, FireWorks: a Dynamic Workflow System Designed for High-Throughput Applications, Concurrency and Computation: Practice and Experience 27 (17) (2015) 5037–5059. doi:10.1002/cpe.3505.
[39] J. Goecks, A. Nekrutenko, J. Taylor, G. T. team@ galaxyproject. org, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences, Genome biology 11 (2010) 1–13.
[40] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, K. Chard, Funcx: A federated function serving fabric for science, in: Proceedings of the 29th International symposium on high-performance parallel and distributed computing, 2020, pp. 65–76.
[41] B. Balis, HyperFlow: A Model of Computation, Programming Approach and Enactment Engine for Complex Distributed Workflows, Future Generation Computer Systems 55 (2016) 147–162. doi:10.1016/j.future.2015.08.015.
[42] M. Albrecht, P. Donnelly, P. Bui, D. Thain, Makeflow: a Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids, in: Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, 2012. doi:10.1145/2443416.2443417.
[43] J. L. Peterson, B. Bay, J. Koning, P. Robinson, J. Semler, J. White, R. Anirudh, K. Athey, P.-T. Bremer, F. Di Natale, D. Fox, J. Gaffney, S. Jacobs, B. Kailkhura, B. Kustowski, S. Langer, B. Spears, J. Thiagarajan, B. Van Essen, J.-S. Yeom, Enabling Machine Learning-Ready HPC Ensembles with Merlin (2021). doi:10.48550/arXiv.1912.02892.
[44] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al., Accelerating the machine learning lifecycle with mlflow., IEEE Data Eng. Bull. 41 (4) (2018) 39–45.
[45] P. Di Tommaso, M. Chatzou, E. Floden, P. Prieto Barja, E. Palumbo, C. Notredame, Nextflow Enables Reproducible Computational Workflows, Nature Biotechnologies 35 (4) (2017) 316–319. doi:10.1038/nbt.3820.
[46] Y. Babuji, A. Woodard, Z. Li, D. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. Wozniak, I. Foster, M. Wilde, K. Chard, Parsl: Pervasive Parallel Programming in Python, in: Proceedings of the 28th ACM International Symposium on High-Performance Parallel and Distributed Computing, 2019, pp. 25–36.
[47] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. Ferreira da Silva, G. Papadimitriou, M. Livny, The Evolution of the Pegasus Workflow Management Software, Computing in Science & Engineering 21 (4) (2019) 22–36. doi:10.1109/MCSE.2019.2919690.
[48] M. Turilli, V. Balasubramanian, A. Merzky, I. Paraskevakos, S. Jha, Middleware Building Blocks for Workflow Systems, Computing in Science & Engineering 21 (4) (2019) 62–75.
[49] F. Mölder, K. Jablonski, L. Brice, M. Hall, C. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. Twardziok, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, Sustainable Data Analysis with Snakemake, F1000Research 10:33 [version 2; peer review: 2 approved] (2021). doi:10.12688/f1000research.29032.2.
[50] I. Colonnelli, B. Cantalupo, I. Merelli, M. Aldinucci, StreamFlow: Cross-Breeding Cloud with HPC, IEEE Transactions on Emerging Topics in Computing 9 (4) (2021) 1723–1737. doi:10.1109/TETC.2020.3019202.
[51] J. Wozniak, T. Armstrong, M. Wilde, D. Katz, E. Lusk, I. Foster, Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing, in: Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013, pp. 95–102.
[52] B. Sly-Delgado, T. S. Phung, C. Thomas, D. Simonetti, A. Hennessee, B. Tovar, D. Thain, TaskVine: Managing In-Cluster Storage for High-Throughput Data Intensive Workflows, in: Proceedings of the 18th Workshop on Workflows in Support of Large-Scale Science, 2023.
[53] J. Vivian, A. A. Rao, F. A. Nothaft, C. Ketchum, J. Armstrong, A. Novak, J. Pfeil, J. Narkizian, A. D. Deran, A. Musselman-Brown, et al., Toil enables reproducible, open source, big biomedical data analyses, Nature biotechnology 35 (4) (2017) 314–316.
[54] F. R. Duro, J. G. Blas, F. Isaila, J. Carretero, J. M. Wozniak, R. Ross, Experimental evaluation of a flexible I/O architecture for accelerating workflow engines in ultrascale environments, Parallel Computing 61 (2017). doi:http://dx.doi.org/10.1016/j.parco.2016.10.003.
[55] J. Ozik, N. T. Collier, J. M. Wozniak, C. Macal, G. An, Extreme-scale dynamic exploration of a distributed agent-based model with the EMEWS framework, IEEE Transactions on Computational Social Systems 5 (3) (2018).
[56] J. M. Wozniak, M. Dorier, R. Ross, T. Shu, T. Kurc, L. Tang, N. Podhorszki, M. Wolf, MPI jobs within MPI jobs: A practical way of enabling task-level fault-tolerance in HPC workflows, Future Generation Computing Systems 101 (2019). doi:https://doi.org/10.1016/j.future.2019.05.020.
[57] M. Hategan-Marandiuc, A. Merzky, N. Collier, K. Maheshwari, J. Ozik, M. Turilli, A. Wilke, J. M. Wozniak, K. Chard, I. Foster, et al., Psi/j:

A portable interface for submitting, monitoring, and managing jobs, in: 2023 IEEE 19th International Conference on e-Science (e-Science), IEEE, 2023, pp. 1–10.

[58] A. Alsaadi, M. Hategan-Marandiuc, K. Maheshwari, A. Merzky, M. Titov, M. Turilli, A. Wilke, J. M. Wozniak, K. Chard, R. F. da Silva, S. Jha, D. Laney, Exascale workflow applications and middleware: An exaworks retrospective (2024). arXiv:2411.10637.
URL https://arxiv.org/abs/2411.10637

[59] P. A. Ewels, A. Peltzer, S. Fillinger, H. Patel, J. Alneberg, A. Wilm, M. U. Garcia, P. Di Tommaso, S. Nahnsen, The nf-core framework for community-curated bioinformatics pipelines, Nature biotechnology 38 (3) (2020) 276–278.

[60] S. Grayson, D. Marinov, D. S. Katz, R. Milewicz, Automatic reproduction of workflows in the snakemake workflow catalog and nf-core registries, in: Proceedings of the 2023 ACM Conference on Reproducibility and Replicability, 2023, pp. 74–84.

[61] O. J. R. Gustafsson, S. R. Wilkinson, F. Bacall, S. Soiland-Reyes, S. Leo, L. Pireddu, S. Owen, N. Juty, J. Fernández, T. Brown, H. Ménager, B. Grüning, S. Capella-Gutierrez, F. Coppens, C. Goble, Workflowhub: a registry for computational workflows, Scientific Data 12 (1) (2025) 837. doi:10.1038/s41597-025-04786-3.
URL https://doi.org/10.1038/s41597-025-04786-3

[62] S. R. Wilkinson, M. Aloqalaa, K. Belhajjame, M. R. Crusoe, B. de Paula Kinoshita, L. Gadelha, D. Garijo, O. J. R. Gustafsson, N. Juty, S. Kanwal, F. Z. Khan, J. Köster, K. P. von Gehlen, L. Pouchard, R. K. Rannow, S. Soiland-Reyes, N. Soranzo, S. Sufi, Z. Sun, B. Vilne, M. A. Wouters, D. Yuen, C. Goble, Applying the FAIR Principles to computational workflows, Scientific Data 12 (1) (2025) 328. doi:10.1038/s41597-025-04451-9.
URL https://doi.org/10.1038/s41597-025-04451-9

[63] [Online] https://aiidateam.github.io/aiida-registry/.

[64] D. Blankenberg, G. Von Kuster, E. Bouvier, D. Baker, E. Afgan, N. Stoler, G. Team, J. Taylor, A. Nekrutenko, Dissemination of scientific software with galaxy toolshed, Genome biology 15 (2014) 1–3.

[65] D. Yuen, L. Cabansay, A. Duncan, G. Luu, G. Hogue, C. Overbeck, N. Perez, W. Shands, D. Steinberg, C. Reid, et al., The dockstore: enhancing a community platform for sharing reproducible and accessible computational protocols, Nucleic Acids Research 49 (W1) (2021) W624–W632.