Scientific Software as Workflows: From Discovery to Distribution

David Woollard[1,2] Nenad Medvidovic[2] Yolanda Gil[2,3] Chris A. Mattmann[1,2]

[1] *NASA Jet Propulsion Laboratory*
4800 Oak Grove Drive, Pasadena, CA 91108

[2] *University of Southern California*
941 W. 37th Place, Los Angeles, CA 90089-0781

[3] *USC Information Sciences Institute*
4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292

Abstract

Scientific workflows – models of computation that capture the orchestration of scientific codes to conduct "in silico" research – are gaining recognition as an attractive alternative to script-based orchestration. Despite growing interest, there are a number of fundamental challenges that researchers developing scientific workflow technologies must address, including developing the underlying "science" of scientific workflows. In this article, we present a broad classification of scientific workflow environments based on three major phases of in-silico research as well as highlight active research projects that illustrate this classification With our tripartite classification, based on the the phases of "in silico" research, scientists will be capable of making more informed decisions regarding the adoption of particular workflow environments.

Keywords

D.2.6 Programming Environments/Construction Tools
D.2.17 Software Construction

1 Introduction

In the last 30 years, science has undergone a radial transformation. In place of test tubes and optics benches, chemists, physicists and experimental scientists in a host of other disciplines are now using computer simulation as a means of discovering and validating new science. This so-called “in silico” experimentation has been fueled by a number of computer science advances including the ability to archive and distribute massive amounts of data and share hardware resources via the Grid [1].

An activity central to “in silico” experimentation is *orchestration* – the assemblage of scientific codes into an executable system with which to experiment. Orchestration is a complex task that involves data management (locating data, reformatting, etc.), managing input parameters for executables, and handling dependencies between processing elements.

Scientific Workflow Models – models of high level scientific tasks as stages of data processing (*workflow stages*) and the data dependencies between these stages – have shown to be a very useful abstraction for scientific orchestration. *Workflow Environments* process these models, mapping stages onto computational (often Gridded) resources and plan the required data movements to satisfy dependencies in the model. This mapping is often referred to as a *workflow instance* or *concrete workflow.* Finally, a *Workflow Engine* steps through the instance, executing the stages.

At NASA’s Jet Propulsion Laboratory, for example, scientists and engineers have developed workflow systems to process data from a number of instruments, satellites and rovers. The recently launched Phoenix mission to Mars and an Earth-observing spectral radiometer mission called the Orbiting Carbon Observatory set to launch in 2008 both use workflow systems to process raw instrument data into scientific products for the external science community.

Despite these early adopter efforts, scientific workflows have not yet reached a large user base. We have found that there are a number of significant challenges in this domain that have yet to be addressed, despite the large number of scientific workflow systems from which to chose [2]. One such challenge is that there is no standard workflow model, nor is there a fundamental “science” of scientific workflows. At a recent National Science Foundation workshop chaired by one of the authors, entitled *”Challenges of Scientific Workflows,”* this problem was cited as a challenge facing workflow researchers today [3].

One manifestation of this challenge which we will explore in this article is that requirements for scientific workflow systems vary significantly between applications, suggesting that a taxonomy of workflow systems based on the scientific research activities that they support is needed. In this article, we show that classifying workflow systems based on phases of “in silico” research to which they are applicable forms a classification useful for the scientist interested in adopting workflow technology.

The discernible phases of “in silico” research – *discovery* (the rapid investigation of a scientific principle in which hypotheses are formed, tested, and iterated on rapidly), *production* (the application of a newly formed scientific principle to large data sets for further validation), and *distribution* (the sharing of resulting data for vetting by the larger scientific community) – each have distinct scientific workflow requirements. By making scientists aware of these requirements, we intend to better inform their decision regarding the adoption of a particular workflow technology.

In addition to this classification, we will present several current research efforts to further our understanding of the science of scientific workflows in each of the three phases of “in silico” research. Not only do these research vignettes highlight salient research in this domain, but they serve to illustrate our tripartite classification for readers.

2 How are Workflows Used?

In order to explore the requirements scientific workflow environments must meet, it is important that we first discuss the role of the workflow in scientific experimentation more explicitly. By framing our workflow requirements discussion with an understanding of scientists' current experimentation practices, we intend to clarify the process of evaluating particular workflow environments for given applications.

2.1 Orchestrating Experiments

Scientists have solved the problem of orchestration via a number of methods, including scientific programming environments such as IDL and Matlab. The predominant orchestration mechanism, however, is the script. Scripting languages such as Perl (and more recently Python) have allowed scientists to perform a number of common orchestration tasks, including:

- specifying overarching process control flow,
- running pre-compiled executables (via command line or run-time bindings), and
- reformatting input and output data.

In common software engineering parlance, these scripts can be considered “glue code” between more well-defined software modules.

There are a number of problems with script-based orchestration, however, that make workflow modeling an attractive alternative. Because the orchestrating script captures the experiment – its setup (in the form of input parameters), its procedure (the control flow, or execution steps), and record of results (formatting and cataloging outputs) – it is integral to the overall scientific effort and is an important artifact in its own right.

Scripts are difficult to maintain and easily obfuscate the original intention of the developer. Lack of inherent structure or design guidelines makes script-based orchestration a largely ad hoc process. Unlike traditional glue code, orchestration in scientific workflows cannot be treated as throw-away!

2.2 A Description of Workflow Environments

Workflow environments are used to develop and process workflow models. Each of the process elements of the workflow model is linked to a executable that forms a *workflow stage*. A workflow engine steps through the workflow model, executing these stages and managing the I/O requirements of each stage as specified by the model. This is akin to the control flow functionality of script-based orchestration.

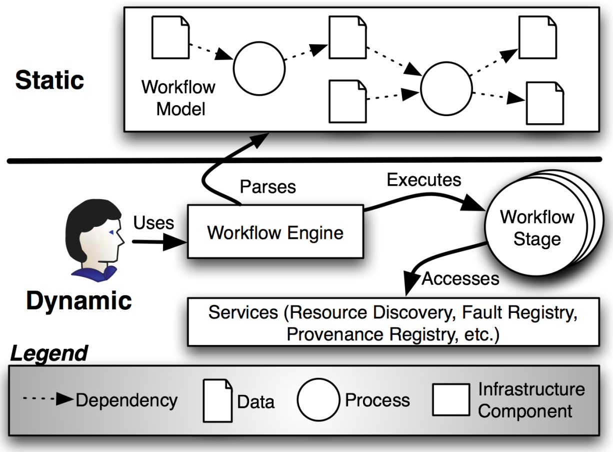


Figure 1: A basic workflow environment.

As illustrated in Figure 1, a workflow environment consist of not only a workflow engine, but also a number of ancillary services. These ancillary services envelop many of the non-control flow aspects of script-based orchestration, including resource discovery for accessing data, fault handling, and data provenance cataloging. While resource discovery services sometimes handle data reformatting issues, it is also common practice to use a preprocessing workflow stage to handle format mismatches.

3 Characterizing Workflow System Requirements

While good taxonomies of workflow environments exist [2, 4], each takes a bottom-up approach, differentiating workflow approaches by the type of model used (e.g., petri nets versus directed acyclic graphs), or by the particular strategy used in computational resource allocation. While this type of characterization is beneficial to the expert audience, we contend that a top-down taxonomy based on *scientific* goals addressed is more useful to the scientist interested in adopting workflow technologies.

3.1 Phases of "in silico" Science

Like all scientific endeavors, *in silico* science has distinctive phases. To take an example from biochemistry, in the early 1920 Macleod and Banting were the first doctors to successfully isolate and extract insulin and are credited with its discovery. Once various efforts to refine their understanding of the structure and properties of insulin were completed, scientists developed a number of techniques for producing insulin in large quantities, eventually settling on genetically engineering synthetic insulin in the late 1970s.

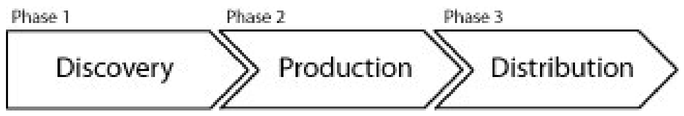


Figure 2: Phases of the "in silico" process.

In Figure 2, we have described three phases of *in silico* science that mostly mirror the processes of *in vivo* and *in vitro* science. These phases are **discovery**, **production** and **distribution**.

Discovery in this context decribes the phase of development in which algorithms and techniques are tested – the scientific "solution-space" is explored – and scientists arrive at a process which yields the desired result. This is what Kepner has described as the lone researcher process [5]. We will describe *production*, akin to the chemical engineering process that was established to synthesize insulin in large quantities, as the engineering and scientific effort to reproduce the process established in the discovery phase on a large scale. Finally, *distribution* is a phase in which results of individual processes are shared, validated, and new research goals are formulated.

We will now discuss each of these phases in greater detail, describing the discernible characteristics of each type of workflow environment as well as the high-level requirements they support.

3.2 Discovery Workflow Characteristics

Discovery workflows are rapidly re-parameterized, allowing the scientist to explore alternatives quickly in order to iterate over the experiment until hypotheses are validated. Discovery workflow environments support this type of dynamic experimentation.

The high-level requirements that a discovery workflow environment should satisfy include aiding scientists in the formulation of abstract workflow models. Discovery workflow environments should transform the abstract models into workflow instances.

3.3 Production Workflow Characteristics

As in the case of producing vast quantities of insulin, production workflow environments are focused on repeatability. These environments should be capable of staging remote, high volume data sets, cataloging results, and logging or handling faults. Unlike discovery workflows, production workflows must incorporate substantial data management facilities. Scientists using production workflow environments care less about the means of abstract workflow representation than they do about the ability to automatically reproduce an experiment.

The high-level requirements of a production workflow environment include handling the non-orchestration aspects of workflow formation such as data resource discovery and data provenance recording. Additionally, production workflow environments should aid the scientist in converting existing scientific executables into workflow stages, including providing means of accessing ancillary workflow services.

3.4 Distribution Workflow Characteristics

Unlike both discovery and production workflow services, distribution workflow environments focus on the retrieval of data. Distribution workflows are used to combine and reformat data sets and deliver these data sets to remote scientists for further analysis.

The requirements for distribution workflow environments include support for rapidly specifying abstract workflows (often focusing on graphical techniques for workflow specification), and for remotely executing the resulting abstract workflows.

3.5 Choosing from the Workflow System Spectrum

It is important to note that many existing workflow systems share the traits of multiple classes of scientific workflow as we have presented them. Additionally, multiple scientific applications the authors have studied fall into more than one of these categories.

While both discovery and distribution workflows must aid the scientist in developing abstract workflows, the emphasis of distribution workflows is on the act of specification rather on the dynamism of the resulting workflow. Likewise, production workflow environments and discovery environments both manage data, although production environments must do so with much greater autonomy in order to process large data sets.

Rather than present a clean taxonomy for its own sake, our principal goal in this work is to aid scientists in understanding their own scientific workflow requirements, showing how this approach can them help them to chose a workflow environment that caters to their scientific goals.

4 Current Workflow Research

In order to illustrate this classification, we will present several current research projects being conducted by the authors. Our intent in this section is two-fold: In showing how each of these projects addresses the high-level requirements of a particular class of workflow application, we will not only (1) highlight salient research topics in the area, but we will also (2) illustrate our classification with real-world workflow environments.

4.1 Workflow Discovery: Wings

Workflow discovery is an exploration activity for scientists, whether systematically trying alternatives or haphazardly looking for a surprising result. The discovery of useful workflows is accomplished by trying different combinations and configurations of components as well as using new datasets or new components [4]. An environment for workflow discovery must:

1. assist users in finding components, workflows, and datasets based on desired characteristics or functions;
2. validate the newly created workflows with respect to requirements and constraints of both components and datasets; and
3. facilitate the evolution and versioning of previously created workflows

Wings [6, 7] is an example of such a workflow discovery system. Wings represents workflows using semantic metadata properties of both components and datasets, represented in Web Ontology Language (OWL). Wings uses (1) workflow representations that are expressed in a manner that is independent of the execution environment and (2) the Pegasus mapping and execution system that submits and monitors workflow executions [6]. In addition, Wings has constructs that express in a compact manner the parallel execution of components to process concurrently subsets of a given dataset [7]. Codes are encapsulated so that any execution requirements (such as target architectures or software library dependencies) as well as input/output dataset requirements and properties are explicitly stated. Code parameters that can be used to configure the codes (for scientists these may correspond to different models in the experiment) are also represented explicitly and recorded within the workflow representation. A model of each code is created to express all such requirements, so they can be used flexibly by the workflow system as workflow components. Component classes are created to capture common properties of alternative codes and their configurations. The methodology for designing workflows, encapsulating components, and formalizing metadata properties is outlined in [8].

Figure 3 shows how Wings represents workflows in data-independent and execution-independent structures. For example, parallel computations over datasets, sketched on the top left, are represented compactly in the Wings workflow on the right. The component representations shown on the bottom left express whether they can process collections or data (e.g., component C-many) or individual datasets (e.g., component C-one) as input. Wings exploits these constraints and represents the parallel computations as a collection of components (e.g., node NC1) that are expanded dynamically into individual executable jobs depending on the size of the dataset bound to the input variables (e.g., to the variable Coll-G). For each of the new data products (e.g., those bound to Coll-Z), Wings generates metadata descriptions based on the metadata of the original datasets and the models of the components.

Using these high-level and semantically rich representations, Wings can reason about component and dataset properties to assist the user in composing and validating workflows. Wings also uses these representations to generate the details that Pegasus needs to map the workflows to the execution environment, and to generate metadata descriptions of new datasets that result from workflow execution and that help track provenance of new data products [6].

To facilitate the evolution of workflows, Wings expects each workflow to have its own namespace and ontology definitions. All workflows are represented in OWL, and follow conventions of web markup languages in importing ontologies and namespaces. Each ontology and namespace has a unique identifier and is never modified or deleted; new versions are given new identifiers. Related workflows can import shared ontology definitions and refer to common namespaces. Current research is exploring extensions to these capabilities for more manageable version tracking, particularly in collaborative settings.

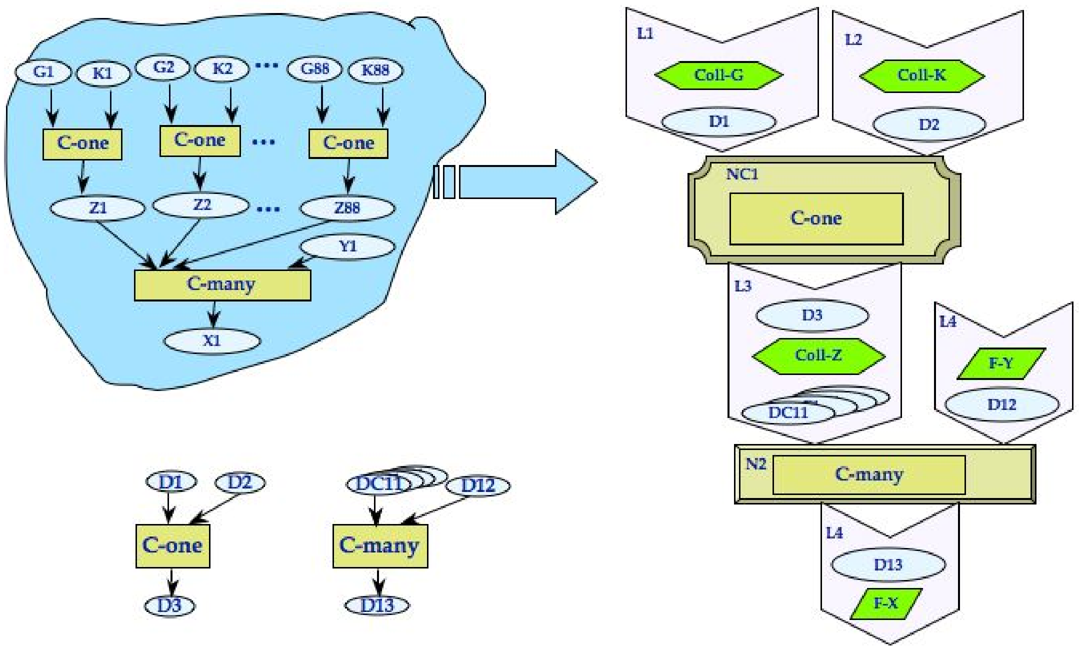
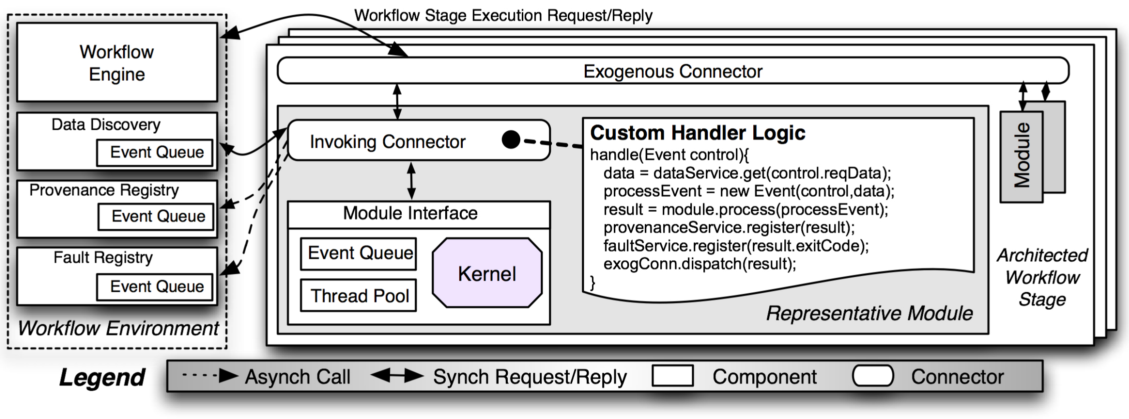


Figure 3: Wings represents workflows in data-independent and execution-independent structures.



```
handle(Event control){
  data = dataService.get(control.reqData);
  processEvent = new Event(control,data);
  result = module.process(processEvent);
  provenanceService.register(result);
  faultService.register(result.exitCode);
  exogConn.dispatch(result);
}
```

Figure 4: An architecture for scientific workflow stages including connectors to workflow services.

4.2 Production Workflow Environment: SWSA

Production workflow environments, as previously stated, incorporate significant data management technology in order to reproduce scientific workflows on very large data sets. Production environments must not only handle the scientific requirements of workflow systems but must also manage the significant engineering challenges of automatic processing of large data sets.

Integration of existing scientific codes is a more significant challenge in production workflow environments than in other workflow environments due to the demands of automatic processing. Locating remote data sets, handling faults appropriately, cataloging large volumes of data produced by the system, and managing complex resource mappings of workflow stages onto Grid and MultiGrid environments require scientific workflow stages to access a host of ancillary workflow services.

Scientific Workflow Software Architecture, of SWSA, is a current research effort at JPL and USC that aims to provide a software architecture for workflow stages that clearly delineates scientific code from the *engineering* aspects of the workflow stage. SWSA provides a componentization of the scientific algorithm, separating out workflow service access to specialized software connectors. This separation of concerns allows both scientists and software engineers to converse about workflow stage design without requiring that each domain practitioner become expert in both scientific and engineering fields [9].

SWSA uses a decomposition of an existing scientific code into scientific kernels. Kernels, like code kernels in high performance computing, are code snippets that implement a single scientific concept. In a graph theoretic sense, a kernel is a portion of the call dominance tree for the source code's basic blocks that has a single source and a single sink – it has internal scope and single entry and exit points. In addition to the scientific kernels, control/data flow of the original source code in the form of a graph of calls made to execute each kernel (its control), and the data dependancies between each kernel (its data-flow) must also be generated. The current SWSA approach requires manual decomposition, though semi-automatic approaches based of software architectural recovery are being explored.

In the second step in the architecting process, SWSA wraps these kernels in a component interface, creating *Modules* as seen in Figure 4. The control flow/data flow of the original program is implemented in a hierarchically-composed exogenous connector as used in [10]. Finally, calls to ancillary workflow services are then made by an invoking connector. An engineer can manage the engineering requirements of the scientific workflow via custom handlers that access services such as data discovery, data provenance registries, and fault registries.

4.3 Distribution Workflow Environment: OODT

Once a production workflow has generated the necessary scientific information, that information needs to be appropriately disseminated to the scientific community to effectively communicate the results. As has been noted previously [11], as the study sample increases, so does the chance of discovery. In the Internet age, scientists have begun to lean on Internet technologies and large scale data movement capabilities to take advantage of this principle, sharing their raw science data and analyses with colleagues throughout the world.

Distribution workflows are those that enable scientific results to be spread to the science community, leveraging data movement technologies, and (re-)packaging technologies to move data to science users. This process is underpinned by distribution scenarios, which specify important properties of a data distribution (e.g., the *total volume* to be delivered, the number of *delivery intervals*, the *number of users* to send data to, etc.). As shown in Figure 5, distribution workflows typically have four distinct workflow stages:

Data Access – The scientific information produced by a production workflow must be accessed (e.g., from a database management system, a file system, or some other repository) before it can be disseminated.

Data Sub-setting – Once accessed, data may need to be (re-)packaged, or subsetted, before being delivered to its external science customers.

Movement Technology Selection – Before data distribution/dissemination, an appropriate data movement technology (e.g., FTP, HTTP, Bittorrent) must be selected.

Data Distribution – Once all prior stages are complete, the data can be transferred to the science community.

To illustrate the complexity of a distribution workflow, consider this representative use case:

> More than 100 gigabytes (GB) of Planetary Data System (PDS) SPICE kernel datasets (that describe planetary mission navigation information) and PDS data sets need to be subsetted and sent across the wide-area network (WAN) from the PDS Engineering Node at NASA's Jet Propulsion Laboratory (JPL) to the European Space Agency (ESA) and from JPL across the local-area network (LAN) to the (local) PDS navigation (NAIF) node. The data must be delivered securely (using port-based, firewall-pass through) to over 100 different engineering analysts, and project managers, totaling ten distinct types of users. The consensus amongst the users is that they would like the data delivered in no more than 1000 delivery intervals, and due to the underlying network capacities of the dedicated LAN to the NAIF and WAN to ESA, the data must be delivered in 1MB to 1 GB chunks during each interval. The data must reach its destination quickly in both cases (due to the sensitivity of the information, as well as the urgency with which it is needed), so scalability and efficiency are the most desired properties of the transfer.

Clearly, such scenarios involve the dissemination of large amounts of information from data *producers* (e.g., the PDS Engineering Node), to data *consumers* (e.g., ESA). There are a number of emerging large-scale data dissemination technologies available that purport to transfer data from producers to consumers in an efficient fashion, and to satisfy requirements such as those induced by our example scenario.

In our experience however, some technologies are more amenable to different classes of distribution scenarios than others: for example, Grid [1] technologies (such as GridFTP) are particularly well suited to handle fast, reliable, highly-parallel data transfer over the public Internet; however, this benefit comes at the expense of running heavyweight infrastructure (e.g., security trust authorities, Web servers, metadata catalog services, etc.). On the other hand, peer-to-peer technologies, such as Bittorrent, are inherently more light-weight and as fast as grid technologies, but at the expense of reliability and dependability. Distribution workflows must be able to decide, either with user feedback or autonomously, the appropriate data movement technology, and dissemination pattern (e.g., peer-to-peer, client/server) to employ in order to satisfy use cases such as the PDS data movement problem described above.

Our recent work has dealt with understanding how to construct distribution workflows that trade the above use cases, and technology choices, in the context of the Object Oriented Data Technology (OODT) data distribution framework [12], at JPL. OODT provides services for data (re-)packaging, subsetting and delivery of large amounts of information, across heterogeneous organizational structures, to users across the world. In addition to OODT, our recent work at the USC has led to the construction of the Data-Intensive

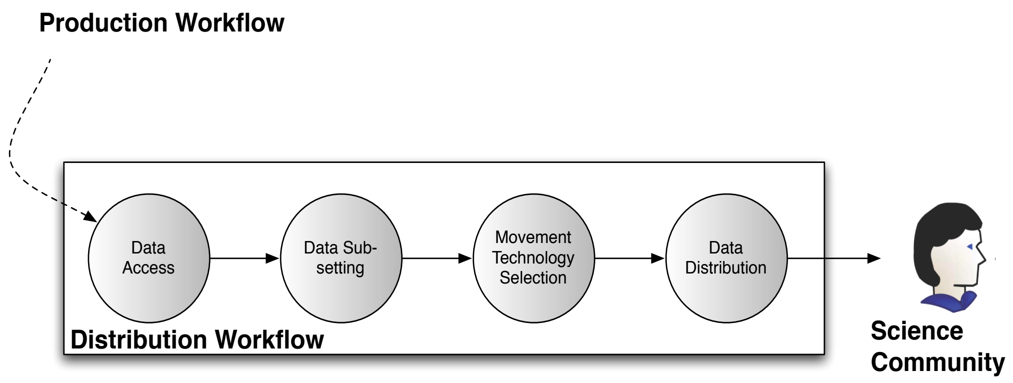


Figure 5: Distribution Workflow Example.

Software COnnectors (DISCO) decision-making framework [13], a software extension built on top of OODT that uses architectural information and data distribution metadata to autonomously decide the appropriate data movement technology to employ for a distribution scenario, or class of scenarios. In our experience, the combination of a data movement infrastructure, and a decision-making framework that is able to choose the underlying dissemination pattern and technology to satisfy scenario requirements are essential to successfully implement and design distribution workflows.

5 Summary and Future Work

In this article, we have presented a classification of scientific workflow environments based on the type of scientific workflow they support. This is a top-down classification based on the type of scientific effort the user is interested in accomplishing, in contrast to existing bottom-up approaches.

In our future work, we plan to continue fundamental research into the “science” of scientific workflow technology. This includes developing more rigorous workflow models that can help to bridge between discovery and production workflow environments.

Additionally, we continue to develop approaches that will ease adoption of workflow technology, including improved integration of legacy scientific codes, better methodologies for adopting workflow modeling over script-based orchestration, and decision making frameworks for delivering resulting data products to the external scientific community.

Acknowledgments

We gratefully acknowledge support from the National Science Foundation under grant CCF-0725332. Additionally, a portion of the research referenced in this article was conducted at the Jet Propulsion Laboratory managed by the California Institute of Technology under a contract with the National Aeronautics and Space Administration.

References

[1] I. Foster et al. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[2] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *Special Issue on Scientific Workflows, ACM SIGMOD Record*, 34(3), 2005.

[3] Y. Gil et al. Examining the challenges of scientific workflows. *IEEE Computer*, 40(12), 2007.

[4] Y. Gil. Workflow composition. In D. Gannon, E. Deelman, M. Shields, and I. Taylor, editors, *Workflows for e-Science*. Springer Verlag, 2006.

[5] J. Kepner. Hpc productivity: An overarching view. In J. Kepner, editor, *IJHPCA Special Issue on HPC Productivity*, volume 18, 2003.

[6] J. Kim et al. Provenance trails in the wings/pegasus workflow system. *To appear in Concurrency and Computation: Practice and Experience, Special Issue on the First Provenance Challenge*, 2007.

[7] Y. Gil et al. Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI)*, July 2007.

[8] Y. Gil, P. A. Gonzalez-Calero, and E. Deelman. On the black art of designing computational workflows. In *Proceedings of the Second Workshop on Workflows in Support of Large-Scale Science (WORKS'07), in conjunction with the IEEE International Symposium on High Performance Distributed Computing*, June 2007.

[9] D. Woollard. Supporting in silico experimentation via software architecture. Submitted to the *30th International Conference on Software Engineering Doctoral Symposium (ICSE08)*, May 2008.

[10] K.-K. Lau, M. Ornaghi, and Z. Wang. A software component model and its preliminary formalisation. In *Proceedings of the 4th International Symposium on Formal Methods for Components and Objects*, pages 1–21. Springer-Verlag, 2006.

[11] D. Crichton et al. A distributed information services architecture to support biomarker discovery in early detection of cancer. In *In Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, page 44, December 2006.

[12] C. Mattmann, D. Crichton, N. Medvidovic, and S. Hughes. A software architecture-based framework for highly distributed and data intensive scientific applications. In *In Proceedings of the 28th International Conference on Software Engineering (ICSE06), Software Engineering Achievements Track*, pages 721–730, May 2006.

[13] C. Mattmann, D. Woollard, N. Medvidovic, and R. Mahjourian. Software connector classification and selection for data-intensive systems. In *Proceedings of the ICSE 2007 Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS)*, May 2007.

Author Biographies

David Woollard is a Cognizant Design Engineer at NASA's Jet Propulsion Laboratory and a 3rd year PhD candidate at the University of Southern California. David's research interests are in software architectural support for scientific computing, specifically support for integration of legacy scientific code into workflow environments. He is a primary developer on JPL's OODT project. He is a member of the IEEE.

Corresponding Address:
NASA Jet Propulsion Laboratory
4800 Oak Grove Drive, Pasadena, CA 91108
e-mail: woollard@jpl.nasa.gov

Nenad Medvidović is an Associate Professor in the Computer Science Department at the University of Southern California. His work focuses on software architecture modeling and analysis; middleware facilities for architectural implementation; product-line architectures; architectural styles; and architecture-level support for software development in highly distributed, mobile, resource constrained, and embedded computing environments. He is a member of ACM and IEEE Computer Society.

Corresponding Address:
University of Southern California
941 W. 37th Place, Los Angeles, CA 90089-0781
e-mail: neno@usc.edu

Yolanda Gil is associate division director for research of the Intelligent Systems Division at the University of Southern California's Information Sciences Institute and research associate professor in the Computer Science Department. Her research interests include intelligent interfaces for knowledge-rich problem solving and scientific workflows in distributed environments.

Corresponding Address:
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292
e-mail: gil@isi.edu

Chris A. Mattmann received the B.S., M.S. and Ph.D. degrees in Computer Science from the University of Southern California in Los Angeles in 2001, 2003 and 2007, respectively. Chris's research interests are primarily in the areas of software architecture and large-scale data-intensive systems. In addition to his research work, Chris is a Member of Key Staff at NASA's Jet Propulsion Laboratory, working in the area of Instrument and Science Data

Systems on various earth science missions and informatics tasks. He is a member of the IEEE.

Corresponding Address:
NASA Jet Propulsion Laboratory
4800 Oak Grove Drive, Pasadena, CA 91108
e-mail: mattmann@jpl.nasa.gov