A Software Architecture for Heterogeneous Engineering Workflow Interoperability & Model Provenance

Andy Gallo, David Hughes, Richard Arthur  
*(GE Research, USA)*

**Abstract**

As the computational workflow landscape becomes increasingly diversified, flexible, and distributed, problems with engineering simulation validation and portable repeatability of results emerge. The lack of interoperability due to variability in the software tooling at nearly every layer of the software computing stack further impedes addressing these challenges, compounding systemic complexity executing engineering workflows such as parametric-sweep ensembles and the orchestration of assorted applications applied in concert. Examples of interoperability barriers include workflow representation, execution and tracking, platform neutral FAIR data and tool management including provenancial reference to the runtimes which created or used them, and security barriers at organizational perimeters. Abandoning abstraction for access to best-of-breed capability and/or comprehensive functionality, single-vendor or explicitly constrained solutions impose limitations to the fully realized and liberal application of modern MBSE tooling.

Based on our industrial experience, we propose a software architecture consisting of three workflow types – in-situ, intra- enterprise or site, and inter-site. Interoperable and digitally threaded workflows can span all three types. For a site – a provider of computing service – to interoperate in a workflow – be it a local user site, or a corporate, government, or cloud-managed computing facility – the site must provide four architectural pillars including security, runtime, data, and resource management. Any commercial, open source, or custom solution which adheres to, or can be made to bend to a published open interface can interoperate in a distributed heterogeneous workflow. This implementation-agnostic view differentiates our approach.

We present the software architecture and provide demonstrated examples of its use across heterogeneous systems for industrial purposes including in-situ simulation steerage, design of experiment-driven workflows with synthetic data from physics-based simulations to populate a training repository, as well as supporting and preparatory software readiness workflows. We will illustrate the general approach of incorporating unique and pre-existing tooling and computing systems into the framework and show how data provenance and model traceability can be maintained throughout the execution process.

# Overview

Heterogeneity in HPC computing systems raises barriers to entry, impeding democratized access to resources, limiting the strategic benefit of diverse human resources applied to real scientific engineering problems. By viewing computing services through the lens of a structured and simplifying software interoperability architecture, distributed workflows can be more readily authored while intrinsically securing and maintaining the provenance – the complete conceptual paper trail or digital thread over critical data – its origins and downstream utility.

The end goal is an interoperable workflow framework which insulates the workflow author or programmer from the details of interacting with the disparate computing facilities which comprise the underlying execution platform in every vertical application of the engineering workflow, and while doing so, maintain the provenancial traceability of execution and data.

In this paper we will first illustrate problems presented by distributed heterogeneous computing systems, and focus on interoperability as a solvable concern. We then discuss the importance of data provenance in model based engineering and argue for its inclusion in a workflow standard model. Next we present the workflow model in three types, from in-situ through to inter-enterprise workflows. Each collaborating enterprise *site* is viewed through a standard interface, which we illustrate. We then turn to implementation, showing our reference software with its interfaces and supporting middleware, and discuss the method for sites to onboard. We finish with a discussion of real world and near future use cases.

# Workflow Complexity & Interoperability

High performance computing (HPC) and other systems tailored for scientific and engineering purposes are individually and collectively in a nearly continuous state of change - in the hardware, operating system, key foundational libraries (e.g. MPI), and the like. Add to that changes in the applications we wish to execute on these systems, and to routine engineering workflows which are themselves applications potentially undergoing their own evolution.

The goal of supporting truly portable and repeatable workflows, executable nearly anywhere, at any time in the future under conditions as they existed at the time of the original workflow execution, with the intent of producing the same result as the original is, at the moment, a pipe dream, and not the topic of this paper. Workflow sustainability, which primarily means repeatability in the face of ongoing system change, includes considerations similar to those found in long-running workflows where the system itself might undergo alteration prior to the conclusion of the entire process. Human resource and organizational knowledge retention and transfer are also important to sustainability.

And while sustainable execution of the workflow over long periods is not our primary concern, neither is workflow representation or mutability the primary topic – there can exist many ways to express the same workflow, to provide varying human interfaces to workflow construction suitable for different purposes and user audiences. There can be a neutral workflow representation suitable for interchange between different custom renderings of the same workflow – as a script, as a directed graph, etc. The Common Workflow Language (CWL) is an attempt to provide that common interchange format using YAML text files, which while potentially useful for interactions between layers in a workflow model and for authoring workflows in a consistent fashion against defined APIs, it is not necessarily the ideal format for all user communities. Some users may prefer a popular programming language like Python with straight calls to the API, or a visual representation, itself interoperable with a standard like CWL [1, 2].

What is of concern here is the ability to write the distributed workflow across heterogeneous systems *at all*, let alone sustainably. The lack of interoperability due to variability in the software tooling at nearly every layer of the software computing stack is a formidable impediment. The barrier to entry of running simulation software on today’s HPC systems is high – from organizational barriers and security perimeters, to data classification constraints and concerns, to the consistent or easy availability of certain base libraries [3], to often significant variations in job scheduling syntax especially in the presence of hardware accelerators [4, 5], to the lack of certain foundational workflow software as described in this paper such as for metadata – data about many kinds of data used and produced directly and indirectly by the workflow – as well as for job chaining, and provenancial tracking throughout the control and data flow.

An additional impediment to workflow interoperability is an often single-minded focus on performance in the scientific and engineering community. Hardware and compute time is expensive, results are valuable, and performance is therefore paramount. An emphasis on performance can come at a cost of other architectural benefits, such as interoperability and standard interfaces. By definition, the existence of a mediating interface layer adds non-zero cost, a cost many performance-centric users are not willing to bear. Thus, emphasis on today’s performance comes at a trade-off to tomorrow’s sustainability.

Distributed and heterogeneous systems increase complexity and raise barriers to entry. As high-performance computing becomes potentially more ubiquitous due to improvements in hardware and conveniences such as cloud computing, to truly drive broad adoption the software interfaces involved will require democratizing and enabling simplification. Lowering barriers to entry reduces an organization’s onboarding and training costs, expanding the range of persons and engineering problems which can be brought into the scope of the computing solution. When the systems in question are funded by the nation, as are the systems produced under the auspices of America’s Exascale Computing Project [6], questions of democratic access can arise – questions which can in many cases be addressed by interfaces tailored to less experienced user communities.

Furthermore, most often engineering applications are not run in isolation but rather include essential pre- and post-processing steps. Thus in a sense, “*the workflow is the app*”. Treating the workflow as an application means considering its programmatic and human interfaces. Heterogeneity in compute and user populations means a single interface is unlikely to cover all systems, users, domains, and scenarios. Thus, we see the need for a specification of that workflow interface while allowing various implementations of that interface to be purpose-built. Loose coupling of defined system interfaces would be an appropriate key feature of an architecture which intends to address broad diversity in usage patterns [7].

Heterogeneity of the computing environment poses problems for the sustainability, repeatability, and portability of the workflow and therefore the reusability, recomposability and trustworthiness of overarching MBSE-enabled design and discovery practices. As noted, complexity raises barriers including those of a social nature. Repeatability is difficult to maintain in the presence of changes over time at every level of the stack – from the hardware to the operating systems to the applications themselves. Containers and emulators bridge some but not all of that gap, as some of the issues are not technical but rather human, economic, and procedural. We intend to focus instead on interoperability – the ability to traceably run applications as part of a workflow across a wide variety of systems at *this* approximate moment in time, and as a key point, doing so while maintaining the provenance of the resulting data. Sustaining these workflows over longer periods of time in the face of the hardware, software, network, and human resource changes described above is out of scope for this work.

# Model Provenance

In model based software engineering, or any applied computational discipline, many different digital entities can be considered data, and von Neumann teaches their equivalency. These might include, but are not limited to: applications, workflows which compose those applications, input and output data which results from execution steps, data-derived models which are evaluated and deployed as surrogates in yet further workflows, and so forth.

Metadata describes the model or data entity itself – not only obvious values such as its creation date, but also arbitrary and domain-specific characteristics. From this data we can determine the provenance for the model – its demonstrable origins – the specific sequence of workflows and workflow steps, with their constituent applications and upstream input and output data, which ultimately produced this specific version of the model.

In a feature-rich computational environment, there will be other entities, some digital and some real-world, which can be described by metadata. For example, the compute resource on which a workflow or workflow step was executed, rollup information about the data center which contains this resource, the human resource record of the user or users involved, etc.

Users of a workflow system might decorate data entities with custom information pertinent to their own use case – i.e. user metadata, separate from system-generated metadata (which is generally centered around “who”, “what”, “where”, “when”, and a bit of “how”). Elusive here is capturing the “why”, which we will discuss later – users may voluntarily provide the “why” (“why did we run this workflow step in this manner to produce this output?”) but humans often find capturing and understanding that data challenging [8].

Another aspect of metadata around workflows, in addition to the metadata about the compute resource used, is metadata about the stateful execution of the workflow – both its potential and its actual. If we track metadata about a prospective *job* – the intended running of a workflow or a step therein – we can potentially match it against metadata for an available set of compute resources and make the best fit. We can also maintain the metadata about the running job, which for example might include a sequence of job status transitions (“pending”, “running”, “finished”, et al). By tracking both metadata about data entities and their creation lineage, as well as metadata about the job itself, we can facilitate both control flow and data flow traceability. Tracking the provenance of a data entity does not necessarily require a formal workflow tool – any suitable metadata repository might be used to store information discernable from the operating system process table, for example. But a comprehensive workflow tool sensitive to data provenance concerns would incorporate such runtime tracking as an implicit and automatic part of the system, below the threshold of the user’s normal need for concern.

If the intent of the workflow is to create a deployable engineering model, metadata about the model can also be back-fitted to include downstream information about the model’s usage. For example, if a model is selected for use from amongst a set of potentially competing models, and it fails as measured by some quality control process to provide the desired high quality outcomes required, the model’s metadata can be updated accordingly to discourage further use under these circumstances.

An existing workflow system might not provide a mechanism to store this metadata, retrieve it, and update it in a version controlled manner – a system based solely on filesystem storage often does not. Thus a software architecture in support of heterogeneous workflows would require the existence of such a metadata store, and provide at least a reference implementation [9].

Such a metadata service within a workflow execution system would be sufficiently flexible to support storage of many kinds of metadata such as those suggested above, but certainly at least able to track metadata about the data, metadata about the execution which used and/or produced the data, and metadata about the workflow itself. This metadata would be managed according to FAIR principles [10, 11, 12].

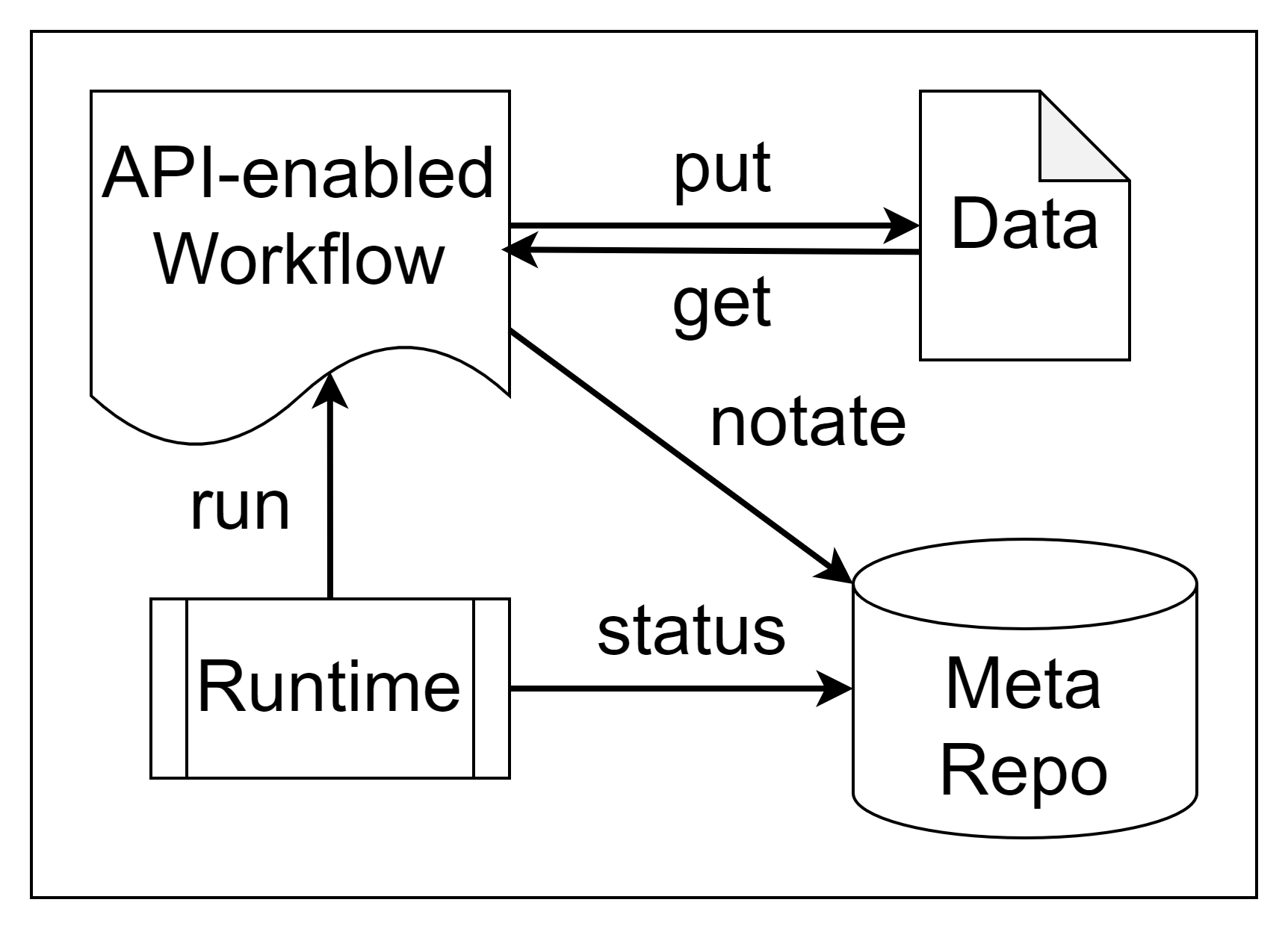


Figure 1: A loosely coupled API-enabled workflow framework which automatically tracks runtime status and metadata about produced and consumed data objects can be used to automatically provide a traceable digital thread for the workflow user.

While data interoperability (e.g. CAD model format) is not in scope for this discussion [13], metadata interoperability is a core consideration. A workflow system which does not natively include a means to track metadata is able to borrow a metadata service with an exposed open API from a reference implementation. Other systems may provide their own metadata storage mechanism, and will do so in their own manner. Other times, the user will employ a metadata scheme at the application level which is appropriate to their scientific or engineering community [14]. An interoperability standard for transporting (import, export) [15] and translating metadata (e.g. localizing the URI of the object being described) between these systems is required. This may be in the form of the metadata as a distinct bundle from the data itself, or, where the metadata is tightly packaged with the data as a single transportable object [16]. More than one interoperability standard might exist, each with best fit to a specific scientific domain or engineering discipline, in more than one schematic format (JSON, XML, etc.). Interoperability here is achieved by line-of-sight to a transformation between a new format and any previously interoperable one.

The purpose of this discussion is not to define that metadata and provenancial interoperability industrial standard here per se – pre-existing consortia are attempting to address that concern – but rather, to simply specify that it exists as part of a well-formed architecture in support of heterogeneous engineering workflows. Now let us turn to the software architecture of which data provenance is an intrinsic part.

# Three Types of Workflows

Workflow as a term has broad definition depending on the domain and can mean a range of activities from human interactive document approvals, to partially automated engineering design processes, to fully automated financial transactions. For the purposes of this paper, we’re primarily focused on scientific and engineering workflows and workloads within the context of a particular design problem being solved. The placement of the solution into a larger organizational or product lifecycle management context is outside the scope, though the concepts could be seen to apply. We’re interested in primarily autonomous workflows, albeit perhaps with some human interaction, for example in providing real time steerage for a running simulation.

Based on industrial design engineering experience, we propose a software architecture consisting of three workflow types – *in-situ*, enterprise or *intra-site*, and *inter-site*.

The first type, in-situ workflows are executed within a running computing allocation. For example, on a shared HPC system, a user is allocated 20 nodes on which they orchestrate a workflow of several parallelizable and serial applications. From the point of view of the job scheduler, or operating system, this is a single job or process, with a single identifier. However, within the allocation, more than one sub-job may be running, each with its own sub-identifier.

In-situ in the HPC context presents an additional performance requirement – workflow functions might operate on a time scale orders of magnitude longer than the running simulation. In implementing software components intended to operate in support of in-situ workflows, asynchronicity likely via message passing is a typical approach, with different network transports utilized depending on the runtime being in-situ or part of a wider enterprise workflow context [17].

The second type of workflow, intra-site, involves multiple distinct jobs run in parallel or serial, independently or in logical sequences. An example is coordinating multiple jobs on an HPC system – the function of the scheduler – “run job 3 when job 1 and 2 complete and free the necessary resources”. In an intra-site workflow we are executing within an enterprise, and there is a single managed pool of job identifiers for the entire site.

The third and final type of workflow is inter-site, those which span enterprises, i.e. span distinct computing systems each with their own job management, and yet, a workflow can be constructed which utilizes more than one of these sites in concert. A cross-site workflow would permit jobs run on one site system to trigger the execution of jobs on another site, and for provenancial data management to be comprehensive over all sites. In order for this to be true, we need to cast the individual sites into some standard model, at minimum to reconcile their individual job state machines into a canonical interoperable form [18]. Even better would be a site and scheduler-neutral abstraction over job execution. The next section of the paper will put forward that site model.

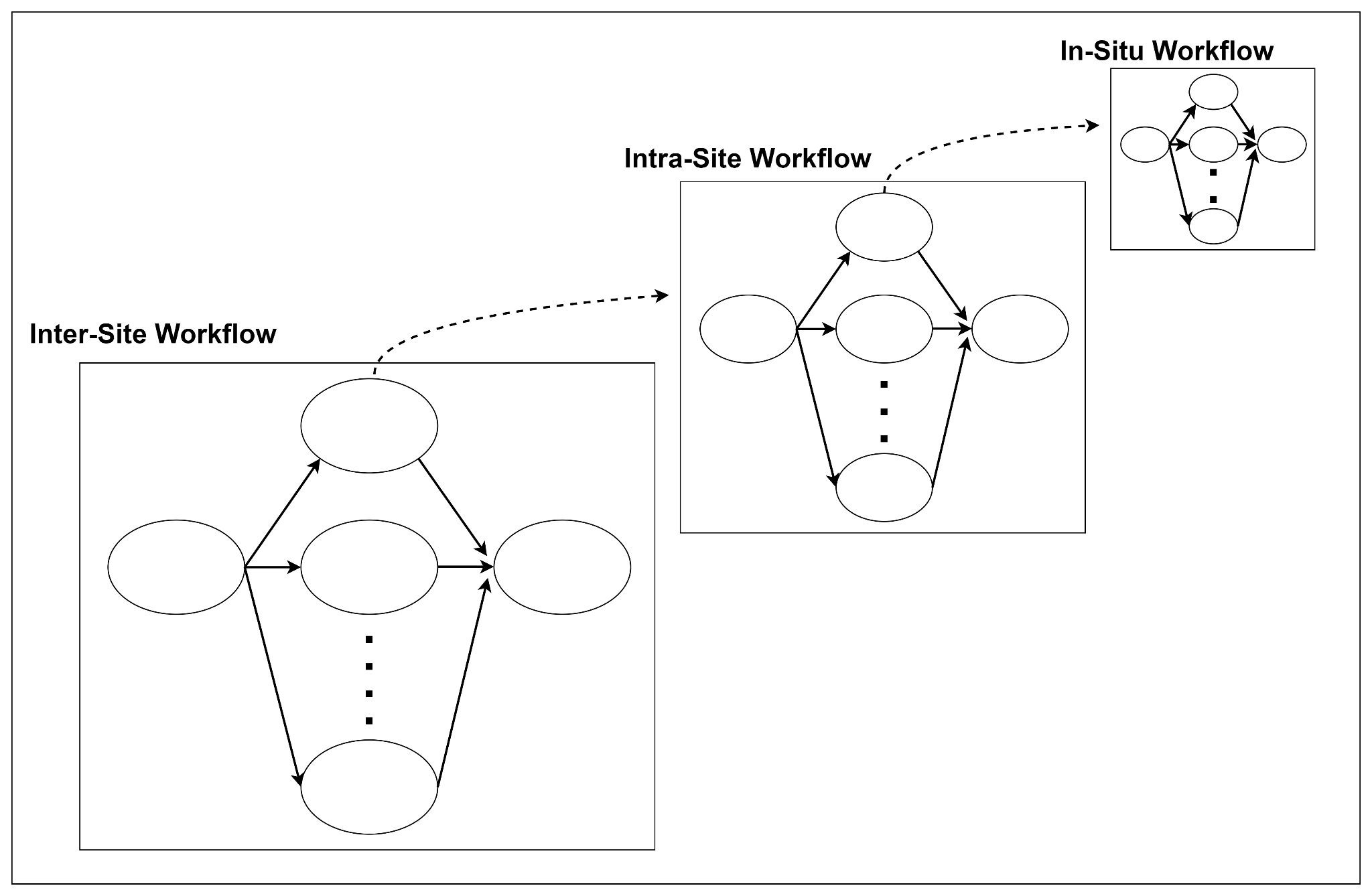


Figure 2: Inter-site workflow invoking an intra-site workflow consisting of multiple jobs, one or more of which may invoke in-situ workflows within a given job allocation. Adherence to a defined site interface and normalization of a site’s job status messages with a canonical set permits interoperability of sites.

In general, the inter-site control flows enabled can be functionally identical to those which are intra-site, except that they may span multiple sites, their site details insulated from the workflow by means of a site framework interface – intra-site workflow tooling is a subset of inter-site tooling, and practically speaking, they are the same.

With job status transformed to an interoperable canonical set, and some standard mechanism for initiating a new job, job chains in an inter-site workflow are enabled, provided the workflow system provides the means for workflows to report their status and gives some means to respond with the issuing of a new job command. For example, if the workflow system maintains a service for the reporting of canonical status, and for the setting of job event handlers to be fired on certain status occurrences, then the service can provide inter-site job chaining, allowing particular workflows to be authored and executed in isolation. Any invocation of a job – either immediately or in a deferred future – is a fork in the workflow. Joins in the control flow are maintained by this event handler service, which exists conceptually in all three workflow types.

A range of conceptual logical arrangements are possible in a fully realized event handler implementation, including:

* Fire handler job when state of monitored job >= some canonical state (can be used for any state, for success or failure or interim state handlers)
* Fire when a set of jobs reaches state (e.g. fire when all upstream jobs have moved to a positive completion status)
* Fire on partial set completion (e.g. when 5 of the 7 upstream jobs complete, fire a another job)
* Fire when satisfied or time T elapses (e.g. fire when all upstream jobs have moved to a positive completion status or T hours have elapsed)
* Fire on a schedule
* Fire on the event recurring N times then evict
* Fire as many times for a given time-to-live, then evict
* Fire when job puts or gets data with metadata M (e.g., “fire when design model reaches approved state”, “fire QC sub-workflow on checkin”)

Interoperable and digitally threaded workflows can span multiple workflow types concurrently. An example is an intra-site workflow which, after performing some pre-processing steps, launches an HPC job which within itself performs an in-situ workflow. Continuing the example, the in-situ workflow can emit discernable events which can trigger workflow steps outside of the HPC job, for example, to deliver interim results, or request interactive human steerage of a running HPC simulation. A later section of this paper will illustrate several enabled workflows.

Today we see tooling which addresses one or two of the above three workflow types, but rarely all three. An example is an HPC scheduler which manages jobs on an HPC cluster and then is itself deployable as an application within the HPC allocation, able to manage sub-jobs within it [19]. Adding inter-site awareness to such a scheduler would complement its functionality. A macro scheduler of this inter-site variety, armed with metadata about available compute sites and capabilities, coupled with metadata about the nature of the candidate job, can perform a best fit of task to compute resource, and scales into the bottom of the technical stack to intra-site and heterogeneous in-situ workflows as well.

# Four Pillars of An Interoperable Site

The wide array of physical engineering domains required to realize actionable insights in complex real world MBSE problems requires an equally wide array of engineering applications, each with its own performance characteristics, domain languages and interfaces. Combined with the diversity of scheduling and pre- and post-processing capabilities commonly available to engineering practitioners, further impetus is lent towards a coherent system able to provide asynchronous and loose coupling for workflows and with functionally-ready integration mechanisms for interoperability and translation between domain languages and system interfaces. The loose coupling approach applies at various scales, including between applications collaborating in an orchestrated workflow, as well as between elements of the same application, for example, in multi-physics co-simulation. In so much as a single application is not likely to elegantly encapsulate the end-to-end engineering process, it follows, as we’ve figuratively said, that “the workflow is the app.”

Therefore, loose coupling necessarily becomes the dominant paradigm for heterogeneous applications. Loose coupling implies contractual interfaces between collaborating components, and thus permits a wide range of implementations – in applications, in workflow constructions and executors, schedulers, and better adapting to growing diversity in the underlying hardware architectures [20]. Asynchronous loose coupling of components is often achieved with message-oriented architectures and event handlers [21].

For a site to interoperate in a loosely coupled inter-site workflow – be it a local user site, or a corporate, government, or cloud-managed computing facility – the site must provide interfaces covering four architectural pillars including security, runtime, data, and resource management, conveniently abbreviated for conversation as *Auth*, *Run*, *Repo*, and *Spin* (a term similarly used by other comparable frameworks such as [22]). Upon these four pillars are layered generic, language-native, and domain-specific interfaces for both automated programmatic workflows as well as human interactions. Multiple interface options means there is flexibility and multiple ways to express a given workflow depending on the needs of the user community. An emphasis on interfaces rather than implementation means these interfaces can be utilized within any other workflow tool which permits programmatic extensibility.

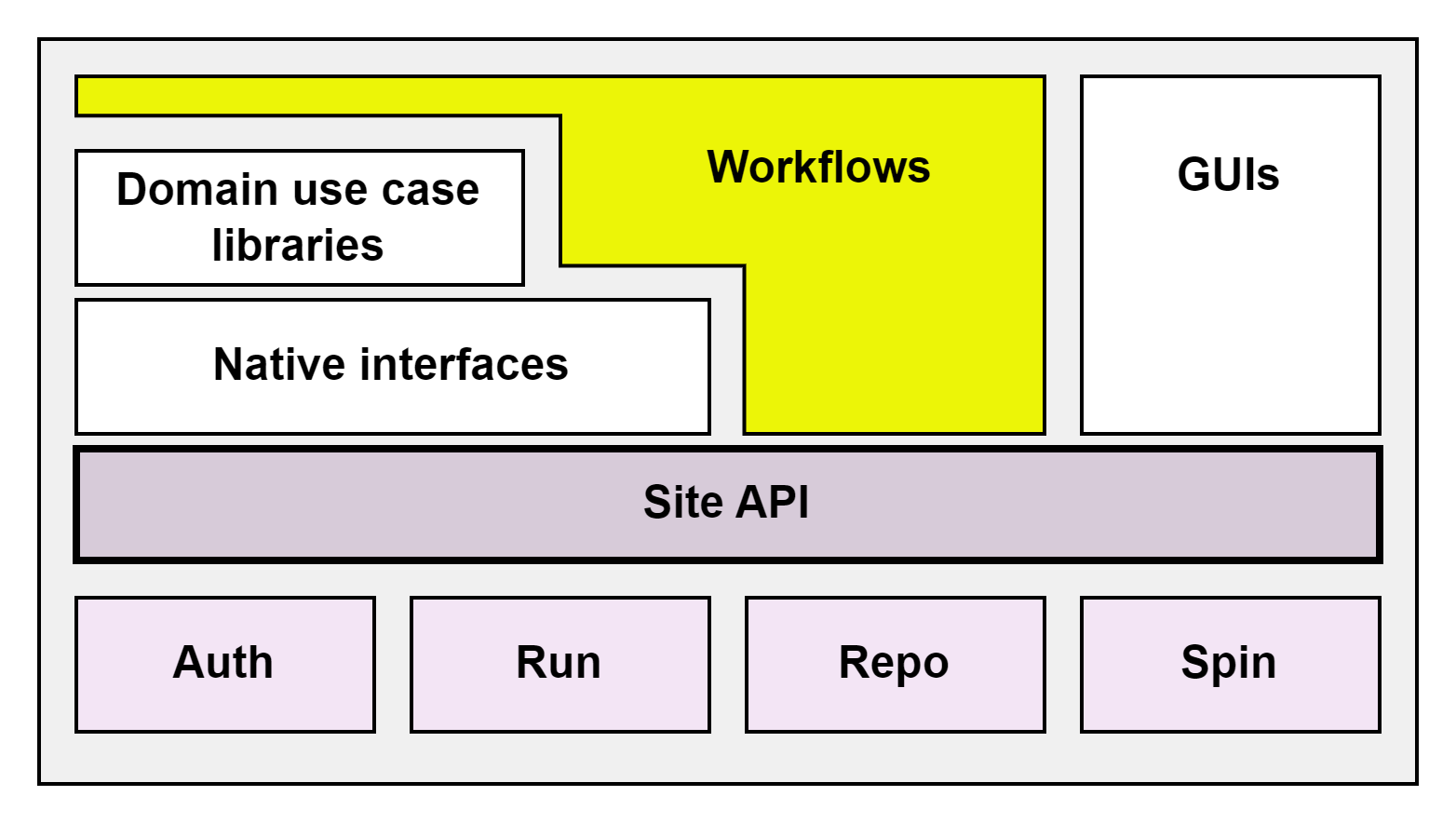


Figure 3: Four-part site API, upon which are layered various human interfaces.

Each of the four site pillars, or interfaces, can be simplistically described by a small set of functional verbs. Auth requires “login”, perhaps a predicate “is logged in” to confirm the validity of the user and its membership in a tenancy. A typical successful Auth interaction might return a secure token which can be used to invoke other API endpoints, such as that used to run jobs.

For its part, Run exposes “run job”, “cancel job” and “get job status” verbs. To this we add “{ set, unset, list } event handler”, which implies an inter-site workflow-enabling middleware component which monitors normalized job status and fires registered event handlers as jobs. For example, an event handler might periodically check to see if certain data as represented by its metadata has appeared in the system, and then initiate a job to process the data.

The Repo interface exposes put, get, notate, and find, which similarly implies the existence of middleware which can respond to the metadata notation and subsequent search request. Actual data blobs, either data or even code, reside in the storage medium of choice (e.g. filesystem, S3, database, HTTP endpoint, container registry, etc.) and are referenced in their location by an address stored in the corresponding *metadata sheet*. Interoperability is further enhanced by defining a standard for metadata exchange between sites.

The Spin subsystem, which is the only non-essential pillar, provides a catalog of computing and other services which can be instantiated, and functions to create/allocate and destroy/free them. By loosely coupling Spin, we avoid the challenge of achieving consensus between the incumbent operators of HPC or cloud aggregator systems and instead focus on the facilitation of access, description of capabilities to be run, and capture of the essential data and metadata required to advance the workflow and overarching design practice. With or without the use of on-demand provisioning, sites may offer more than one computing solution, or *compute type*, for example a CPU node, or a GPU-enabled node, or an entire turnkey HPC system. To address heterogeneity, jobs may be targeted at compute types, though as we will see below, it is entirely the responsibility of the site itself to determine how to implement that targeting – at some level of abstraction sites and compute types are interchangeable concepts and a given data center can be modeled as a single site with multiple compute types, or as multiple sites, whichever is more conceptually convenient.

Interface methods such as “run job” include a generic argument, an arbitrary map of “name=value” pairs which pass through and permit any number of vendor-specific runtime and scheduler features and functions. Abstractions cover some healthy percentage of the use cases with the ability to cover the remainder by way of these custom arguments. This pass-through option in the interface gives the user full access to the most performant system features, albeit with some loss of flexibility for reuse, i.e. interoperability. Thus the user is still in control of where the sweet spot is in terms of performance versus other architectural qualities.

When considering the subsystems of Auth, Run, Repo, and Spin applied to computing services fronted by a suitable API, it's not terribly difficult to see the model applied to other kinds of API-fronted shared services, of the kind increasingly common in large scientific instruments. For example, remote operation of high-end microscopes has been established for some time [23]. In order to accomplish this, there is some type of authentication of the user (Auth), the system is provisioned (Spin), samples are moved in (a physical activity, with its accompanying metadata in the digital world) and resulting images moved out (Repo), and in between the experiment is conducted (Run).

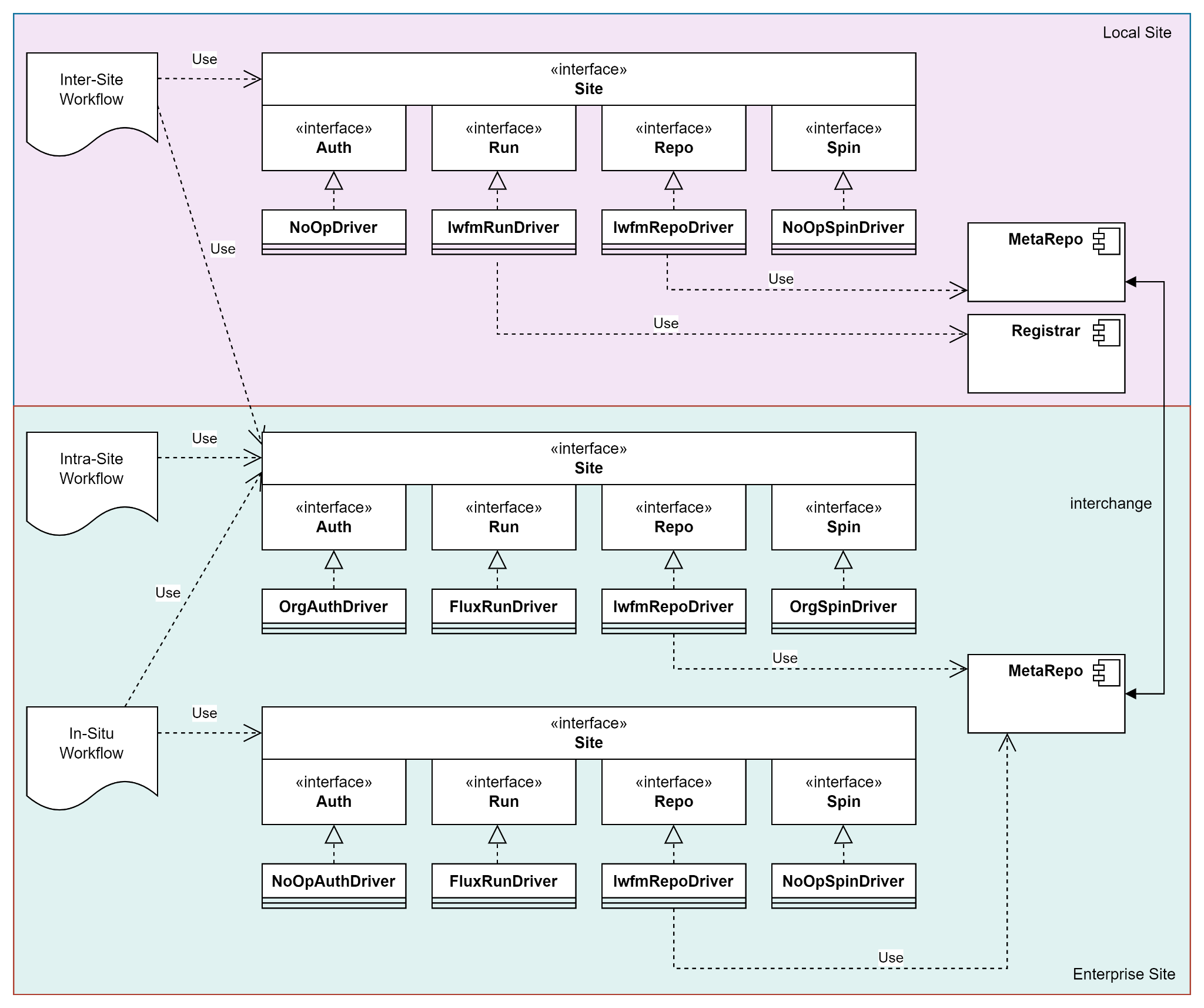


Figure 4: Three workflow types - in-situ, intra-site, and inter-site - each use the standard four-part site interface, each with an appropriately implemented driver. Normalized job definition and status constructs permit inter-site workflows, supported by a minimum of back-end components to track provenancial metadata and job state. Metadata is interoperable.

As the framework is specifically focused on interoperability of site facilities, notably not in scope are:

* Data formats. While metadata interchange is in scope, the data itself is not. Nor is the native representation of the metadata – sites are free to implement the metadata store, use the reference implementation described below, or have none at all.
* Site implementation. Sites are free to be composed of any compute resources, scientific devices, or any other shared resource which for the purpose of workflow authoring can be viewed through the lens of the Auth, Run, Repo, Spin site interfaces. Some compute services will be homogeneous – a single node, a set of common nodes. Some services might be heterogeneous – some nodes with accelerators, some not. The site Run interface permits addressing the service as it is, and can represent the resource as its own site, or as a compute type within it.
* Workflow step portability. In a workflow, a given step, or task in the workflow sequence, is likely to involve running some application on some input to produce some output for potential downstream usage. The running of the application may have complex preconditions – chiefly, that the application exists on the site, or can be made to exist just in time (for example, via a managed build tool like Spack or using containers [24]). In other cases, the application may simply not be available on some sites at this time, nor perhaps at some time in the future. Thus the body of a given workflow step is not guaranteed to be portable – this framework simply aims to make the metadata about the step portable.
* Workflow representation. The framework is API-centric. Thus the base functions – to authenticate, run jobs, put and get data with metadata decoration – can be rendered in a number of different layered formats – as scripts, as structured text, as a directed graph in a GUI, etc.

# Reference Implementation

As an open source project, Local Workflow Manager (lwfm) [25] includes:

* Python declarations of the Auth, Run, Repo, Spin interfaces
* Implementations of these interfaces – i.e. site drivers for several sample systems including a local site runtime and the NERSC Superfacility API, etc. (Other site drivers have been authored for private industrial sites, and these are not shared as open source.)
* The minimum middleware needed to coordinate intra- and inter-site workflows: the MetaRepo, and a Registrar to handle runtime events.
* A simple proof-of-concept GUI which gives runtime visibility into the state of running workflows via the stored metadata, as well as the data associated via navigation of the digital thread maintained by the above middleware.

Various deployment configurations of lwfm are possible:

* With middleware deployed locally, by a user on their own machine, used to coordinate workflows across resources within their own enterprise, including their own local machine(s), with potential to also include jobs running on external sites. Each site, including the enterprise, external sites, and the local site, are represented by site drivers invocable by the user’s workflow which spans all sites.
* Same as above, but with the lwfm middleware also deployed at a remote site. Assuming organizational security barriers at the site perimeter, we assume no connectivity between the lwfm instances. Instead we have access only to the API exposed by the site, wrapped by an lwfm site driver. Thus intra-site workflows which run at the remote site can use the middleware deployed at the site. A comprehensive workflow can include exporting the remote MetaRepo at the end of a set of remote jobs and transferring the information – the data and the metadata – back to the local enterprise via the Repo portion of the site interface. The entire process is thus an inter-site workflow.
* With the lwfm middleware deployed in a managed fashion at the site by the site administrators, providing consistent intra-site workflow functions to all users including ability to author inter-site workflows with access to external sites as well as a user’s own local runtime, while maintaining data provenance throughout.

Many workflow systems are tailored at a specific scientific or engineering domain [26, 27] but lack generality. Some workflow frameworks offer benefits in exchange for pinning to a specific technology stack [28, 29]. Some closely similar approaches which have growing community support lack important elements, most especially over metadata and FAIR data management and digital thread traceability [30], but could be extended to cover those gaps. This architecture provides a comprehensive lens on those systems and product evolutions. For example, a study of similar-to interfaces (e.g. NERSC Superfacility API [31]) suggests, with perhaps the addition of certain administrative functions, that these four pillars are sufficiently comprehensive.

# Onboarding Site Facilities

Any commercial, open source, or custom solution which adheres to, or can be made to bend to a published open interface which addresses these four pillars can interoperate in a distributed heterogeneous workflow. This implementation-agnostic view differentiates our approach. We illustrate the general approach of incorporating unique and pre-existing tooling and computing systems into the framework and show how data provenance and model traceability can be maintained throughout the execution process.

A site which intends to expose its functionality to inter-site workflows needs to provide an implementation of the Auth, Run, Repo, and optionally Spin interfaces. Given the existence of a site API it has been our experience that a site driver can be produced in a mere few hundred lines of Python code, and a reference implementation shows several examples. While a site typically already internally handles authentication, provides a runtime manager or scheduler, and has some kind of data storage even if it's just filesystem, the site and executors of workflows benefit from adoption of a metadata store and event monitoring middleware. Our reference implementation contains both and provides for inter-instance metadata exchange between sites. Run subsystem job identifiers are notated at the time of Repo data put or get, which includes both the apps and their versions as well as the data and its version. The “who”, “what”, “where”, “when”, and “how” of the workflow, along with custom user metadata, are notated in a provenancial log. (A deeper semantic “why”, however, is notoriously elusive and is the subject of future research.)

# Examples of Enabled Use Cases

We present demonstrated examples of the architecture in use across heterogeneous systems for industrial purposes including in-situ simulation steerage, design of experiment-driven workflows with ML training from physics-based simulations, as well as supporting and preparatory software readiness workflows.

* A physics-based application is driven in a design of experiment workflow which explores a potential design space. Each sample is used as input to the physics application producing an output which together are used to train a machine learning surrogate. The training data and its results are tagged with metadata describing the runtime conditions under which they were created. The surrogate is then deployed in lieu of the more expensive physics-based application in its area of expertise.
* An in-situ workflow for CFD analysis includes an in-house C++ app fitted with our in-situ library which is co-launched by an MPMD controller written with the same library in Python. The controller monitors interim results and tests conditions relative to bounds, modifying or halting the simulation accordingly.
* The same in-situ workflow, modified for interactive human steerage, emits interim results to the intra-site interface. The site workflow interface – generated declaratively – presents the results to the user, asks for guidance, and communicates these directions back to the in-situ controller, as shown here:

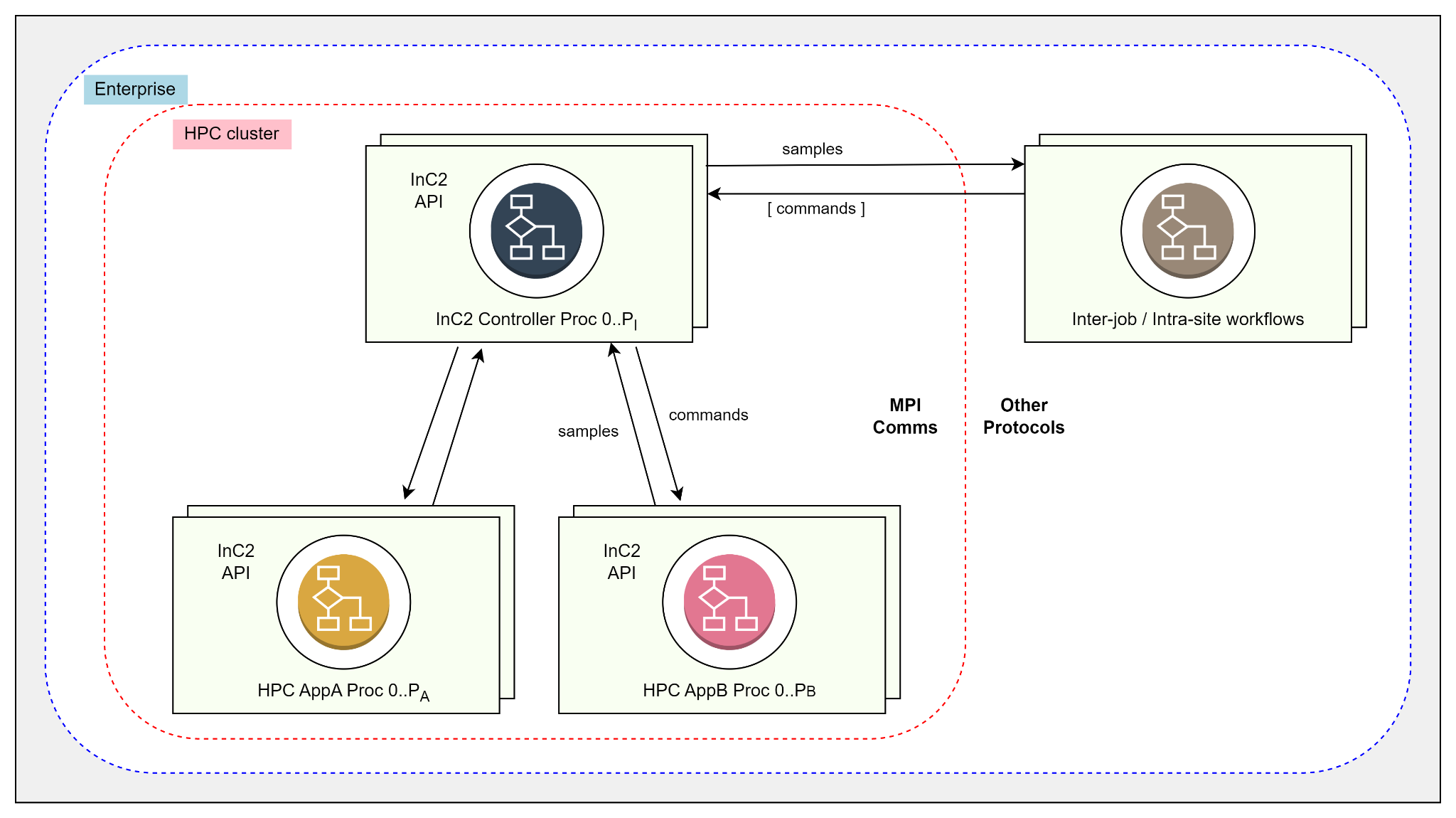


Figure 5: In-situ workflow with intra-site interactive steerage.

* A vendor-provided task-specific workflow tool is used as an application in itself, wrapped by an intra-site workflow script which incorporates it into a broader engineering process context.
* A build and test CI/CD pipeline is constructed as an inter-site workflow, distributing source code changes to target sites, initiating builds, testing and other QC steps, and deployment. Reuse of build artifacts where applicable is provided from site to site. Thus the readiness of M applications is maintained on N platforms, enabling inter-site workflows which use them, as shown below:

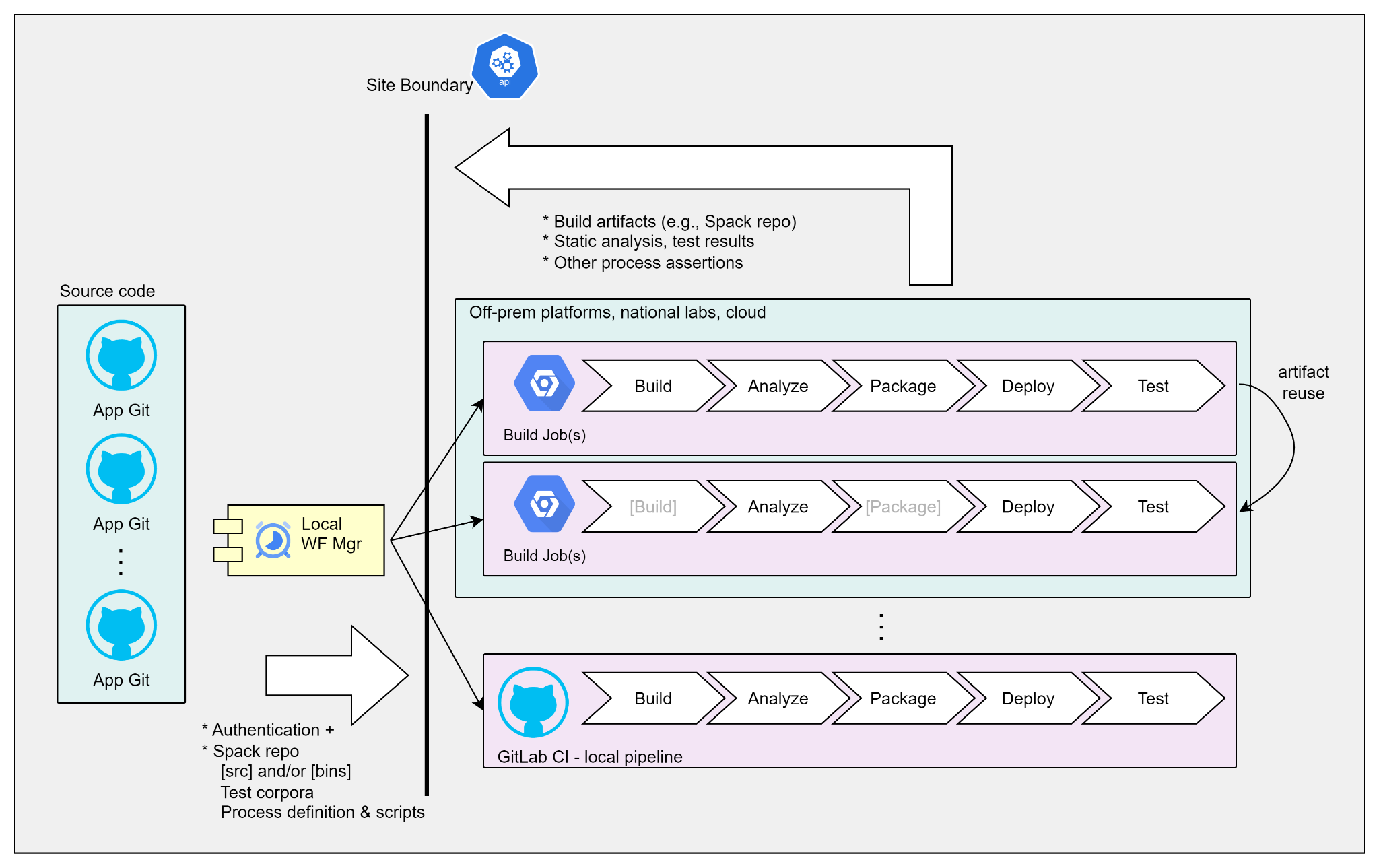


Figure 6: Inter-site workflow maintaining the readiness of M applications on N platforms using commonly available CI/CD tooling.

* Consider a “Surrogate Model University” which consists of a repository of surrogates, each described by its domain, specific competencies, and its level of expertise, which roughly equates to the cost of training the model. Once a model, or ensemble of models, is deployed and selected for use in solving a particular problem, the model metadata can afterwards be updated with some assessment of its performance, leading to better selection of the model for workflows in the future, and/or scheduling the model for a return to training to improve its applicability.
* A framework for authoring heterogeneous workflows, normalizing differences across sites, and providing traceable data management is a simplification of an otherwise complex landscape. Simplification lowers barriers to entry and increases the diversity of the user population which can make practical use of the computing resources, potentially increasing novel applications and benefits. When the resource is a national asset this is especially beneficial. Simplifying interfaces also have benefits in educational contexts and can be useful in workforce development.

# Future Work

Several aspects of the work may continue with collaboration:

* Metadata interoperability standard. The lwfm reference implementation defines an exchange format for metadata between adherent metadata repositories. This is by no means a standard, though as with any structured format, it's likely a structured transform can be constructed to mutate one format into another. Thus it seems all formats are roughly equivalent. The coalescing around a particular standard is a community activity.
* Workflow validation. Guaranteeing the workflow is well-formed, will complete, will execute as intended – a workflow, being a program, shares these considerations with any piece of software. Applying formal best practices for V&V is a topic of future work.
* In a “Surrogate U.” as described above where models are decorated with metadata describing their domain of applicability, assessments of their precision, UQ, records of their use in real world workflow applications, etc., an interoperability standard for assessing [32] and notating those characteristics of the model is needed. This goes beyond the syntax of the metadata interchange to the actual semantic meaning of the model attributes being tracked.
* Similarly, if a dynamic runtime scheduler which is aware of inter-site workflows has access to metadata about the sites available, and access to metadata about the job intended to be run, then optimizations in job allocation across sites can be performed. This suggests the need for a standard format for this site and job metadata, and a reference implementation of an inter-site scheduler which can take advantage of this information.
* Visualizing workflows with dynamic futures. In this framework, workflows can set handlers to fire on workflow events. These events may or may not occur. Event handlers themselves can be created at runtime dynamically, and even cancelled or otherwise evicted. New jobs can be initiated at any time. In addition to the V&V concerns, there is the question of how best to visualize the static and dynamic workflow for the human consumer, again, in a democratized manner.
* Decision provenance. “What are you doing, right now?” “Why did you decide to alter the simulation parameters in this way?” While the architecture described has the capability to manage and make use of many kinds of metadata including for data provenance, and while the metadata which is easy to collect is collected automatically (“who did what action on which data using what application, and when”), the question of “why” is often elusive, but critical in understanding the purpose of a workflow at some time in the future, its scientific or engineering applicability to some problem at hand. Some assistive agent, with today’s technology likely some generative AI, backed by modern language models [33] seems highly useful, and its use by real world researchers and engineering practitioners can be explored. Similarly, the generative technology can be applied to workflow authoring code assist [34].
* Workflows, even interoperable ones, can potentially be long running. A long running workflow raises issues similar to those encountered when studying workflow repeatability and sustainability - the underlying computing system might change between or even during runs.
* Across workflow types, messaging between distributed loosely coupled components is the main paradigm. For in-situ workflows, MPI is common, both within a parallelizable application, and between applications running in the same allocation. Within an intra-site workflow system, the enterprise is free to adopt any message passing transport, for example, a message bus. At the inter-site level, security perimeters come into play, enterprise middleware is not generally accessible, and the messaging formats utilized are appropriate to task – HTTP, REST, and others. When we consider that workflows can cross types – an in-situ workflow might not only communicate with other in-situ elements, but also with enterprise workflow constructs, a burying of the specific messaging formats and transports under a unifying interoperability layer is desired.

# Conclusion

Heterogeneity in the increasingly complex MBSE computing landscape requires simplifying architectures and tooling to increase interoperability, lower barriers to entry, improve productivity for seasoned users, and bring FAIR regularity to data management for engineering and C-suite business considerations. Using the design described here, workflows are supported in-situ and within and across enterprises. Using this four-pillar site framework, models which drive engineering decisions and manufacturing processes are managed like data, decorated with metadata which describes itself and its origins from both control and data flow perspectives. The results are more interoperable workflows across a changing and heterogeneous computing landscape, with tight control of the critical model data which results from these efforts.

# References

[1] C. W. Language (CWL), “Common Workflow Language,” *Common Workflow Language (CWL)*, 2023.<https://www.commonwl.org/>

[2] P. Amstutz, M. Crusoe, and N. Tijani, “Common Workflow Language, v1.0,” *figshare.com*, Jul. 2016, doi:<https://doi.org/10.6084/m9.figshare.3115156.v2>.

[3] T. Gamblin, M. LeGendre, and M. Collette, “The Spack package manager: bringing order to HPC software chaos,” presented at the SC ’15: Proceedings of the International Conference for High Performance Computing, 2015.

[4] “Using Perlmutter - NERSC Documentation,” *docs.nersc.gov*.<https://docs.nersc.gov/systems/perlmutter/>

[5] “Summit User Guide — OLCF User Documentation,” *docs.olcf.ornl.gov*.<https://docs.olcf.ornl.gov/systems/summit_user_guide.html>

[6] “Home Page,” *Exascale Computing Project*.<https://www.exascaleproject.org/>

[7] J. Lewis and M. Fowler, “Microservices: a definition of this new architectural term,” *https://martinfowler.com/articles/microservices.html*, 2014.

[8] S. Ram and J. Liu, “Understanding the Semantics of Data Provenance to Support Active Conceptual Modeling,” presented at the International Workshop on Active Conceptual Modeling of Learning, 2006.

[9] D. Hughes and A. Gallo, “hughed2/MetaRepo2,” *GitHub*, 2023.<https://github.com/hughed2/MetaRepo2>

[10] M. D. Wilkinson *et al.*, “The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, Mar. 2016, doi:<https://doi.org/10.1038/sdata.2016.18>.

[11] M. Barker *et al.*, “Introducing the FAIR Principles for research software,” *Scientific Data*, vol. 9, no. 1, Oct. 2022, doi:<https://doi.org/10.1038/s41597-022-01710-x>.

[12] C. Goble *et al.*, “FAIR Computational Workflows,” *Data Intelligence*, vol. 2, no. 1–2, pp. 108–121, Jan. 2020, doi:<https://doi.org/10.1162/dint_a_00033>.

[13] “Functional Mock-up Interface,” *Fmi-standard.org*, 2023.<https://fmi-standard.org/>

[14] “Inland Electronic Navigational Charts (IENC),” *Data.gov*, 2023.<https://catalog.data.gov/dataset/inland-electronic-navigational-charts-ienc>

[15] “PROV-Overview,” *www.w3.org*, 2013.<https://www.w3.org/TR/prov-overview/>

[16] “Research Object Crate (RO-Crate),” *Research Object Crate (RO-Crate)*.<https://www.researchobject.org/ro-crate/>

[17] “ADIOS 2: The Adaptable Input/Output System version 2 — ADIOS2 2.9.0-rc1 documentation,” *adios2.readthedocs.io*.<https://adios2.readthedocs.io/en/latest/>

[18] J. Chen, Q. Guan, and Z. Zhang, “BeeFlow: A Workflow Management System for In Situ Processing across HPC and Cloud Systems,” presented at the 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC), 2021.

[19] “Home Page,” *Flux Framework*, 2023.<https://flux-framework.org/>

[20] T. Coughlin, “Compute Cambrian Explosion,” *Forbes*, 2019.

[21] M. Fowler, “What do you mean by ‘Event-Driven’?,” *martinfowler.com*, 2017.<https://martinfowler.com/articles/201701-event-driven.html>

[22] “Spin - NERSC Documentation,” *docs.nersc.gov*.<https://docs.nersc.gov/services/spin/>

[23] A. Yamada, “A practical method for the remote control of the scanning electron microscope,” *Journal of Electron Microscopy*, vol. 52, no. >2, pp. 101–109, May 2003, doi:<https://doi.org/10.1093/jmicro/52.2.101>.

[24] K. Vahi, M. Rynge, and G. Papadimitriou, “Custom Execution Environments with Containers in Pegasus-Enabled Scientific Workflows,” presented at the 15th International Conference on eScience, 2019.

[25] A. Gallo and D. Hughes, “lwfm,” *GitHub*, 2023. <https://github.com/agallojr/lwfm>

[26] “KNIME,” *Knime.com*, 2023.<https://www.knime.com/>

[27] “Galaxy Community Hub,” *galaxyproject.org*, 2023.<https://galaxyproject.org/>

[28] “StreamFlow – Towards Cloud-HPC Continuum,” *StreamFlow*, 2023.<https://streamflow.di.unito.it/>

[29] “eFlows4HPC,” *eflows4hpc.readthedocs.io*, 2023.<https://eflows4hpc.readthedocs.io/en/latest/>

[30] “ExaWorks,” *exaworks.org*, 2023.<https://exaworks.org/>

[31] NERSC, “Superfacility API Documentation - NERSC Documentation,” *docs.nersc.gov*, 2023.<https://docs.nersc.gov/services/sfapi/>

[32] R. Arthur, “Model Maturity Web: An (evolving) rubric for gap assessment & road mapping,” *Medium*, Nov. 2020.<https://richardarthur.medium.com/co-design-web-6f37664ac1e1>

[33] OpenAI, “ChatGPT: Optimizing Language Models for Dialogue,” *OpenAI*, 2022.<https://openai.com/blog/chatgpt>

[34] “GitHub Copilot · Your AI pair programmer,” *GitHub*, 2023.<https://github.com/features/copilot>