

# Workflow Requirements

Fermi LQCD Workflow Team  
(Dec. 12, 2006)

## 1 Functional Requirements

### 1.1 *Designing Workflow Templates*

The workflow design system should provide means for designing (modeling) workflow templates in graphical (e.g. Triana or Kepler interface) and/or text mode (e.g. through editing a workflow via workflow languages, such as the AGWL supported by Askalon, or by using XML).

The modeling tools should include provisions for expressing participants and their dependencies, parameterized inputs or actual constraints (such as resources parameters), conditionals, and physics parameter types. User must be able to add annotations and comments to the model.

A way to identify and manage workflow templates is needed. Each template must have a unique identifier. Templates must be able to be composed using other workflow templates and/or workflow instances. For example a template may have the same set of configurations that are staged in by multiple workflow instances.

### 1.2 *Workflow Instantiation*

Workflow instantiation is done by filling in the parameter values and constraints required by the template. The system must be able to verify whether the user supplied values are, for example, of the correct type and within a valid range defined in the template.

The system must be able to validate whether the template can be instantiated.

Instantiations must have a unique identifier generated that represents them. Instantiations that are equivalent may have the same identifier value. What it means for two instantiations to be equivalent needs to be determined.

The system must provide a way to manage workflow instantiations.

### 1.3 *Workflow Execution*

#### 1.3.1 Scheduling

The scheduling system has to perform the following four functions:

Identify the participants that need to run.

Map the participants to particular resources.

Resolve dependencies across system boundaries.

Schedule the execution of the tasks.

The first three should be supported or partially supported by workflow. The last one can be considered as local scheduling (e.g., cluster scheduler, participant scheduler, interactions with dcache and other participants) and thus supported by the participants and not by the workflow system.

The workflow scheduling system should enforce the data and task dependencies of a campaign, i.e. a task only can be executed when its input data become available or when a dependent task completes. In the LQCD system the workflow manager must work closely with the local cluster scheduler (currently PBS/MAUI).

Note: PBS/MAUI may not be aware of the dependencies between participants, under these conditions, the workflow system will have to direct which tasks are available to run.

#### **1.4 Monitor Progress**

The workflow system must

- Track the execution of workflows.
- Be able to record and display (actively and passively) relevant states of all the executing workflows, such as participant and milestone status.

A passive display allows users to query the state of their executing workflows. An active display allows users to register for notifications on completion or specific states of workflow elements.

A separate monitoring facility may be used to provide run time information. Some examples are:

- Participant status
- Ready
- Waiting on a dependency
- Running
- Finished (success or failure)
- Participant time parameters
- Start time
- Elapsed time
- Finish time

#### **1.5 Workflow Execution History**

All workflows submitted for execution must have statistics and states recorded (possibly within a database). The workflow system must be able to collect run-time information from the monitoring system as well as update the overall workflow status. The system must provide information about all the submitted,

executing and completed workflow instances. These historical details may be used for predicting future executions and help to improve performance.

### 1.6 Handle Multiple Workflow Instances

Multiple workflow instances, submitted by multiple users or a single user, must be managed concurrently. The execution of these campaigns may be coordinated (co-scheduled) to optimize the utilization of resources.

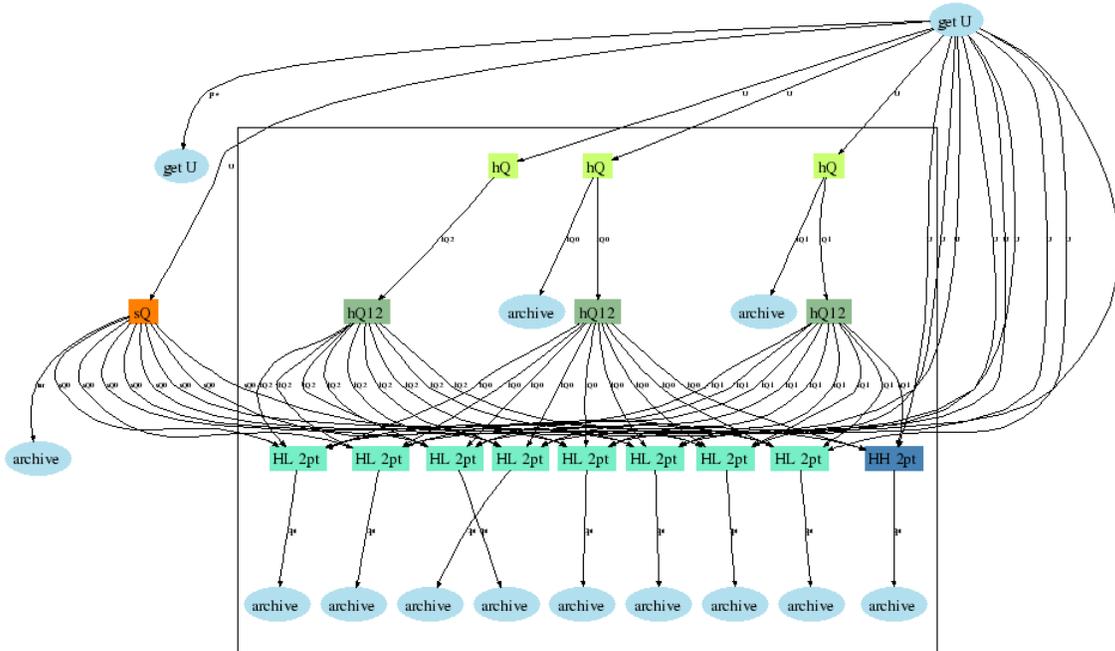
### 1.7 Quality of Service

Various workflow instances may be scheduled according to user specified priorities. The system may attempt to execute a workflow instance within a specified time period, and/or report any deadline that was not or cannot be met.

### 1.8 Staging of Data Files

The workflow system may provide users a method of specifying data that needs to be prefetched. The workflow system is not required to know anything about application I/O, so participants must carry any prefetch action. Prefetched data may be cached by the system for later use. We assume that data consistency is not an issue in the prefetching and caching.

This may imply the existence of different types of participants, identified by the workflow system. The workflow modeling tool may need to define a way to identify participant type e.g. by modeling element or property. A concept such as this may allow for distinguished initial participants to be scheduled differently.



Example of a two point analysis workflow

Most LQCD workflows start from a configuration file that is not available locally. Fetching this file is represented by the 'getU' participant in the example two-point dataflow diagram. Usually there are many instances of similar workflows running in parallel. The system could have these files fetched while the workflows are not running.

### 1.9 Fault Tolerance

The workflow system should interact with the monitoring systems to identify any hardware and software failures and take appropriate action to avoid or mitigate the fault penalty.

Example of simple action: restart/resume from the last milestone.

It is desirable for the system to provide a way for applications (participants) to supply data that is opaque to the workflow system, but tracked (as a blob) while a participant is running. When the restart/recovery action is triggered, the application will be given the last blob that it stored.

### 1.10 Assumptions regarding Intermediate Files (a statement about scope)

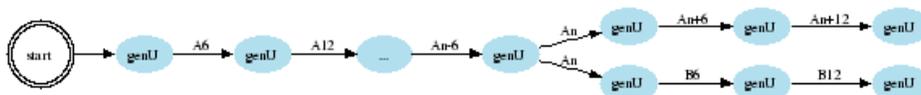
The workflow system must check whether intermediate input files are available before executing a participant. Capabilities of the storage system are not defined as a part of the workflow system. The workflow system is not responsible for managing (e.g. the lifetime of) these files.

### 1.11 Data Provenance

The workflow system must provide a mechanism for managing schemas describing data objects. A schema is a definition of the attributes that describe the contents of a data product from a participant. The workflow system must store records conforming to these schemas (i.e. data product types) for all participant generated data files. In other words, we must be able to store data product description types and data product descriptions of those types.

Users and the workflow system must be able to query the provenance repository to discover facts about the data. The workflow system may use this information to determine whether an intermediate product is already generated, which may avoid rerunning a workflow step.

The provenance repository must be able to track multiple parent relationships. In other words, track the ancestry of output data, assuming that output data are derived from input data, thereby recording each input as a parent.



As an example, suppose that in the configuration generation diagram another chain of participants is added to the fourth 'genU' participant. We have however already executed the original workflow. Supposing that the system kept track of files generated by the previous execution then the new chain of 'genU' participants can start from the point where the latest intermediate file was generated, avoiding rerunning the first  $N$  steps of the original workflow.

Similarly, in the two-point dataflow diagram, the output of the 'sQ' participant was generated using parameters  $A=123$ ,  $B=456$ ,  $C=789$  and a configuration file `xyz.config` fetched by the 'getU' participant. Suppose there is a very similar workflow, which happens to have the same 'sQ' participant, with the same parameters and configuration file. The system could check whether that participant has been invoked with those same parameters and reuse the previous output, in case the output file is still available.

### ***1.12 Campaign Execution***

The workflow system should manage the execution of campaigns. A typical campaign consists of taking an **ensemble** of vacuum gauge **configurations** and using them to create intermediate data products (**propagators**) and output **meson n-point functions** for every configuration in the ensemble. The input ensemble, associated physics parameters and data outputs define the data processing dependence of a campaign.

A user has to define the workflow instance and the system executes it.

A user should be able to interrupt an executing workflow instance, and be able to inform the workflow system to stop executing some participant(s).

A campaign must have persistent state, such that it can be continued or extended from a set of completed workflow instances.

Participant is not required to be fault tolerant.

### ***1.13 Dispatch Campaigns***

The user submits a campaign, in terms of workflow instance(s), to the system and the system executes the workflow instances to completion.

The system should support workflow scheduling and the participants could be remote (data movement, perhaps remote execution of workflows).

A campaign execution consists in running  $N$  independent versions of workflows similar to the two-point dataflow diagram. Each workflow instance has its own parameters and configuration files. There should be some level of control over this group of workflows, such as stopping them all or restarting it with extended workflow with new participants and dependencies.

## 2 Non-Functional Requirement

Availability (source code)

User Friendly Interface – there must be a command line interface and an optional graphical interface. The system must have and help available. Help pages must be easily accessible to the users.

Robustness – failures should be rare; a single failure should not bring the whole system down. The workflow system should be able to recover (e.g. restart from latest workflow system checkpoints and workflow milestones) from all software and hardware failures. The workflow system should query the status of the participants, which are capable of returning this information, when recovering from a failure and update the workflow system's state accordingly.

Extensibility – ability to easily incorporate new functionalities to the workflow system.

Scalability – support thousands of participants and hundreds of workflows, tens of campaigns. Actively track thousands of output files.

Modularity – decoupled components, e.g. underlying monitor system can be modified without changing other workflow management components. Components should preferably run on different machines thus distributing the work load and avoiding single points of failure.

Cross platform – software should be UNIX system compatible -- Linux (JLab and FNAL) and AIX (QCDOC at BNL) are currently used. LQCD may run on other platforms, such as Cray at Pittsburgh and ORNL and Sun at TACC.

## 3 Glossary of Workflow Terms

### 3.1 General Definitions

**Participant:** Participant is an object that transforms inputs into outputs. Example of participants is: dCache (dccp), PBS (qsub), user applications and shell scripts. All Participants are considered to be atomic operations from the executing workflow's point of view.

**Participant Product:** Participant product is the output generated by a participant including provenance data (participant inputs and parameters).

**Workflow:** Workflows are the procedures whereby data and control are passed among participants according to a defined set of rules (e.g. data and control dependencies) to achieve a specific goal.

**Workflow management system:** A workflow management system manages and executes workflows on computing and storage resources. It is responsible for resolving dependencies, keeping track of data products, scheduling and fault tolerance.

**Reliability and Fault Tolerance:** A system is said to *fail* when it cannot meet its promises. An *error* is a part of a system's state that may lead to a failure. The cause of error is called a *fault*. *Reliability* is a measure of the continuous service accomplishment (or, equivalently, of the time to failure) from an initial reference point. *Fault tolerance* means that a system can provide its services even in the presence of faults.

**Quality of Service:** Quality of service is defined as the level of performance in a computing system. For example, to finish a job in a given time limit. A workflow could have different performance constraints, such as time, cost, fidelity, reliability, and security. The primary focus of the LQCD workflow will be *time to complete a participant*.

**Checkpoint:** Checkpointing is obtaining a snapshot of the system's current state that is stored for later use. Checkpointing could be at the system level or at the application level. An application level checkpoint records the present state of a user job (process).

**Restart:** Restart is defined as starting from the originally defined starting point.

**Resume:** Resume is defined as starting from the last checkpoint (or last milestone in terms of LQCD computation).

### 3.2 Workflow Types Definitions

**Template:** The workflow template is a pattern or parameterized description of how a particular problem is solved.

**Ideal:** An ideal workflow assumes unlimited resources and it contains only information relevant to solving the problem in the best possible circumstances. In other words, no resource constraints are included.

**Instantiated:** An instantiated workflow is the result of the application of relevant input files and parameters to a template by a user. It could include multiple versions of the participants.

**Workflow Instance** = workflow template + applications + input parameters and data + output data

**Executable:** An executable workflow is produced by applying cluster resource usage policies and constraints to an instantiated workflow by the workflow system. In other words, the workflow graph is transformed from ideal to something that can be run on an existing cluster.

### 3.3 Domain Specific Definitions

**Lattice:** A lattice is a cubic four dimensional space-time grid. Boundary conditions are normally periodic in the spatial directions and (anti)periodic in the time direction. *Sites*, the vertices of the grid, are referred to by coordinates  $(x,y,z,t)$ . There are eight *links* connecting a site to its nearest neighbors. Lattices are typically cubic:  $L_s = L_x = L_y = L_z$ . Some lattice dimensions now in use include  $L_s^3 \times L_t = 16^3 \times 48, 20^3 \times 64, 28^3 \times 96, 40^3 \times 96$  and  $48^3 \times 128$ .

**(Gauge) Configuration:** A gauge configuration is a four dimensional (space-time) snapshot of the *gluon* field. On each *link* of the lattice, the gluon variable is represented by an SU(3) (complex 3 x 3) matrix. Configurations are stored in configuration files containing  $3 \times 3 \times \text{sizeof}(\text{complex}) \times 4 \times L_s^3 \times L_t$  bytes.

**Ensemble:** An ensemble is an ordered collection of gluon configurations sharing the same physics parameters e.g. lattice spacing (or QCD coupling strength), number of sea quarks and sea quark masses. Configurations are generated in a Markov sequence. At each step, the last configuration serves as the starting gluon configuration which is evolved forward in simulation time by Monte Carlo techniques. At regular intervals in Monte Carlo simulation time, gluon configuration snapshots are saved. Configurations within an ensemble are labeled by a unique sequence number or *configuration number* which is typically

the number of steps in simulation time. An ensemble may contain a fork where more than one Monte Carlo evolution sequence was started from the same input configuration. Forks are identified by a unique *series label*.

**Configuration generation:** Configuration generation is a campaign describing the creation of an *ensemble*. Each step of the workflow(s) of this ensemble generates a gauge configuration which depends upon the output gauge configuration of the previous step.

**Quark Propagator:** The quark propagator is the field that describes how a quark propagates or hops from site-to-site of the lattice. A quark is represented by complex matrix at every site of the lattice. *Clover quarks* are represented by a  $4 \times 4$  (spin)  $\times 3 \times 3$  (color) matrix while *staggered quarks* are represented by a  $3 \times 3$  (color) matrix at every lattice site. Quark propagators are the solution of a sparse matrix problem. The sparse matrix has dimension equal to the number of sites on the lattice times the size of the quark matrix. Conjugate gradient techniques are used to solve for quark propagators. Quark propagators are produced (and saved) as intermediate results during a campaign. During a campaign, quark propagators are computed for every configuration in an ensemble for every given set of physics input parameters e.g. quark mass, quark source type.

**Meson 2- and 3-point functions:** Meson 2- and 3-point functions are generated by connecting together quark propagators to form operators that create and annihilate mesons. These n-point functions describe how mesons, particles composed of quarks, propagate on the lattice. Meson n-point functions typically have one or more time coordinates fixed and they are normally summed over the spatial dimensions before they are stored. A meson 2-point function is typically stored as  $L_t$  complex numbers per configuration. A campaign might generate hundreds of distinct meson n-point functions.

**Ensemble average:** Ensemble averages are the physical results that are calculated in a statistical analysis of meson *n-point functions* that have been averaged over every configuration in an ensemble.

**Campaign:** A *campaign* is a coordinated set of calculations aimed at determining a set of specific physics quantities - for example, predicting the mass and decay constant of a specific particle determined by computing *ensemble averaged 2-pt functions*. A typical campaign consists of taking an *ensemble* of vacuum gauge configurations and using them to create intermediate data products (e.g. *quark propagators*) and computing *meson n-point functions* for every configuration in the ensemble. An important feature of such a campaign is that the intermediate calculations done for each configuration are independent of those done for every other configuration.

An example campaign could consist of a single workflow, where the intermediate products are used immediately, or it could be broken up into multiple workflows, with the intermediate products stored for later use.

**Milestone:** A milestone is the persistent state of the last intermediate result reached by a workflow instance. A milestone can be used for recovering a workflow instance or to extend a campaign. A milestone is typically the output of a participant. The milestone has information about the workflow instance that generated it, including parameters and input files that were used (provenance).

**Extend Campaign:** A previously executed campaign can have new input files added, an act that requires further workflow execution. It could also use milestones from a previous campaign and generate additional output files.

**Campaign** = workflow instance(s) = template(s) + ensemble + applications + parameters + desired outputs