LQCD Workflow Review

L.Piccoli

Feb 23 2009

Goals of this review

- The major goal of the LQCD workflow project is to create a system for designing, maintaining, and running LQCD workflows, and provide for the storage and use of their results. There are four things we would like you to concentrate on for this review:
  - How well does the design and prototype implementation fit the requirements?
  - How well does the current system match the configuration generation analysis?
  - What improvements should be made to the code (including design and data model)?
  - What areas are in need of further study, specification, or review

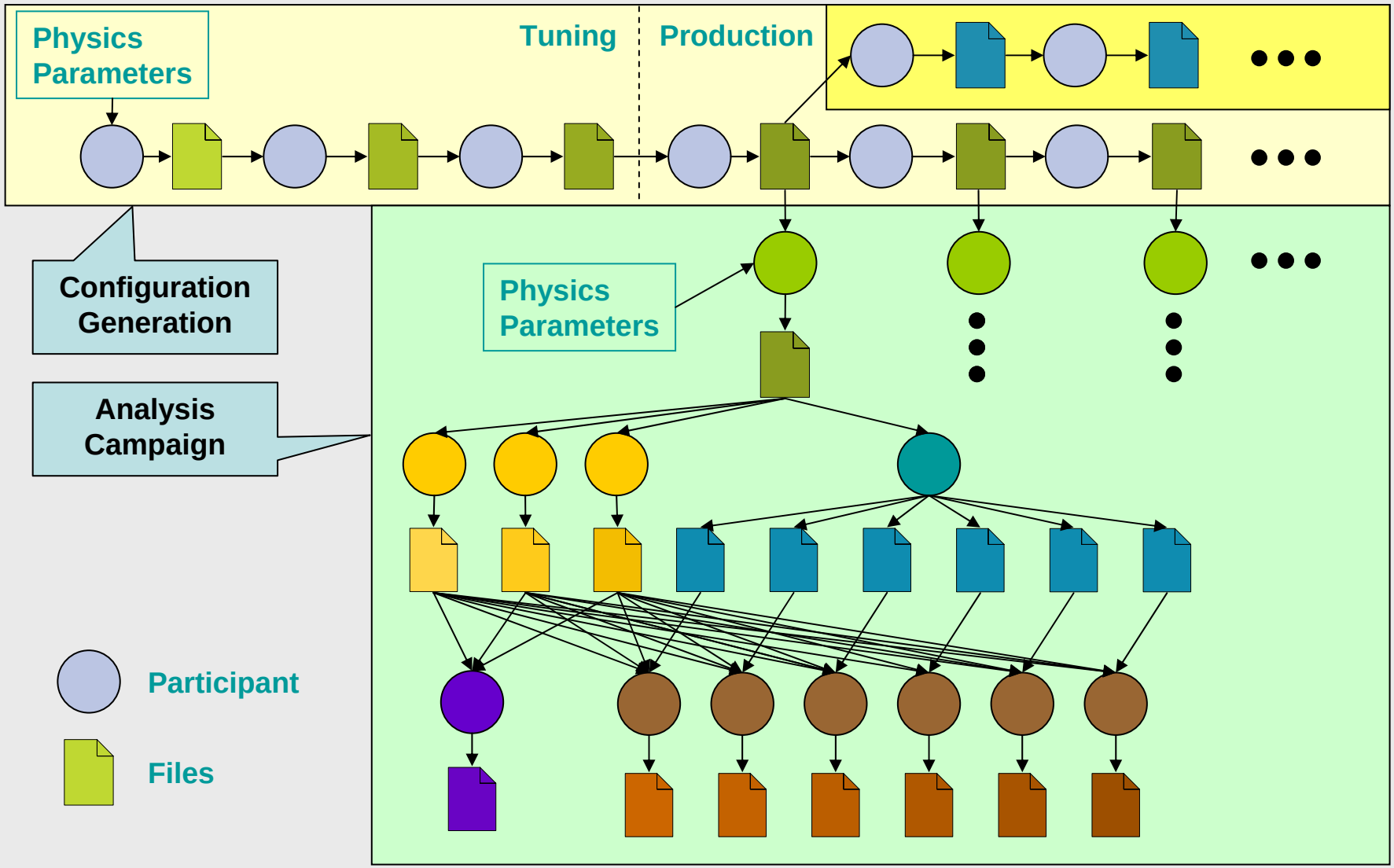
Goals of this review

- Aspects to be considered:
  - Abstractions and concepts
  - How well the package expresses the LQCD configuration generation workflow (and others)
  - Interfacing with legacy applications - Providing wrappers to applications for different workflow engines
  - Parameter set management
  - The design of the package as a whole

Current LQCD workflows

- Parameter control
  - Physics parameters on emails, encoded in file names, script files
- Produced data
  - No rigorous file location control
- Run time
  - Running scripts must be closely watched
  - Hard to debug current scripts, need to look through log files to find possible failure causes
  - Need for a simple and effective workflow language

Typical LQCD workflows

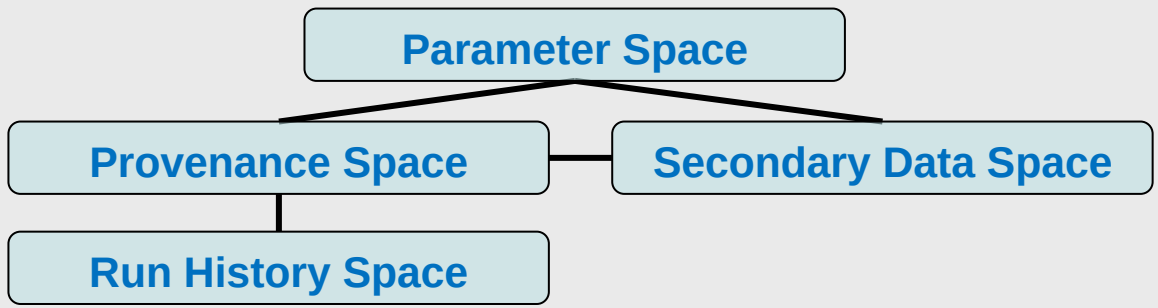


Proposed solution

- Define a data model for LQCD workflows
  - Configuration generation
  - Analysis campaign
- Separate data model from the workflow execution system
  - Enable use of alterative workflow execution systems using the same underlying data representation

Abstractions - data model

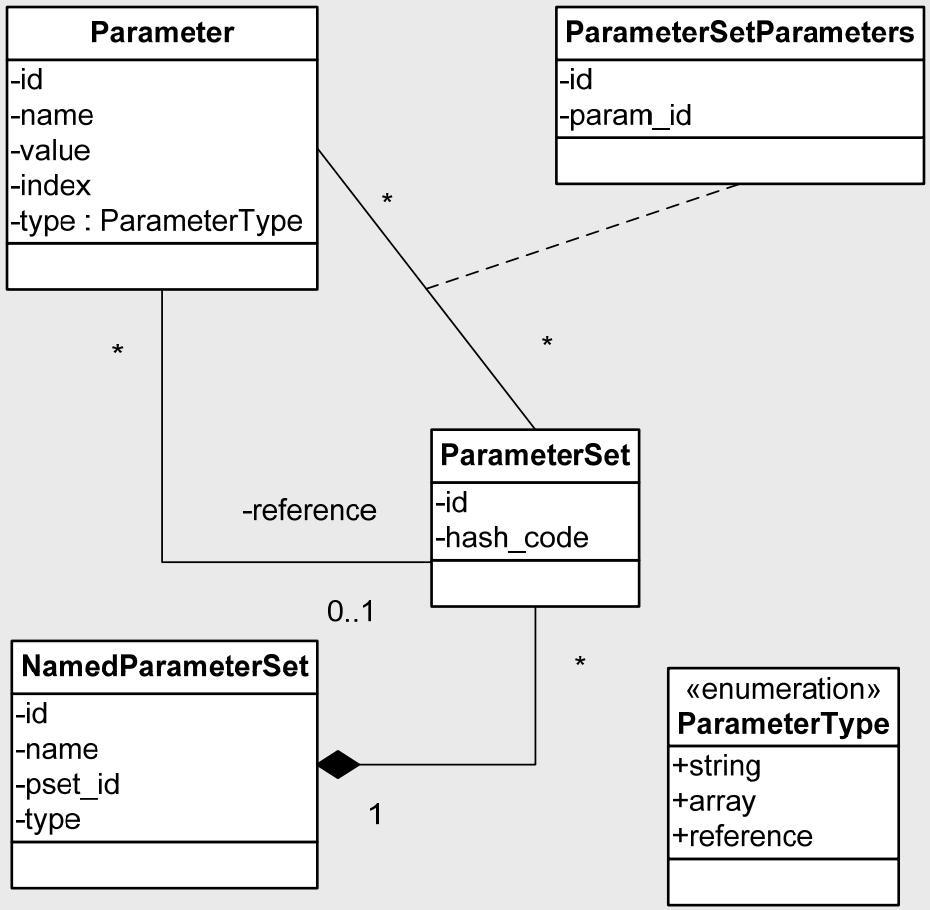
- Ruby on Rails data model are divided into these categories:
  - Parameter space
  - Provenance space
    - Secondary space
  - Run history space



Parameter space

- Contains all parameters used as input for workflows, including
  - Physics parameters (e.g. quark masses)
  - Algorithmic parameters (e.g. convergence criteria) and
  - Execution parameters (e.g. number of nodes used).
- Parameters are grouped as sets, which can be named.
  - Named sets are used an parameters to workflows:
    - Run MyWorkflow psetA psetB

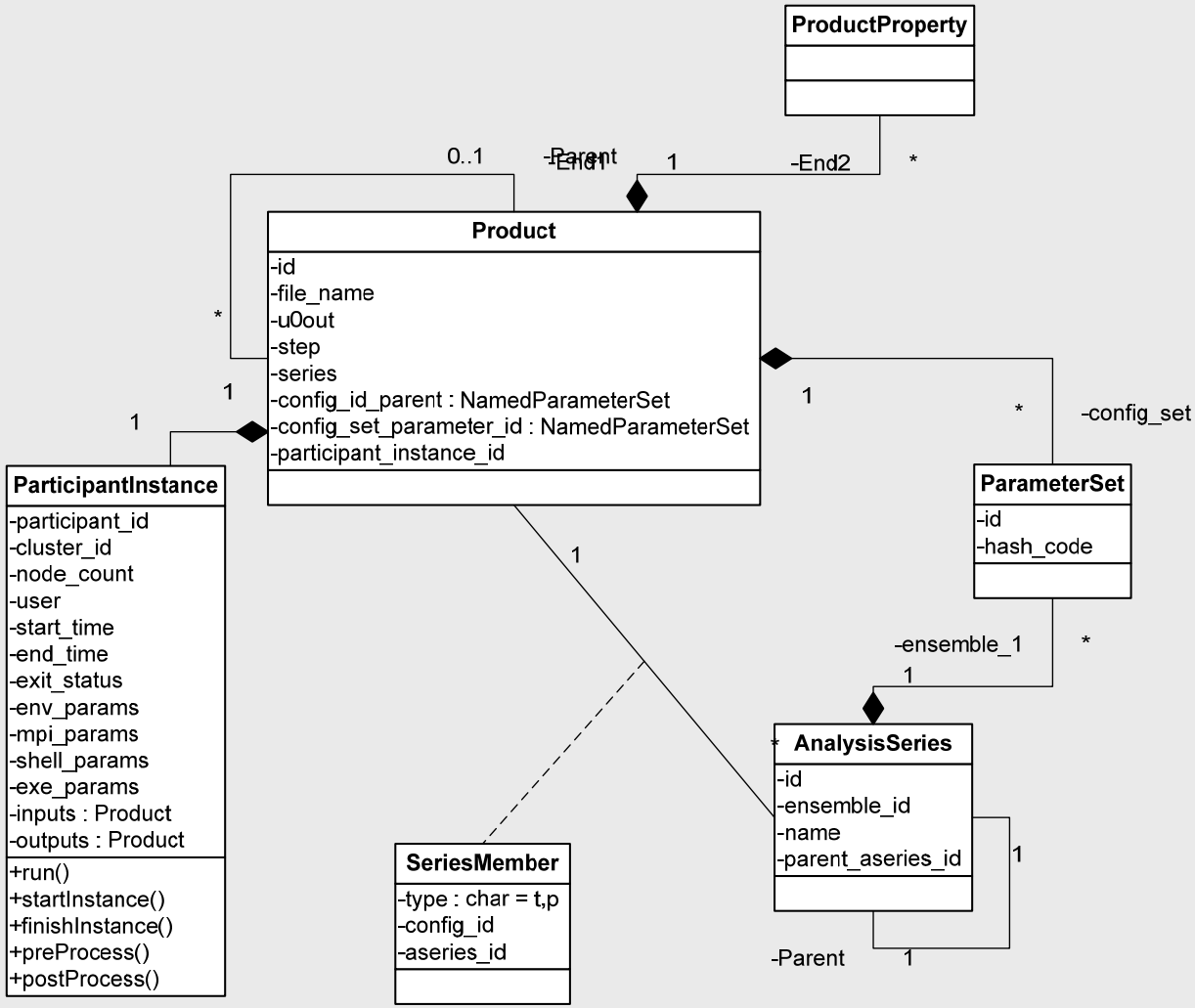
Parameter space



Provenance space

- Information regarding inputs and outputs of each workflow participant.
  - Product (Files) has Product_Properties (Metadata)
  - Products have parent/children relationship to track data origins
- Secondary space can be represented by the product properties
  - Current implementation has both

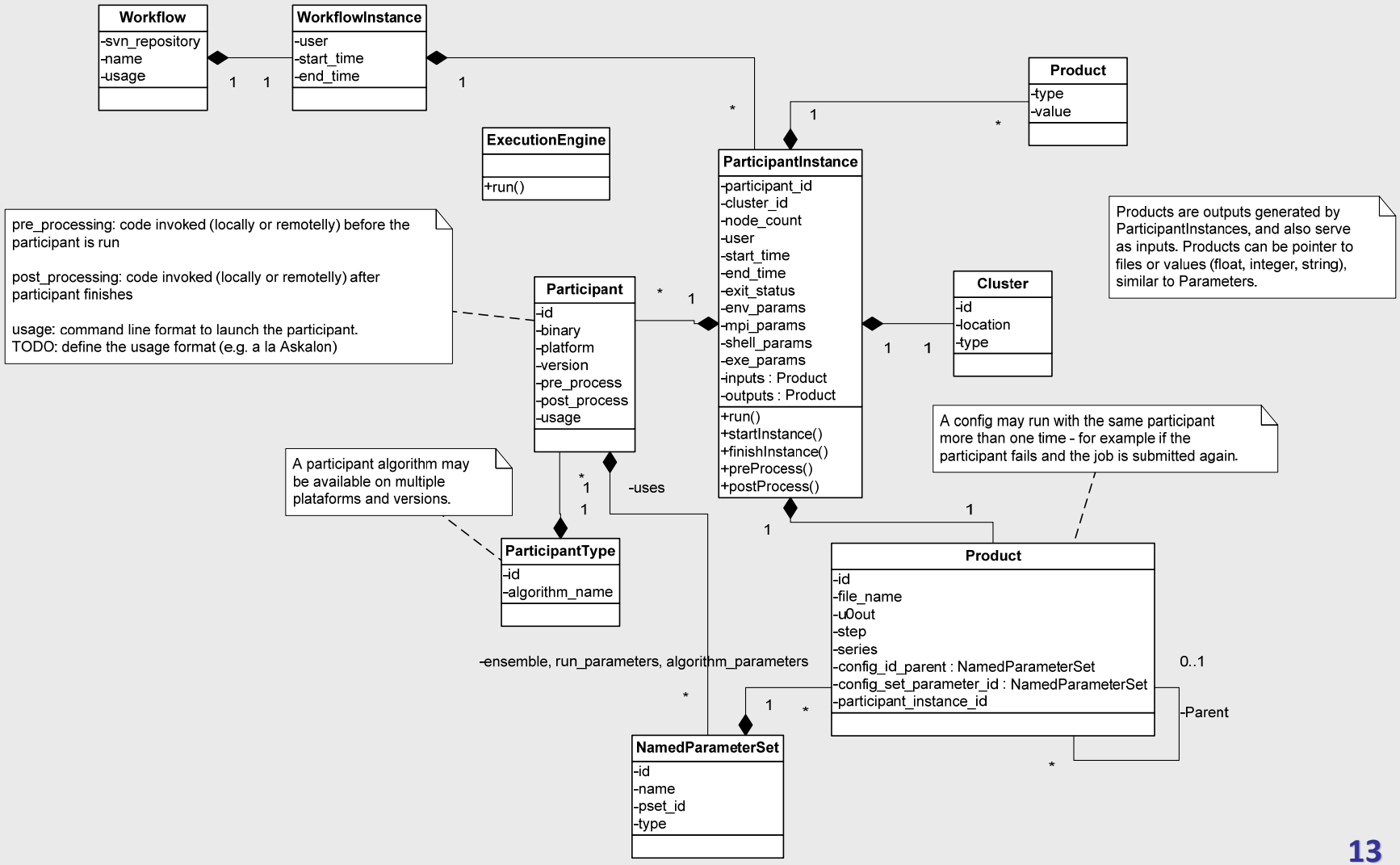
Provenance space



Run history space

- Detailed information about execution
  - Algorithm version
  - Nodes used
  - Outputs generated
  - Environment variables
  - Workflow information

Run history space

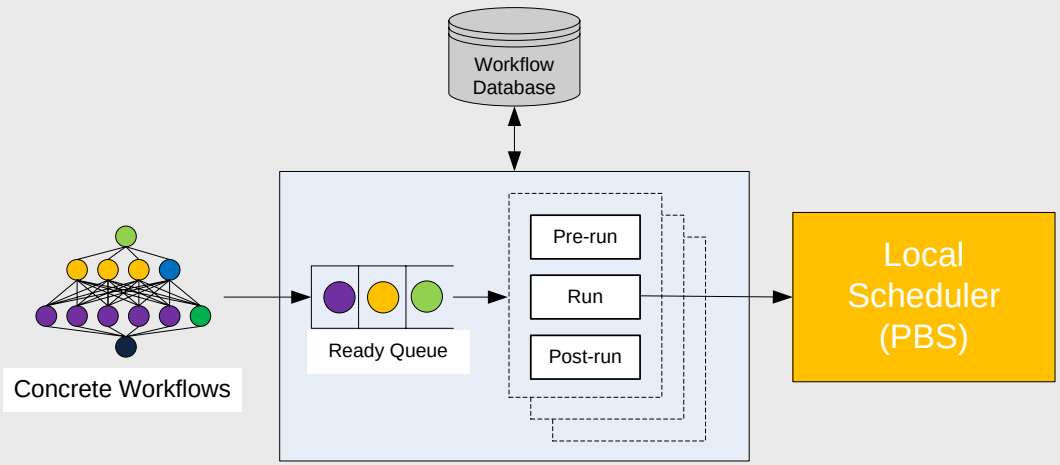


Workflow representation

- Workflow is composed by linking participant inputs and outputs
- Relationship between participants is not represented in the model
- Workflow description dependent on workflow system used
  - Ruote, Pegasus, Swift, Askalon, …

Workflow integration

- Participants interact with database through pre and post run scripts
  - Pre-run: get participant information (binary location), retrieve parameters from database, verification of input files.
  - Post-run: store provenance information in database (file locations and metadata), save participant execution status.



Workflow example (Swift)

```
(file config_out) fetchConfigFile (string config_in) {
  app {
    cp config_in @filename(config_out);
  }
}

(file config_out) runConfGenFile (file config_in, string cluster,
                                  int node_count, string algo_name, string config_pset) {
  app {
    runConfGen @filename(config_in) @filename(config_out) cluster node_count algo_name config_pset;
  }
}

file config[]<ext; exec="/home/piccoli/confgen_db/workflow/swift/bin/mapper.rb", s=conf,d=N>;
config [0] = fetchConfigFile (@strcat ("/", @filename(config[0])));

iterate i {
  config [i+1] = runConfGenFile (config[i], cluster, node_count, algo, conf);
  string fn = @filename(config[i]);
  print (fn);
} until (i == N - 1);
```

Participant

```
Localhost runConfGen /confgen_db/workflow/swift/bin/runConfGen.rb INSTALLED INTEL64::LINUX null
```

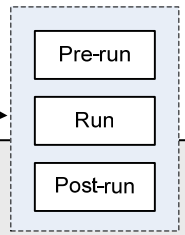
Ruby script that invokes actual binary.
Retrieves parameters from the database based on the parameter set name.
Save results to database.

Workflow example (Ruote)

```
class ExtendProcessDefinition < OpenWFE::ProcessDefinition
  cursor do
      get_parameters

      _break :if => "${f:error} != none"

      prepare_extend

      _break :if => "${f:error} != none"

      _loop do
        extend participant
        _break :if => "${f:counter} - ${f:steps} == ${f:initial_counter}"
        _break :if => "${f:error} != none"
      end
      finish_workflow
  end
end
```

Participant

```
…
p = ParticipantInstance.new
p.runPBS(run_data)
…
```



Parameters are written to a YAML file passed to the application wrapper.
Results are passed back is another YAML file.

Prototype web interface

- Prototype access (porting to MILC)
  - http://jpsi2.fnal.gov:3000/
- Workflows available to users
  - Configuration generation requires selection of single participant
  - Analysis campaign requires selection of several participants – they all need to be compatible
- Workflow run as a single user, which is running the web server