## Pladipus2 Design Plan

Pladipus2 is a tool with which a user can chain together various proteomics tools into reusable workflows, passing the output generated by one step as input into the next. Batches of data may be uploaded so that multiple runs of a workflow can be automatically dispatched, either running locally or on a cluster of worker machines for speed and efficiency.

### Tools

Several Compomics tools will be provided for use by Pladipus2, but it is designed so that other tools may be added with relatively little fuss. To add a tool, a single Java class needs to be created, and added to the pladipus-tools project in the com.compomics.pladipus.tools package. In order to be recognised as a tool, the class should extend com.compomics.pladipus.tools.core.Tool and be annotated @PladipusTool with a displayName attribute, which should be unique and descriptive. For example,

@PladipusTool(displayName = "My New Tool")

public class MyNewTool extends Tool {

…

}

The tool class should contain details of:

* jar file to be run
* allowed input parameters

The base Tool class will contain code to combine the tool and parameter information to construct the final command line instruction to be run. If there is something unique about how a particular tool is run (e.g. not a jar file), this method can be easily overridden in the individual tool class.

Available tools will be discovered by Spring scanning upon starting Pladipus2. This will make tool information (names and parameters) available for the GUI and workflow validation. It will also register the tools in a ListableBeanFactory so they may be called upon and run by worker threads.

### 

### Workflow Template

A new workflow is loaded from an XML template. This may be added from a manually created file via the command line, or automatically generated from a design submitted in the GUI.

A template needs a unique name so the workflow can be easily identified to the user. The XML file format is as follows:

<template name="unique workflow name">

<global></global>

<steps>

<step>

<id>s1</id>

<name>Tool1</name>

</step>

</steps>

</template>

Each <step> represents one run of a tool. The <name> tag should match the displayName of one of the available tools. The <id> tag should be a unique short identifier for each step in the workflow. This id will be used to distinguish parameters in batch files if there is more than one step with the same-named parameter to be populated. It will also be used in chaining the steps together, as described below.

Global parameters are those which may be used by multiple steps, and are contained within the <global> tag. Common examples of global parameters might be output folder locations, or fasta files.

Each <step> and the <global> section may contain a <parameters> element.

<parameters>

<parameter>

<name>parameter\_name</name>

<value>value1</value>

<value>value2</value>

...

</parameter>

</parameters>

Each <parameter> must have a name. For a step, the name must be one of the available input parameters for the tool. If one or more <value> elements is provided, the parameter value(s) will be set for the workflow and will be used for each future run. If the user wishes a parameter to be set on a run-by-run basis (e.g. spectrum files) then this should be included in the XML template with no <value> elements. This means that Pladipus2 will expect a value for that parameter to be loaded in a batch, and will throw an error if it is not.

Parameter values may be just string values, or may contain the following special substitutions

* Step dependencies

{$s1.out} - this will take the output of step with id s1 as input

* Defaults

{$DEFAULT.default\_name} - it is possible to save commonly used values, such as fasta file locations, in the database. This substitutes the named default value.

* Globals

{$GLOBAL.global\_name} - this maps the named global parameter from the template to the step parameter.

The template will undergo strict validation before it is saved. The XML will be checked against an xsd schema to check it is correctly formatted. Then the data contained will be checked against the Pladipus2 tool list to check that the steps have valid tool names and parameters. We also validate that there are no circular dependencies between steps (e.g. step2 takes output from step1, which takes output from step2) and that any ‘default’ parameter values do exist in the database.

Mandatory parameters will be checked for each step. If any mandatory parameter is not listed in the XML, a default value will be added if one exists in the tool. Otherwise, an exception will be thrown requesting the user adds the parameter.

A workflow may be edited and saved again with the same name, but it will be given a new ID in the database and any existing batches will need to be re-loaded to run against the new workflow.

### Database

TODO schema diagram

When a workflow is loaded, the following tables are populated:

* workflows
* workflow\_steps
* workflow\_global\_params
* workflow\_step\_params
* step\_dependencies
* workflow\_global\_values
* workflow\_step\_values

### Loading batches

Each run of a workflow must be loaded with the values for parameters which are not set at workflow level. A batch of run parameter values may be uploaded from a csv or tsv file, with headers being the parameter names. A header file can be generated for the user to populate by searching the database for parameters which are in the workflow\_global\_params and workflow\_step\_params tables without any corresponding values in the workflow\_global\_values and workflow\_step\_values tables. If multiple steps have the same-named parameter to be populated, the header will be in the form step\_id.param\_name to preserve uniqueness.

As in the workflow template, default and global values may be used by using the {$DEFAULT.default\_name} or {$GLOBAL.global\_name} substitutions.

Validation will be done on parameter values before saving. If a parameter has multiple values, we assume values are comma-separated so it should be quoted if using a csv file to upload.

A batch should be given a unique friendly name so that the user knows exactly what is to be run / is currently running. This will generally be taken from the name of the csv/tsv file but may be specified separately.

The following database tables will then be populated:

* batches
* runs
* run\_global\_params
* run\_step\_params

A batch with the same name as an existing batch may be uploaded, but this will replace the current batch (marks current batch as inactive in the database).

### GUI

All Pladipus2 tasks may be run from the command line, or a more user-friendly GUI.

TODO Add diagrams

Login screen

Overview dashboard – available workflows, batches to be run, status of runs

Workflow design screen – similar to Knime, boxes joined by arrows, click to add parameters.

### Running Batches

Steps for a batch to be loaded into run\_steps table with all parameter value substitutions done. Step\_status table populated with run step ID and status – ready, waiting on another step to complete, on queue, processing, complete, error.

If run needs to be run locally, either because user has requested this, or because some file location is only locally accessible, push to local queue and start worker thread. Otherwise push ‘ready’ run steps to ActiveMQ distributed queues. Workers should listen on threads dependent on their operating system, available memory, or anything else which may be a prerequisite for a task to run. These prerequisites will be set as message properties when the step is pushed to the queue.

e.g. Running a tool with a particular parameter set means it can only be run on Windows. Set message property “os=win” when pushing to queue. Windows workers will listen to queues with message selector to check for this.

When a worker takes a step from the queue, we should log in the worker\_steps table which step is being processed by which machine. Possibly also start time so we can check if it has been running for too long. And/or have keep alive check on workers.

Each step should be independent of the worker thread. In other words, the same worker does not have to complete the entire run. We prioritise the same worker completing a whole run, to minimise the amount of potential file copying required, but it is not mandated. After each step is completed, output will be copied to the specified shared output folder, and the worker will check whether any more steps exist for the run and if it is able to run them. If so, it will continue with the next step. If there is a failure, or the worker is unable to run the next step, for example if it can only be run on Windows, another worker can process the step from the queue.

On completion of step, check for dependent steps and set their status to ready/on queue.

### Errors / Logging

All exceptions to be mapped to user-friendly reportable error messages. Reportable exception messages to be read in from a separate properties file for ease of update and potential for support of languages other than English.

Use logback for logging, user able to configure logging level.

We need a mechanism for checking errors when a worker is processing a step. Tools may just output error text to the command line. We need to figure out whether it is an error with the data and any worker would fail, or whether there has been a problem with the worker and we should log it and put the step back onto the queue. Details to be ironed out - ActiveMQ error queue? Database table step\_errors to hold worker error messages for retrieving and reporting to user.

### Configuration

Select a user-accessible location for configuration files which will be used by all modules - database/ActiveMQ settings, logback.xml

Should be able to manually set values in these files, or configure on initial setup. On starting Pladipus2, we should check if all configuration values are set, at least one user exists in the database, etc, and provide means to administrator to set them if not - open setup window in GUI or open config file from command line.

### Testing

Unit tests to be written as code is developed. Extensive use of Mockito to isolate the part of code being tested. Integration tests to be developed using an in-memory database and dummy tools.

### Open Design Questions

* Group parameters - i.e. one of these must be provided. For example, the search engine used in SearchGUI - xtandem, ms\_amanda etc all default to 0 but at least one must be 1 in order to run.
* Boolean flag parameters which don’t take a value
* Input parameter types/validation - multiple values possible? File or folder?
* Output parameters needed for tools? If more than one thing is output, or more than one type possible, should this be specified for input of the next tool? Do we need to be more specific than {$s1.out}?
* Default global parameters and standard mapping to tool input parameters, e.g. many tools have a variation on output folder. Map out, out\_folder, output, etc to global out?
* User groups and access. Should multiple users have access to shared workflows? Who should be allowed to run a job? Worker should only pull jobs for the user?
* Database security.
* Priorities? Ability to promote runs to be processed first?
* Details of worker error handling.