Wings to OPMW Mapping and Requirements

**Authors**: Daniel Garijo, Varun Ratnakar.

**Date**: 20 November 2014.

**Version**: 1.0

**Abstract**: This document specifies how to convert a Wings workflow (specified with p-plan, Wings ontologies and the Wings catalogue) to OPMW. Therefore this document will first describe the set of requirements we need for having a representation of the workflow template and its provenance, and then a mapping between the the existing terms in Wings to convert them to OPMW.

**Challenges to address**:

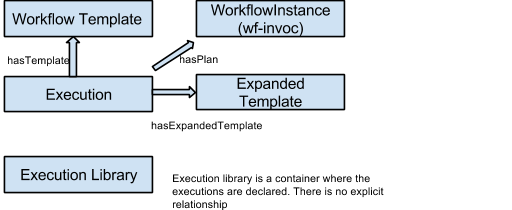
* How to deal with derreferenceable URIs and accessibility to resources?.
* Type of the components and data. This was an issue in the old mapper. Are the classes still instantiated to exploit the relationships in the template level? If they are, then a “class” should be added to all the types found.
* If we archive the current endpoint, what will happen to the currently derreferenceable URIs? (here if we change the endpoint will be fine, even if the uris that used to be there are no longer derreferenceable. Will have to check if you can set up pubby to point at 2 endpoints).
* Which metadata shall be mapped?
* Do we have new metadata to map?
* How do we map collections? This can be a whole new paradigm.

# Workflow execution lifecycle in Wings

The lifecycle starts with the definition of a **workflow template**, which specifies the dependencies among the components of the component catalog. When a user selects to execute a workflow template, a specific plan is prepared with the data bindings (**Workflow Instance**), using the **p-plan/wf-invoc** vocabulary. The data bindings are identifiers for the URIs where the results will be deposited. This means that before executing the workflow, the workflow instance specifies all the containers for the intermediate and final results.

Then the workflow is submitted for execution, and two files are produced. The first one is the **execution process file (called Execution)**, which specifies the execution plan plus all its steps and metadata. It also contains links to the template (through the **hasTemplate** relationship), workflow instance (through the **hasPlan** relationship) and the expanded template (through the **hasExpandedTemplate** relationship, which links the expanded plan to all the resources produced in the workflow execution. This file also contains metadata on their size, language, type, etc.

Finally, all the executions are stored in a **library**, which contains the beginning and ending time of each execution, its status, etc

.

## Example

Below we can find an example of the different stages of a workflow, from its template to its execution, according to the definitions made above.

**Library of executions:** A summary of all the executions of all workflows. We will focus on the execution “Words\_abs-3cb7ac-f3cc24fb-0638-446d-a9c8-86885f9597fb “.

http://www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/executions/library.owl

**Workflow template:** An abstract plan containing the dependencies of a workflow. In our example we will be using the following workflow template:

http://www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/workflows/Words\_abs.owl#Words\_abs

**Workflow Instance:** Invocation plan containing the data bindings and code bindings for a template execution.

http://www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/executions/Words\_abs-3cb7ac-fead07e4-3916-4477-aafa-1e128d45c2fb.owl#Words\_abs-3cb7ac-fead07e4-3916-4477-aafa-1e128d45c2fb

**Execution file:** Plan containing an execution and the start/end time of its steps. Each plan is specialized for a single execution.

http://www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/executions/Words\_abs-3cb7ac-f3cc24fb-0638-446d-a9c8-86885f9597fb.owl#Words\_abs-3cb7ac-f3cc24fb-0638-446d-a9c8-86885f9597fb

**Expanded template:** Plan containing all the data results and their metadata.

www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/executions/Words\_abs-3cb7ac-ff82a8bd-89c7-4d0c-bffb-4aa4b8e695b1.owl#Words\_abs-3cb7ac-ff82a8bd-89c7-4d0c-bffb-4aa4b8e695b1

The two last files are directly related to the provenance capture. The workflow template is the file which will be used to produce the OPMW specification part..

# Mapping Wings terms to OPMW

The mapper follows a two step process: one for transforming templates and another for transforming executions. This two step process is intended, in case one chooses to publish only templates or only provenance executions of their workflows.

The OPMW will expose just the provenance trace, template and their links. It will not expose the execution activity of the Workflow Execution Account or the intermediate results when invoking the workflow, etc.

## Template mapping

The main template mapping has not suffered many changes in the latest versions of wings. The core is in itself the same, and therefore maps to OPMW directly, since OPMW was designed initially for Wings.

**Wings namespace used**: http://www.wings-workflows.org/ontology/workflow.owl#

|  |  |  |
| --- | --- | --- |
| **OPMW Term** | **Wings voc term** | **Rationale/type of mapping** |
| WorkflowTemplateProcess | ComponentVariable | Equivalent classes |
| WorkflowTemplateArtifact | None | A template artifact is a superclass of data variable and parameter variable. In Wings you have the term Variable which is for grouping those plus component variables. |
| DataVariable | DataVariable | Equivalent classes |
| ParameterVariable | ParameterVariable | Equivalent classes |
| WorkflowTemplate | WorkflowTemplate | Equivalent classes |

In this document I will avoid stating the rest of the relationships because they are not direct mappings. For example, one has to take the “inoutlinks” of the template, extract which are the destination nodes and link everything together. They imply a series of transformations that were implemented in the old version of the mapper.

**Expected input**: The owl of the workflow template.

Eg:<http://www.wings-workflows.org/wings-portal/export/users/admin/CompareFiles/workflows/Interleaved_Simple.owl#Interleaved_Simple>

**Expected output**: An owl file with the OPMW description of the template.

**Note:** The types of the processes have to be fixed according to the ontology. This means that right now when a workflow execution process is from a class we don’t add the “Class” to its type, but in some cases we should. This is a problem of the previous mapper.

## Workflow Execution Mapping

Workflow executions have suffered more changes since the previous release of the OPMW mapper, and now the knowledge is divided on different files.

Workflow Library file

It holds the metadata of the Workflow Execution Account (wings:Execution). Before we could find the diagrams created by Wings, but they are no longer available. The properties in OPMW will remain there.

**Wings namespace used**: http://www.wings-workflows.org/ontology/workflow.owl#

|  |  |  |
| --- | --- | --- |
| **OPMW Term** | **Wings voc term** | **Rationale/type of mapping** |
| WorkflowExecutionAccount | Execution | Equivalent Classes |
| overallStartTime | hasStartTime | Equivalent Properties |
| overallEndTime | hasEndTime | Equivalent Properties |
| hasStatus | hasExecutionStatus | Equivalent Properties |
| hasOriginalLogFile | hasLog | Equivalent Properties |

Workflow Instance File, Workflow Execution File, Expanded Template (All three are linked together)

**Wings namespace used**:

[http://www.wings-workflows.org/ontology/workflow.owl#](http://www.wings-workflows.org/ontology/workflow.owl) (template, ns used:temp)

[http://www.wings-workflows.org/ontology/execution.owl#](http://www.wings-workflows.org/ontology/execution.owl)" (execution, ns used: exec)

|  |  |  |
| --- | --- | --- |
| **OPMW Term** | **Wings voc term** | **Rationale/type of mapping** |
| WorkflowExecutionArtifact | no direct mapping | A workflow execution artifact will have a Unique URI.  Taken from expanded template. It corresponds to the DataVariables there. |
| WorkflowExecutionProcess | exec:ExecutionStep | Similar classes.  **In workflow Execution File.**  Workflow execution processes have unique URIs, not based on ExecutionSteps URIs.  From here the next properties are taken: hasStartTime (new), hasEndTime (new),hasExecutionStatus (new), hasLog (new). Start time and end time can be modeled with prov statements. |
| executedInWorkflowSystem | No mapping |  |
| hasExecutableComponent | hasComponentBinding | Semantically they are equivalent. The domain it’s not the same.  Can be found in the Expanded template or the Workflow template. |
| correspondsToTemplateArtifact | derivedFrom | Semantically equivalent. DerivedFrom is more generic |
| correspondsToTemplateProcess | derivedfrom | Semantically equivalent. DerivedFrom is more generic. |
| correspondsToTemplate | hasTemplate | In the execution file |
| hasValue | parameterValue | (just for parameters) |
| hasFileName | No mapping |  |
| hasLocation | hasDataBinding | Can be found in the Expanded template or the Workflow instance. The data cannot be accessed directly, but we can create a workaround to replace it with available URLSs |
| hasSize | hasSize | Equivalent data properties |
| usedAs\_\*role\* |  | \*role\* refers to the role of the output, as it was assigned by the user. This property was created as an extension of opmv:used, to specify the role of an input. |
| generatedAs\_\*role\* |  | \*role\* defines the same as above. It is linked in a similar way as it’s done in the template |

**The URL from the agent responsible of the execution can be extracted from the URL of the domain**

**Expected input:** Four files: the workflow instance has the code bindings; the Execution has the status of each step plus its metadata. The Extended template has the data bindings and their metadata and the Execution library has metadata on the overall status of the workflow plus its overall start time and end time. In the end, the four files have been simplified to two: the execution file and the execution library file. The rest of the files are derived from the RDF data.

**Expected output:** A coherent and grouped description of the workflow execution, summarized according to OPMW, and exposed also in PROV.