Workflow Generation with wfGenes

1st Mehdi Roozmeh *Steinbuch Centre for Computing*
Karlsruhe Institute of Technology, Eggenstein-Leopoldshafen, Germany, mehdi.roozmeh@kit.edu
2nd Ivan Kondov *Steinbuch Centre for Computing*
Karlsruhe Institute of Technology, Eggenstein-Leopoldshafen, Germany, ivan.kondov@kit.edu

***Abstract*—Performing simulation and data analysis on supercomputers requires using workflow management systems (WMSs). Due to the high diversity of application requirements there are currently many different WMSs in use with very different input languages and enactment mechanisms. This hinders the reuse of workflows across these WMSs and makes the transition from one WMS to another difficult. We propose a lightweight framework based on a generative approach to make this affordable for the WMS end user. With a proof-of-concept implementation, wfGenes, workflows for different workflows systems are generated automatically from an abstract workflow description. We tested wfGenes for a use case in computational nanoscience to generate workflows for FireWorks and SimStack. The framework can be readily extended to support further WMSs and to adopt a more standardized workflow description language for its input.**

***Index Terms*—scientific workflow, workflow management system, workflow description language, interoperability**

I. Introduction

Workflow management systems (WMSs) are software environments deployed to orchestrate complex simulation and data analysis consisting of many tasks. Primarily, a WMS provides functionality to schedule the execution of the individual tasks according to their data dependencies and resource requirements and resource availability, and to exploit task and data parallelism. Moreover, some WMSs capture and persist provenance metadata and state information allowing partial and/or repeated execution of the workflow, making the overall application more resilient and more reproducible.

Due to their inherent complexity, scientific workflow applications have diverse technical requirements to the WMS. To mention only a few, these can be capability to process independent tasks in parallel, to allow changes during execution (dynamic workflows), to allow complex hierarchies and subworkflows. Additionally, different educational background, scientific discipline and IT skills of the end users may influence the selection of an existing or the development of a new WMS. Some WMSs, such as Taverna [1], UNICORE [2], [3], Airavata [4], Kepler [5], gUSE [6], Wings [7] and Galaxy [8], provide graphical user interfaces facilitating the management of workflow applications in their full life cycle, including authoring, scheduling and execution, analysis, archiving, and reuse. Other WMSs provide command-line and/or programmatic interfaces, e.g. Pegasus [9], FireWorks [10], Makeflow [11], Swift/T [12], Parsl [13], [14] and Dask [15]. Therefore, designing a single WMS to meet even a small subset of the use-case requirements seems to be a challenge [16], [17]. The large number of use cases of scientific workflows has caused a continually growing number of different WMSs used in different communities and application areas. Almost three hundred WMSs and workflow languages have been reported [18]. Therefore, it has become common that inventories of production workflows have to be ported into another workflow system either due to end of support of the original WMS or because several WMSs are used in a research collaboration. In addition, developing workflows from scratch for two or more different WMSs would cause a steep learning curve even for experienced scientists. Also translating existing workflows to the languages of several different WMSs represents an ”n-to-n” problem that limits their reuse. Promoting interoperability between similar WMSs via standardized workflow description languages or standardized interfaces has been recommended recently [17] as a strategy to tackle this issue.

In the present work, we adopt an alternative approach that neither relies on a common language interface in the WMSs nor on translating already existing workflows using adapters. In particular, we are targeting the case when a scientist is supposed to author a new workflow that can later be deployed in two or more different WMSs. The approach comprises three steps: 1) the workflow designer authors a workflow description that is more abstract than the description used in the WMS, i.e. all WMS-specific details are left out. In principle, the workflow description can be written in a standardized workflow description language; 2) using the workflow description, a workflow in the language of the target WMS is generated automatically including the steps (tasks) with integrated application code and the dependencies describing the control flow and the dataflow; 3) validation of the generated workflow against a schema provided from the target WMS. In this work, we present a proof of concept by implementing this approach in a prototype called wfGenes. Furthermore, we demonstrate the capability of the approach for a use case from computational nanoscience by generating a workflow for two different WMSs, FireWorks [10] and SimStack [19]. The use case is motivated by the demands of a collaborative project in which these two WMSs are used by different partner groups.

The paper is organized as follows. In the next Section II we will provide an overview of related approaches. In Section III we will outline the concept and some implementation details of our approach. After that, in Section IV we will illustrate the method showing how wfGenes is employed to generate workflows for FireWorks and SimStack for the calculation of the adsorption free energy that is a use case from a current collaborative project.

978-1-6654-0452-5/20/$31.00 ©2020 European Union
DOI 10.1109/WORKS51914.2020.00007

II. Related Work

Numerous WMSs have been developed and used in the last two decades and different aspects of workflow interoperability have been addressed: syntactic versus semantic interoperability, as well as interoperability at the level of workflow components to enable integration of data and code into existing workflows. Some of the previous work suggests generative approaches to create workflow instances on different WMSs using formal workflow descriptions (workflow models). Here, we rather propose a lightweight mechanism to accomplish this process that can be readily adopted by end users.

Several projects have established common repositories supporting various WMSs. For instance, myExperiment is an online research repository for bioinformatics workflows developed in Kepler, Taverna and Galaxy [20]. With similar motivation, the SHIWA platform has been developed to provide workflow interoperability in two different levels: coarse-grained interoperability focusing on nesting multiple workflows in a common framework to achieve heterogeneous workflow execution and fine-grained interoperability refers to workflow migration from one WMS to a more preferred one [21]. Interoperability interfaces (adapters) have been developed to achieve the integration of several different WMSs deployed in so called science gateways [22], [23]. Furthermore, a meta-model based data modeling approach combined with integration of workflow data as web services can enable interoperability at the dataflow level [24].

Considerable work has been done in developing workflow languages for use in more than one workflow system. For example, the Simple Conceptual Unified Flow Language (SCUFL) has been adopted in Taverna [1] and Airavata [4]. The Common Workflow Language (CWL) [25] aims at establishing a standard formal description language for scientific workflows by including the most common semantics. The language has been adopted in several different tools and WMSs, e.g. in Arvados, Toil and REANA. Recently the CWL has been extended with CWLProv [26] that is a format for structured provenance representation using the standardized W3C PROV model. CWL workflows can be created and edited with a graphical editor [27]. Although CWL comes from the bioinformatics community it is a generic workflow language. Nevertheless, CWL's generic tool types may need to be extended to enable further use cases, especially from other scientific domains. CWL is a formal description language, whereas wfGenes is a generator that enables the adoption of such a language in different WMSs. In this work, we do not use CWL as a workflow description language because a much simpler definition is sufficient to satisfy the requirements of our use case and more effective for demonstrating the proof-of-concept implementation.

Swift [28] is a domain-specific language enabling concurrent programming to exploit task and data parallelism implicitly rather than describing the workflow as a static directed acyclic graph (DAG). A Swift compiler generates the workflow for different target execution backends. Swift/T [12] provides a Swift compiler to generate code for the dataflow engine Turbine [29] that uses the asynchronous dynamic load balancer (ADLB) based essentially on Message Passing Interface (MPI), a standard in high performance computing. More recently developed tools, such as Parsl [30] and Dask [31], implement a very similar concept for use with the Python language.

III. Proof-of-concept implementation

Figure 1 provides an overview of the three process stages in wfGenes followed by the validation stage. In the following we will outline the workflow description, the wfGene stages and the validation.

A. Workflow Description

wfGenes requires a single workflow description (WConfig) as input including an abstract description of the workflow, i.e. it includes no information about the target WMS and computing resource. Typically WConfig describes names of workflow steps, names of functions and packages, and names of input and output data. Based on these specifications wfGenes constructs the workflow nodes and the workflow graph taking into account data dependencies, and generates the input code for specified WMSs.

Currently, WConfig has intentionally a very simple schema, as shown in Snippet 1. The schema is sufficient to demonstrate the concept and the benefits of the generator tool and to satisfy all requirements of the addressed use case (see Section IV). However, in the design of wfGenes we considered possible extensions of the workflow description schema or a possible later replacement with a standardized, though more complex workflow language, such as CWL.

Snippet 1: Schema for the workflow description (WConfig)

```
$ref: '#/WConfig'
WConfig:
  type: object
  properties:
    nodes:
      type: array
      items: {$ref: '#/Node'}
      minItems: 1
    workflow_name: {$ref: '#/Name'}
  required: [workflow_name, nodes]
  additionalProperties: false
Node:
  type: object
  properties:
    id: {type: integer}
    name: {$ref: '#/Name'}
    tasks:
      type: array
      items: {$ref: '#/Task'}
      minItems: 1
  required: [id, name, tasks]
  additionalProperties: false
Task:
  type: object
  properties:
    function:
      type: array
      items: {$ref: '#/Name'}
      minItems: 1
    inputs:
      type: array
```

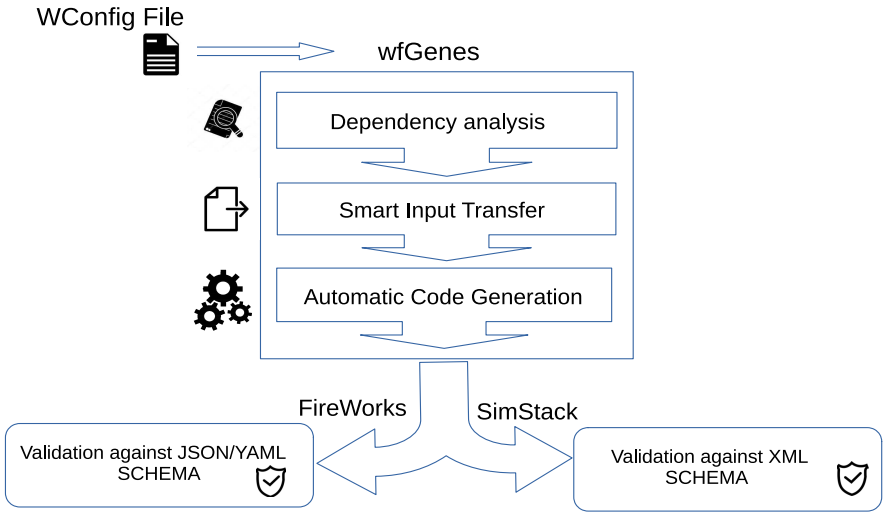


Fig. 1: Workflow description, wfGenes process stages and subsequent WMS specific validation

```
      items: {$ref: '#/Name'}
    outputs:
      type: array
      items: {$ref: '#/Name'}
    kwargs: {type: object}
  required: [function, inputs, outputs]
  additionalProperties: false
Name: {minLength: 1, type: string}
```

The workflow model (cf. Snippet 1) consists of two main elements: nodes and tasks. The nodes correspond to the vertices of the abstract workflow graph, while the tasks do not enter the graph. Tasks in one node are executed in a sequence sharing the same local computing resources, i.e. the same processing elements, launch folder and data in it. In contrast, nodes are actions that may be executed in parallel and/or distributed over different resources. On a high performance computing system, a node is usually executed as a batch job. Every task has a function, wrapping the executed code in the task, and data inputs and outputs. WConfig includes no control flow information and no elements that govern the control flow, such as if-statements.

The WConfig workflow model is data-type agnostic, i.e. the types of data defined in inputs and outputs can be arbitrary, as long as the number of data items is consistent and their labels are used as metadata by wfGenes to construct the dataflow graph. Moreover, the model does not provide any form of data structure (such as arrays) and no control-flow elements (such as maps) to perform distributed computation on such structures (factorial design) [32].

To illustrate the WConfig schema we show two examples in YAML format in Snippets 2 and 3 that define simple workflows, each with three nodes with different graphs, that we call *nabla* and *delta*, respectively. Data entities passed from one task to another or from one node to another, i.e. entities defining the dataflow, are highlighted in yellow color.

Snippet 2: Workflow description for a nabla workflow

```
workflow_name: Nabla workflow
nodes:
- name: node_1
  id: 1
  tasks:
  - function: [source_1 , module_1]
    input: [input1_id1]
    outputs: [ output1_id1 ]
  - function: [source_1, module_2]
    inputs: [input2_id1, output1_id1 ]
    outputs: [ output2_id1 ]
- name: node_2
  id: 2
  tasks:
  - function: [source_1, module_3]
    inputs: [input1_id2, input1_id2]
    outputs: [ output1_id2 ]
- name: node_3
  id: 3
  tasks:
  - function: [source_2, module_1]
    inputs: [ output2_id1 , output1_id2 ]
    outputs: [output1_id3]
```

Snippet 3: Workflow description for a delta workflow

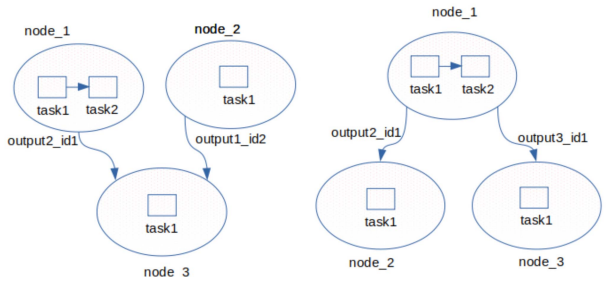
```
workflow_name: Delta workflow
nodes:
- name: node_1
  id: 1
  tasks:
  - function: [source_1, module_1]
    input: [input1_id1]
    outputs: [ output1_id1 ]
  - function: [source_1, module_2]
    inputs: [input2_id1, output1_id1 ]
    outputs: [ output2_id1 , output3_id1 ]
- name: node_2
  id: 2
  tasks:
  - function: [source_1, module_3]
    inputs: [input1_id2, output2_id1 ]
    outputs: [output1_id2]
- name: node_3
  id: 3
  tasks:
```

```
- function: [source_2, module_1]
  inputs: [input1_id3, output3_id1 ]
  outputs: [output1_id3]
```

B. Dependency analysis and input setup

wfGenes constructs the dataflow by matching names of data entities in WConfig. This is why data entities must have globally unique names within a workflow description. For example, in Snippet 2, the data entity `output1_id1` in `node_1` is only used by another task in the same node (intra-node dependency), whereas the entity `output2_id1` is an output of `node_1` and input in `node_3` (inter-node dependency). As illustrated in Fig. 2, the corresponding workflow graph has two levels of granularity: inter-node and intra-node level.

wfGenes generates one Python wrapper that executes all tasks in the node according to the specified sequence in the WConfig. The benefit of this is sharing module imports, sharing input data that are common for two or more tasks and zero-copy transfer of data between tasks (smart input transfer as shown in Figure 1). Example for this is the data entity `output1_id1` in the nabla workflow described in Snippet 2 and in Fig. 2a). Obviously, this optimization assumes that all tasks within a node will be executed on the same computing resource.



(a) Nabla workflow (b) Delta workflow

Fig. 2: Workflow graphs generated from two different workflow descriptions

C. Generating FireWorks workflows

The workflow description schema (WConfig) introduced in Sec. III-A covers only a common subset of the workflow languages of different WMSs. For example, the task and node elements defined in the WConfig can be uniquely mapped to *Firetasks* and *Fireworks*, respectively, in FireWorks WMS. Optionally, wfGenes adds WMS specific information to the generated workflow. For example, a very specific feature of FireWorks is duplicates detection [10] that is a suitable cross-workflow mechanism for data reuse in high throughput computing. Adding a DupeFinderExact object to a workflow node will cause a check at launch time for identical nodes (with identical tasks and data) are running or completed. If there is an identical node in any workflow instance in the database, then the node is not executed but only starts sharing launch data, particularly the outputs of the identical node. Such and further FireWorks specific information can be added to the generated FireWorks workflow.

FireWorks has different mechanisms for managing dataflow implemented with different combinations of keywords and objects. One mechanism is based on data file transfers (whereby only metadata is stored in the WMS) and another one for metadata and any JSON serializable data using an FWAction object. The dataflow methods can be selected prior to calling wfGenes so that a specific mechanism is used in the generated workflow.

WConfig does not include explicit control flow description and the control flow, i.e. the order of execution of the nodes, is derived from the data dependencies, as outlined in Sec. III-B. In FireWorks the control flow must be defined explicitly in the workflow in the links object that is added automatically by wfGenes, e.g. for the nabla workflow it is:

```
links: {'1': [3], '2': [3], '3': []}
```

The generated workflow can be extended with parameters specifying the computing resources and the execution environment in the different nodes. This information is intentionally not included in the WConfig because it is specific for the target computing resources, software environment and on the software deployment model. One good way to tackle this is to use lightweight containers that can be defined as resources and define suitable container images for the execution of different nodes.

D. Generating SimStack workflow active nodes (WaNos)

SimStack has a graphical interface for workflow construction, resource configuration and allocation, job submission and monitoring. A workflow in SimStack consists of workflow active nodes (WaNos) and there is no explicit control flow definition, i.e. SimStack derives the order of node execution at launch time. Every node in WConfig is mapped to one WaNo each including four files. The tree below shows all generated files by wfGenes for a workflow with three nodes, such as the nabla workflow:

```
Output_Directory
└── SimStack
    ├── node_1
    │   ├── node_1_Wrapper.py
    │   ├── node_1.xml
    │   ├── run.sh
    │   └── setenv.sh
    ├── node_2
    │   ├── node_2_Wrapper.py
    │   ├── node_2.xml
    │   ├── run.sh
    │   └── setenv.sh
    └── node_3
        ├── node_3_Wrapper.py
        ├── node_3.xml
        ├── run.sh
        └── setenv.sh
```

The Wrapper.py file includes python interfaces, in particular the functions defined in the task elements of WConfig. The

XML file contains a model for rendering the graphical user interface of the particular WaNo in SimStack. The model includes also the definitions of inputs and output files and further input data that are encoded in YAML during WaNo execution. The run.sh file is the executed script that essentially sources the environment from setenv.sh and starts the Wrapper.py. The thus generated WaNos can be imported in the graphical editor of SimStack. Therein, the end user can interactively compose workflows based on these and other available WaNos. Nevertheless, the generator also creates a full workflow from the information available in the WConfig and this can also be imported and further processed in SimStack.

E. Implementation

In the following, we provide some implementation details for the analysis and code generation phases shown as pseudo code in Algorithm 1 and Algorithm 2, respectively. Further details can be found directly in the wfGenes source code [33]. In the first analysis phase, wfGenes converts WConfig to multiple list objects i.e. input, output and function lists. Afterwards, join operations are performed on list pairs to establish required relations for constructing directed acyclic graph (DAG) and optimized code generation. In the next phase, Algorithm 2, wfGenes basically produces all necessary code and files for specific WMS by filling the respective template. In fact, these templates are reusable code snippet available with wfGenes framework that are filled based on the data-flow derived in the analysis phase. Finally, wfGenes integrates the generated code to construct the complete workflow by establishing global dependency links in the specific WMS format.

F. Validation

In the last stage, all workflows generated by wfGenes are validated against a workflow schema if such schema is provided by the WMS. This stage ensures that the workflows will be imported in the WMS without errors. JSON schema is used for FireWorks workflows in combination with the jsonschema package. Because in SimStack workflows and WaNos are written in XML, we have created an XML schema and used the xmlschema package.

IV. Use case: Adsorption free energy calculation

The wfGenes tool has been first employed in the setting of a collaborative project in which different participants use two different WMS. This often requires defining a set of simulation workflows simultaneously in two different languages. One of these workflows is the adsorption energy workflow that we will briefly demonstrate in this section.

The adsorption free energy is an important property used in catalysis modeling and is the free energy of the reaction of adsorption

$$\Delta G_{\text{ads}} = \Delta G_{\text{complex}} - (\Delta G_{\text{substrate}} + \Delta G_{\text{adsorbate}}) \ . \quad (1)$$

The calculation requires the free energies $\Delta G_{\text{substrate}}$, $\Delta G_{\text{adsorbate}}$ and $\Delta G_{\text{complex}}$ of the substrate, the adsorbate, and the substrate–adsorbate complex, respectively. Figure 3 shows the graph of the workflow calculating the dissociative adsorption energy of molecular oxygen (O_2) on a metal surface slab. In the nodes of type `Relax structure` the atomic structures of the substrate (*), the adsorbate (O_2) and the substrate–adsorbate complex (*O) are relaxed by minimizing their total energies. Using the relaxed structures the molecular vibrations are computed in the nodes with names `Normal mode analysis` to calculate the zero-point vibrational energies and the vibrational entropies. All these data are merged and used in the final node `Adsorption energies` to calculate the adsorption free energy using Eq. (1).

The workflow description of three of the nodes processing the *O species is shown in Snippet 4. The calculations are performed in functions `build_structure`, `structure_relaxation` and `vibrations` in module

Algorithm 1 Analysis phase

```
for node in nodes do
  for task in node.tasks do
    From WConfig load input, output, function/module
    names in different array instances
  end for
end for
Step 1: Check for duplicate input
for node in nodes do
  for input1 in node.inputs do
    for input2 in node.inputs do
      if input1 == input2 then
        MARK input2 as duplicate
      end if
    end for
  end for
end for
Step 2: Perform local dependency analysis
for node in nodes do
  for input in node.inputs do
    for output in node.outputs do
      if input == output then
        MARK output as local dependent
      end if
    end for
  end for
end for
Step 3: Perform global dependency analysis
for node1 in nodes do
  for node2 in nodes do
    for input in node1.inputs do
      for output in node2.outputs do
        if output == input then
          MARK node2 as global dependent to node1
        end if
      end for
    end for
  end for
end for
```

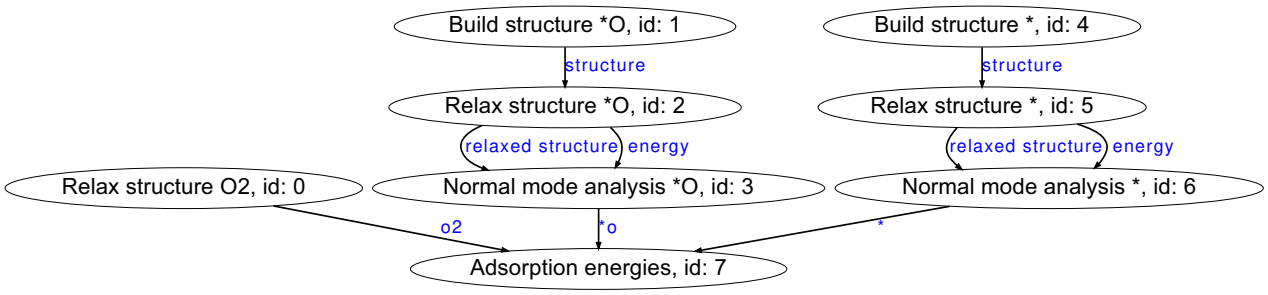


Fig. 3: Graph of the workflow for computing the adsorption energy of oxygen

Algorithm 2 Code generation phase

```
for node in nodes do
  for task in node.tasks do
    Fill python template to generate task wrappers
  end for
end for
if Users preference == SimStack then
  for node in nodes do
    for task in node.tasks do
      Fill XML template for each node to render SimStack
      WaNos (Simulation Blocks)
    end for
    Fill XML template to compose complete workflow in
    SimStack
  end for
  Validate against XML schema to assure format and
  datatype consistency before job submission
end if
if Users preference == FireWorks then
  for node in nodes do
    for task in node.tasks do
      Fill FireWorks template to construct nodes
    end for
    Establish global dependency to compose complete
    workflow in FireWorks
  end for
end if
Validate against JSON/YAML schema to assure format and
data-type consistency before job submission
```

`ueffumax`. The builtin function `MERGE` aggregates specified data from previous tasks and nodes into a new data entity.

The generated and validated FireWorks workflow was added to the LaunchPad and executed on an high performance computing cluster. The generated and validated WaNos were imported to the SimStack client and the composed workflow was submitted independently to the same computing cluster. After this the results of the two workflows were verified by comparison.

Snippet 4: Part of the WConfig of the adsorption workflow

```
workflow_name: Adsorption energy
nodes:
- name: Build structure *O
  id: 1
  tasks:
  - function: [ueffumax, build_structure]
    input: [parameters_id1]
    outputs: [structure_id1]
- name: Relax structure *O
  id: 2
  tasks:
  - function: [ueffumax, structure_relaxation]
    inputs: [parameters_id2, structure_id1]
    outputs:
    - relaxed structure_id2
    - energy_id2
    - forces_id2
    - dipole_id2
- name: Normal mode analysis *O
  id: 3
  tasks:
  - function: [ueffumax, vibrations]
    inputs:
    - parameters_id3
    - relaxed structure_id2
    outputs:
    - vibrational energies_id3
    - vibrational entropy_id3
    - vibrational_entropy_term_id3
    - vibrational_partition_function_id3
    - zero_point_energy_id3
    - transition_state_id3
    - energy_minimum_id3
  - function: [BUILTIN, MERGE]
    inputs:
    - energy_minimum_id3
    - energy_id2
    - zero_point_energy_id3
    - vibrational_entropy_term_id3
    - rotational_entropy_term_id3
    - translational_entropy_term_id3
    outputs:
    - *o
```

V. Conclusion

In this work, we presented a lightweight interoperability framework including a formal but simple workflow description WConfig, a three-stage code generation and validation against workflow schema of the generated workflows. With the proof-of-concept implementation wfGenes we have generated FireWorks and SimStack workflows from a simple description. We have verified the framework in a real-life use case from computational nanoscience using high performance computing resources. Currently, the wfGenes tool can be also used to

accelerate the prototyping of new applications using these two WMSs.

The presented framework and its proof-of-concept implementation wfGenes allow extensions in several directions: The workflow description can be replaced by a standardized workflow description language such as CWL. The type of data and code resources specified by the workflow designer are currently restricted to YAML/JSON data files, and Python functions and local executables, respectively, but these can be generalized in future. Additionally, an approach for semantic annotation, such as JSON-LD can be adopted in the generator, based on XML/JSON schemas provided by the target WMSs. Adding support for further WMSs and employing CWL for workflow description are expected to speed up the adoption of the tool.

VI. Acknowledgment

The authors gratefully acknowledge support by the GRK2450.

References

[1] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. W1, pp. W557–W561, Jul. 2013. [Online]. Available: http://academic.oup.com/nar/article/41/W1/W557/1094153/The-Taverna-workflow-suite-designing-and-executing

[2] A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak, S. Bergmann, R. Breu, J. Daivandy, B. Demuth, A. Eifer, A. Giesler, B. Hagemeier, S. Holl, V. Huber, N. Lamla, D. Mallmann, A. Memon, M. Memon, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, T. Schlauch, A. Schreiber, T. Soddemann, and W. Ziegler, "UNICORE 6 — Recent and future advancements," *Annals of Telecommunications*, vol. 65, no. 11, pp. 757–762, 2010. [Online]. Available: http://dx.doi.org/10.1007/s12243-010-0195-x

[3] K. Benedyczak, B. Schuller, M. P.-E. Sayed, J. Rybicki, and R. Grunzke, "UNICORE 7 — Middleware services for distributed and federated computing," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, Jul. 2016, pp. 613–620. [Online]. Available: https://doi.org/10.1109/HPCSim.2016.7568392

[4] M. Pierce, S. Marru, L. Gunathilake, T. A. Kanewala, R. Singh, S. Wijeratne, C. Wimalasena, C. Herath, E. Chinthaka, C. Mattmann, A. Slominski, and P. Tangchaisin, "Apache Airavata: Design and Directions of a Science Gateway Framework," in *2014 6th International Workshop on Science Gateways*. Dublin, Ireland: IEEE, Jun. 2014, pp. 48–54. [Online]. Available: http://ieeexplore.ieee.org/document/6882068/

[5] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006. [Online]. Available: http://doi.wiley.com/10.1002/cpe.994

[6] A. Balasko, Z. Farkas, and P. Kacsuk, "Building science gateways by utilizing the generic WS-PGRADE/gUSE workflow system," *Computer Science*, vol. 14, no. 2, pp. 307–325, 2013. [Online]. Available: http://dx.doi.org/10.7494/csci.2013.14.2.307

[7] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, "Wings: Intelligent Workflow-Based Design of Computational Experiments," *IEEE Intelligent Systems*, vol. 26, no. 1, pp. 62–72, Jan. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5396300/

[8] E. Afgan, D. Baker, M. van den Beek, D. Blankenberg, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. Grüning, A. Guerler, J. Hillman-Jackson, G. Von Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor, A. Nekrutenko, and J. Goecks, "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Research*, vol. 44, no. W1, pp. W3–W10, 05 2016. [Online]. Available: https://doi.org/10.1093/nar/gkw343

[9] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, May 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X14002015

[10] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson, "FireWorks: A dynamic workflow system designed for high-throughput applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037–5059, 2015. [Online]. Available: http://dx.doi.org/10.1002/cpe.3505

[11] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, ser. SWEET '12. New York, NY, USA: ACM, 2012, pp. 1:1–1:13. [Online]. Available: http://doi.acm.org/10.1145/2443416.2443417

[12] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. Delft: IEEE, May 2013, pp. 95–102. [Online]. Available: http://ieeexplore.ieee.org/document/6546066/

[13] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, I. Foster, M. Wilde, and K. Chard, "Scalable Parallel Programming in Python with Parsl," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning) - PEARC '19*. Chicago, IL, USA: ACM Press, 2019, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3332186.3332231

[14] Y. Babuji, I. Foster, M. Wilde, K. Chard, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, and J. M. Wozniak, "Parsl: Pervasive Parallel Programming in Python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '19*. Phoenix, AZ, USA: ACM Press, 2019, pp. 25–36. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3307681.3325400

[15] M. Rocklin, "Dask: Parallel Computation with Blocked algorithms and Task Scheduling," in *Python in Science Conference*, Austin, Texas, 2015, pp. 126–132. [Online]. Available: https://conference.scipy.org/proceedings/scipy2015/matthew_rocklin.html

[16] V. Curcin and M. Ghanem, "Scientific workflow systems - can one size fit all?" in *2008 Cairo International Biomedical Engineering Conference*, 2008, pp. 1–9.

[17] R. M. Badia, E. Ayguade, and J. Labarta, "Workflows for science: A challenge when facing the convergence of HPC and Big Data," *Supercomputing Frontiers and Innovations*, vol. 4, no. 1, pp. 27–47, Mar. 2017. [Online]. Available: http://superfri.org/superfri/article/view/125

[18] "Computational data analysis workflow systems," https://s.apache.org/existing-workflow-systems, accessed: 2020-07-31.

[19] *SimStack: The Boost for Computer Aided-Design of Advanced Materials*. [Online]. Available: https://www.simstack.de/

[20] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure, "myExperiment: A repository and social network for the sharing of bioinformatics workflows," *Nucleic Acids Research*, vol. 38, no. suppl_2, pp. W677–W682, Jul. 2010. [Online]. Available: https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkq429

[21] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, A. Balasko, and Z. Farkas, "Enabling scientific workflow sharing through coarse-grained interoperability," *Future Generation Computer Systems*, vol. 37, pp. 46–59, Jul. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X14000417

[22] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, "WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities," *Journal of Grid Computing*, vol. 10, no. 4, pp. 601–630, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10723-012-9240-5

[23] P. Kacsuk, Ed., *Science Gateways for Distributed Computing Infrastructures*. Cham: Springer International Publishing, 2014. [Online]. Available: http://link.springer.com/10.1007/978-3-319-11268-8

[24] A. Bender, A. Poschlad, S. Bozic, and I. Kondov, "A service-oriented framework for integration of domain-specific data models in scientific workflows," *Procedia Computer Science*, vol. 18, pp. 1087–1096, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2013.05.274

[25] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0," p. 5921760 Bytes, 2016. [Online]. Available: https://figshare.com/articles/Common_Workflow_Language_draft_3/3115156/2

[26] F. Z. Khan, S. Soiland-Reyes, R. O. Sinnott, A. Lonie, C. Goble, and M. R. Crusoe, "Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv," *GigaScience*, vol. 8, no. 11, p. giz095, Nov. 2019. [Online]. Available: https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giz095/5611001

[27] http://docs.rabix.io/rabix-composer-home, accessed: 2020-07-31.

[28] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011. [Online]. Available: https://doi.org/10.1016/j.parco.2011.05.005

[29] J. M. Wozniak, T. G. Armstrong, K. Maheshwari, E. L. Lusk, D. S. Katz, M. Wilde, and I. T. Foster, "Turbine: A Distributed-memory Dataflow Engine for High Performance Many-task Applications," *Fundamenta Informaticae*, vol. 128, no. 3, pp. 337–366, 2013. [Online]. Available: https://doi.org/10.3233/FI-2013-949

[30] Y. N. Babuji, K. Chard, I. T. Foster, D. S. Katz, M. Wilde, A. Woodard, and J. M. Wozniak, "Parsl: Scalable parallel scripting in python." in *IWSG*, 2018.

[31] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, no. 130-136. Citeseer, 2015.

[32] P. Alper, K. Belhajjame, and C. A. Goble, "Static analysis of Taverna workflows to predict provenance patterns," *Future Generation Computer Systems*, vol. 75, pp. 310–329, Oct. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X17300225

[33] M. Roozmeh, *wfGenes*, https://git.scc.kit.edu/th7356/wfgenes.