KadiStudio: FAIR Modelling of Scientific Research Processes

RESEARCH PAPER

LARS GRIEM
PHILIPP ZSCHUMME
MATTHIEU LAQUA
NICO BRANDT
EPHRAIM SCHOOF
PATRICK ALTSCHUH
MICHAEL SELZER

*Author affiliations can be found in the back matter of this article

ABSTRACT

FAIR handling of scientific data plays a significant role in current efforts towards a more sustainable research culture and serves as a prerequisite for the fourth scientific paradigm, that is, data-driven research. To enforce the FAIR principles by ensuring the reproducibility of scientific data and tracking their provenance comprehensibly, the FAIR modelling of research processes in form of automatable workflows is necessary. By providing reusable procedures containing expert knowledge, such workflows contribute decisively to the quality and the acceleration of scientific research. In this work, the requirements for a system to be capable of modelling FAIR workflows are defined and a generic concept for modelling research processes as workflows is developed. For this, research processes are iteratively divided into impartible subprocesses at different detail levels using the input-process-output model. The concrete software implementation of the identified, universally applicable concept is finally presented in form of the workflow editor KadiStudio of the Karlsruhe Data Infrastructure for Materials Science (Kadi4Mat).

CORRESPONDING AUTHOR:

Lars Griem

Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany

lars.griem@kit.edu

KEYWORDS:

FAIR principles; workflows; research data management; electronic lab notebook; inputprocess-output model

TO CITE THIS ARTICLE:

Griem, L, Zschumme, P, Laqua, M, Brandt, N, Schoof, E, Altschuh, P and Selzer, M. 2022. KadiStudio: FAIR Modelling of Scientific Research Processes. *Data Science Journal*, 21: 16, pp. 1–17. DOI: https://doi.org/10.5334/dsj-2022-016

1 INTRODUCTION

Through technological advances in instrumentation and computational performance, the amount of data produced in engineering sciences, and especially materials science, has increased significantly over the past decades. This development paves the way for a new scientific paradigm, commonly known as data science (Hey et al. 2009), that focuses on the systematic analysis of data to generate new knowledge or insight. It allows to accelerate the innovation of new materials and can thus be seen as a driving force for future developments. Prerequisite for this paradigm is the availability, completeness, and reproducibility of the research data to be examined.

Establishing the paradigm thus requires an extensive data sharing concept that enables structured storage and management of research data according to the FAIR – Findable, Accessible, Interoperable, and Reusable – principles (Draxl & Scheffler 2020; Wilkinson et al. 2016). A sophisticated infrastructure in form of a repository in which data can be recorded and administered as well as analysed, transformed, and visualised is therefore beneficial. Moreover, a system capable of modelling scientific processes and data flows as automatable and configurable workflows is necessary. It not only ensures the datas' reproducibility and tracks their provenance comprehensibly but also allows to generate new knowledge and insight by processing the stored data. In this way, workflows contribute decisively to the quality assurance and acceleration of scientific research. As for scientific data, workflows need to be formulated in a FAIR manner in order to be accessible and usable for a broad scientific audience as well as for data science approaches. Implementing a system capable of FAIR modelling of research processes as such workflows requires two contradictory conditions to be met. Firstly, as scientific research exhibits heterogeneous tools and procedures, the proposed workflow system must be kept generic and easily extensible. Secondly, it must be simple and intuitive in use to minimise the effort required to formulate workflows and thus increase acceptance among researchers (Pizzi et al. 2016).

Infrastructures which integrate the creation, exchange and execution of workflows are, to date, already realised in various implementations, such as Jupyter Notebooks (Kluyver et al. 2016), Galaxy (Afgan et al. 2018), Fireworks (Jain et al. 2015) and AiiDA (Pizzi et al. 2016). The aforementioned infrastructures as well as all other implementations known to us, however, do not satisfy the named conditions of simple usability, generic extensibility and FAIR process modelling. Jupyter Notebooks for example, enables the modelling of scientific processes, but its focus lies on computer-aided scientists. Hence, programming experience is required on part of the user to formulate workflows, which in our experience, is a hindrance for many scientists. Galaxy, on the other hand, allows researchers without programming knowledge to formulate workflows. However, these are limited to the field of life sciences and thus do not represent a generic solution for FAIR process modelling. Fireworks is also limited to a specific domain that is simulations and the management of computer resources. A generic, domain-independent use that also includes manual work steps is therefore not possible. AiiDA presents a generic solution for formulating workflows in form of scripts. Nevertheless, as the focus is on computational science, it is not possible to implement manual steps into workflows, thus excluding scientists working in analogue from the target group. Additionally, programming experience is again required. Consequently, to our knowledge, there is no system that offers a generic, domain independent approach to formulate workflows in a FAIR manner, which targets not only computer-based working scientists with an affinity for programming but also analogue working researchers with little programming expertise. In this paper we therefore introduce a possible solution for FAIR modelling of scientific processes that takes these requirements into account and present its concrete implementation in form of the workflow editor KadiStudio. For this, a concept is first developed that allows to abstract scientific processes according to a uniform schema that serves as the basis for FAIR modelling. Subsequently the technical implementation of the openly accessible software KadiStudio (Kadi4Mat Team & Contributors 2022e) (available under https://doi.org/10.5281/zenodo.6810891) is presented in detail, with reference to this concept.

2 CONCEPT

The development of a generic system for modelling research processes requires the identification of a common structure that can be imposed on any process. For this purpose,

we iteratively disassemble scientific processes into atomistic descriptions at different levels of detail. The term *atomistic description* here refers to the subdivision of a process into impartible parts. This description allows to identify generic elements within processes that can be reused in other use-cases. Basis for the disassembly is the Input Process Output (IPO) model presented in Figure 1, which is known from systems analysis (Goel 2010; Zelle 2004).

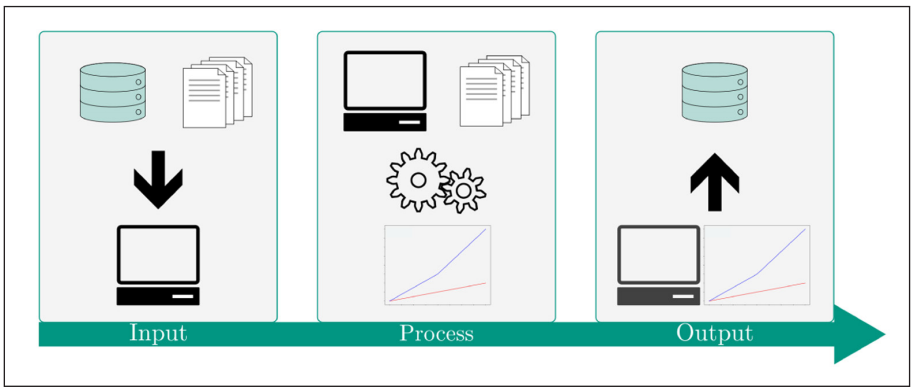

Figure 1 Schematic description of the IPO concept. Through defined inputs a process is parameterised and subsequently executed. The generated results are available via defined outputs.

This model describes processes as the combination of input, process and output. Accordingly, a process commences with the collection and preparation of the data to be investigated through specified inputs. The subsequent processing of the collected data is then performed according to a defined process. The results obtained in this process are finally available for further use via defined outputs.

The most extensive atomistic process description is at project level and includes the complete process as shown in Figure 2. This is to be understood in the sense that a research project can only be described by the entire process. Describing the experimental investigation of a sample, for instance, requires the entire experiment to be modelled; no further subdivision is possible.

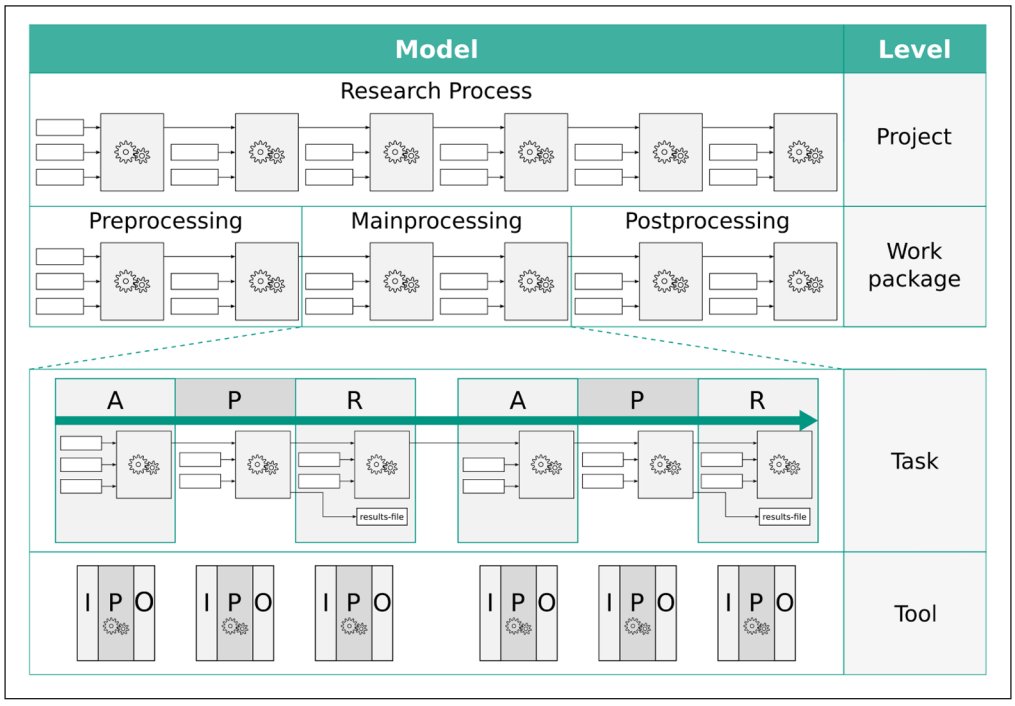

Figure 2 Abstraction of a research process at different levels of detail. Each grey box with gears models a work step while the white boxes represent their parameterisation. APR refers to the description of tasks as data acquisition, data processing, and data routing. Iteratively structuring a research process according to the IPO model ultimately defines it via multiple generic tools.

To describe the process at the less complex work package level, the IPO model is imposed onto it. This model divides the research process into three sequential work packages that correspond to the different elements of the IPO-model. These packages are pre-processing, main-processing, and finally post-processing. Applied to the aforementioned experimental investigation of a sample, the pre-processing corresponds to the preparation of the experiment, the main-processing to the actual experiment and the post-processing to the final analysis

of the obtained data or the interpretation of the results. Each of these work packages can consist of an arbitrary number of work steps. The pre-processing of an experiment, for instance, could include the grinding and polishing of a material sample as well as the calibration of the microscope. Abstracting the process into the work packages pre-, main- and post-processing already enables the identification of elements that might be reusable in other use-cases. The pre-processing of the experiment in form of the microscope calibration and the sample preparation can for example be reused for similar investigations. This level of detail however is not sufficient to be used as a basis for a generic workflow modelling system. Instead, a more in-depth description of the individual work packages is required. For this purpose, the IPO model is applied to each coherent task within the identified work packages. A coherent task refers to the logically separable work steps within a work package. The pre-processing of the aforementioned experimental investigation of a sample could for instance contain the tasks *sample grinding* and *microscope calibration*. Structuring these tasks according to the IPO model results in a process description at task level, which we refer to as **APR** structure, that is schematically illustrated in Figure 2. It describes the data flow within and between the individual steps of a task through data **A**cquisition, data **P**rocessing and data **R**outing. In this abstraction, the data acquisition contains all work steps that collect and prepare the necessary data for the corresponding task and then forwards them to data processing. Within data processing, these inputs are processed to generate new data. The data thus obtained in form of results or intermediate results are finally forwarded in the data routing. The routing of results can be realised to any desired destination such as a file or the data acquisition of a subsequent task. The work packages Pre-, Main- and Post-processing can consist of any number of such APR processes as indicated in Figure 2, and can hence be understood as the concatenation of these elements.

Accordingly, the APR model mainly serves to describe the data flow within each task. This allows for the comprehensible traceability of the data flow within them, promoting a better understanding of the process. Moreover, the identified APR structures can be reused and applied to different use-cases. However, as the APR processes consist of a defined combination of multiple individual work steps, they are too use case specific to be used as a generic workflow modelling structure. Consequently, the IPO model is again applied to each work step within the APR processes. This results in an atomistic description of the research process at tool level. The individual work steps of each APR processes are now defined as generic and reusable tools with specific inputs and outputs as well as a process. In the considered sample preparation example, these could be the work steps grinding, polishing, and etching. Since this structuring corresponds to the original definition of the IPO model, it is here also referred to as IPO. The actual data generation of the research project takes place within this abstraction step. The data is generated in the process step and then forwarded through the output. This allows the origin of the generated data to be precisely determined and to be provided with the corresponding metadata and dependencies, thus enabling its FAIR storage. The generic descriptions of the work steps can now be used and rearranged to form any desired workflow making a further abstraction of the process unreasonable at this point.

Summarising, the applied method illustrated schematically in Figure 3 can be applied to reduce the complexity of arbitrary research processes up to the atomistic descriptions of the individual work steps, that is tools. Consequently, the inverse use of this concept enables the implementation of a generic system for modelling research processes. This modelling is thus based on the general description of the used tools according to the IPO model. Subsequently adding information through connections describing the data flow and the parameterisation of the used tools as described by the APR model, enables the further specification of the process. The concatenation of multiple APR processes finally allows to model the pre-, main- and post-processing, which in sum ultimately represent the complete research process. Similar concepts that use atomistic tool descriptions according to the IPO model which are linked together to form a workflow are used in many well-known workflow management systems. CWL (Crusoe et al. 2022), snakemake (Mölder et al. 2021) and nextflow (Di Tommaso et al. 2017), for example, implement it in a script-based system, while programmes such as KNIME (Berthold et al. 2009) or Orange (Demšar et al. 2013) realise it within in a graphical interface. This widespread use of similar concepts in established systems illustrates the suitability of the found concept for the formulation of workflows.

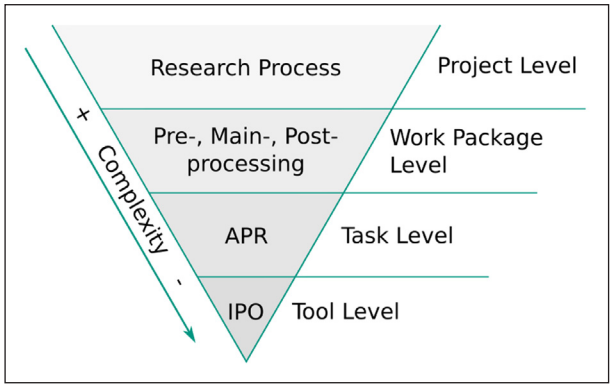

Figure 3 Reducing the process complexity by iteratively applying the IPO model to the identified processes. On each abstraction level, the process can be atomistically described. The tool level description presents a generically usable approach for modelling arbitrary research processes.

The concrete implementation of the described concept into a generic workflow system for the FAIR formulation of research processes, incorporated into the research data infrastructure Kadi4Mat, will be presented in the following.

3 IMPLEMENTATION

The Karlsruhe Data Infrastructure for Materials Science – Kadi4Mat – (Kadi4Mat Team & Contributors 2022b) offers its users multiple functionalities, illustrated in Figure 4. They can be summarised as a *Community Repository* and as an *Electronic Lab Notebook* (*ELN*). While the community repository provides an extensive data sharing and managing infrastructure, the ELN allows for the logging of conducted research, the visualisation, transformation and analysis of stored data, and the generation of reproducible workflows. These workflows serve to model recurring processes in the work of researchers in form of digital twins that allow to process data stored within the repository and to guarantee the reproducibility of said data. The digital twins thus not only facilitate the day to day work of researchers by automation, but also allow to make process knowledge accessible and repurposable for a wider scientific community indicating the importance of their FAIR formulation.

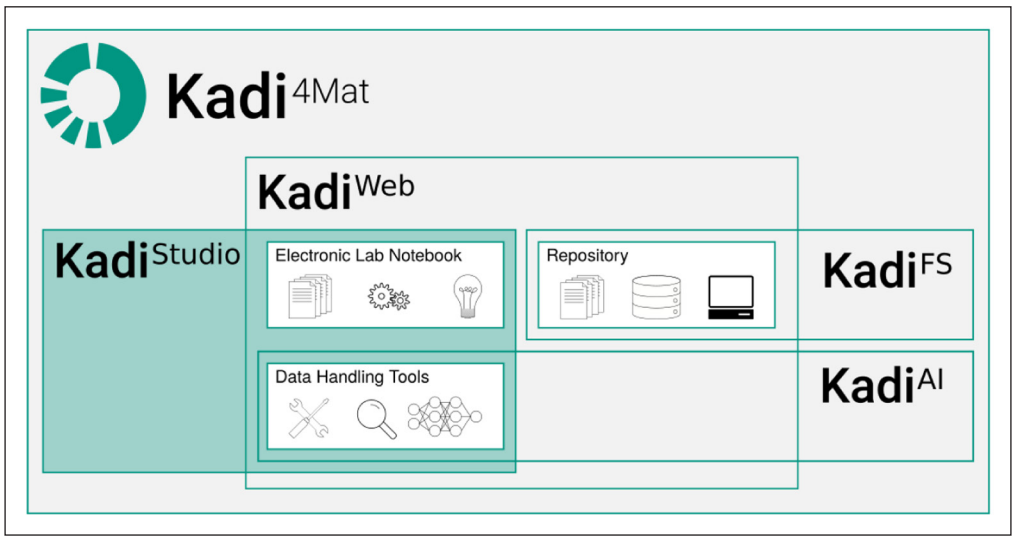

Figure 4 Conceptual overview of Kadi4Mat. Currently two software modules are available: (1) KadiWeb, a web-based virtual research environment incorporating ELN functionalities and repositories and (2) KadiStudio, a desktop-based software version which allows for the formulation and execution of workflows. Further modules such as a machine learning implementation referred to as KadiAI and a desktop-based repository called KadiFS are subject of current developments.

Aiming to provide a user friendly software solution that incorporates the FAIR modelling of research processes, a workflow editor, which is based on an open source node editor library for the Qt GUI framework (Dmitry 2017), has been created and integrated within the ELN functionality of Kadi4Mat. Basis for this workflow system was the process structuring concept presented in the previous section. Within the framework of Kadi4Mat two versions of this editor exist, which both use the same JSON-based data format to describe workflows, ensuring their interoperability. On the one hand a desktop-based, standalone software version called **KadiStudio** exists, which can be used without an internet connection or running web

server, and on the other hand, a web-based version that is integrated into **KadiWeb**. KadiWeb (available under https://kadi.iam-cms.kit.edu/) refers to the generally accessible web version of Kadi4Mat, which incorporates both the community repository and the ELN functionalities including its built-in data handling tools. In Figure 4 the structure of Kadi4Mat is illustrated schematically. Apart from the described components – KadiWeb and KadiStudio – additional modules are currently being developed, including a machine learning tool set called **KadiAI** and a filesystem integration for the repository referred to as **KadiFS**. Generally, Kadi4Mat can be understood as an overarching concept, that encompasses various modules, which in sum, create a generic research data infrastructure, extensible to all kinds of research disciplines in the future.

To illustrate how the workflow system implemented into the infrastructure of Kadi4Mat enables FAIR modelling of research processes, the term FAIR will first be defined in more detail. FAIR was introduced by Wilkinson et al. (2016) and is the acronym for Findable, Accessible, Interoperable, and Reusable. These principles impose the following requirements on the storage of scientific data:

- **F**indable: Data must be provided with descriptive metadata that can be searched specifically by humans and machines alike.
- **A**ccessible: Stored data must be accessible, possibly with appropriate authentication or authorisation.
- **I**nteroperable: Data must be formulated in a broadly applicable language and thus be interoperable with applications and workflows.
- **R**eusable: Data should be reusable. For this, metadata and data need to be rich in information and associated with a detailed provenance.

When applying these principles to scientific processes, however, their definitions have to be partially adjusted. While the wording of Findable and Accessible can be adopted, the elements Interoperable and Reusable need to be adapted. The principle Interoperable must additionally imply that various generally accepted data formats can be used in a modelled workflow. When necessary, an existing workflow must be adaptable accordingly. Moreover, according to the interpretation of the Reusable principle described in (Draxl &Scheffler 2020), a workflow must, on the one hand enable the reliable reproduction of results and, on the other hand, be fully or partially repurposable for different use cases. The conditions Findable and Accessible are already fulfilled in the case of the workflow system presented here through the direct integration in Kadi4Mat. This allows workflows to be stored in the community repository and thus to be shared with the scientific community. As for ordinary research data, the workflows stored in Kadi4Mat can be equipped with descriptive metadata, that can be selectively searched. Consequently, these principles need not to be specifically considered in the implementation of the actual workflow system. The technical implementation of the workflow editor presented hereafter, therefore, concentrates on the interoperability and reusability of the modelled workflows. Additionally, the requirements of the editor to be generic and simple in use (Pizzi et al. 2016) are taken into consideration.

4 RESULTS

The scheme described in Chapter 2 for the abstraction of scientific processes, defines individual functions or tools described according to the IPO model as the basic building blocks of a process. These basic building blocks are implemented in KadiStudio in the form of various nodes.

In both workflow editors — standalone and web-based version —, these nodes can be added and connected to model a process within a graphical user interface (GUI) using an intuitive point-and-click mechanism as shown in Figure 5.

DOI: 10.5334/dsj-2022-016

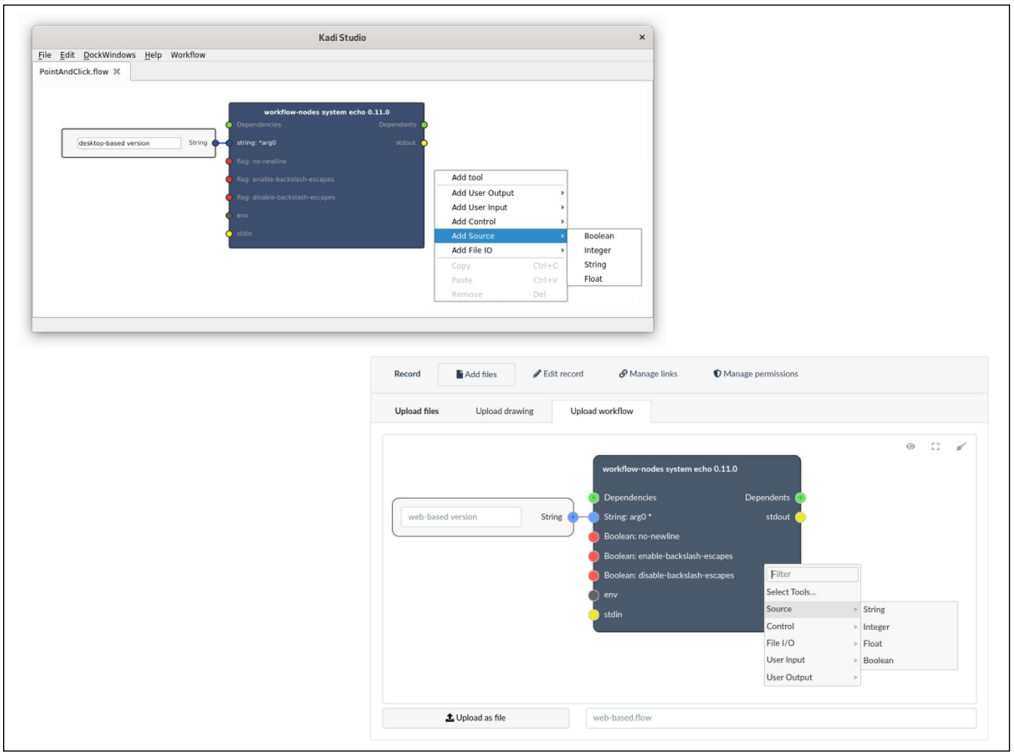

Figure 5 Overview of the available workflow editors, showing the GUI of the desktop (top) and the web-based version (bottom). Workflows can be modelled by adding and connecting nodes using a point-and-click interface.

Each of the insertable nodes represents a certain process, modelled according to the IPO model and can be differentiated between (1) tool nodes, which serve to integrate various programs or functions, (2) environment nodes, which are used in combination with tool nodes, and finally (3) built-in nodes, which allow to influence the execution of the workflow and add interactive options as well as variables to the editor. These node types are presented in Figure 6.

Figure 6 Available node types. Built-in nodes are grey, environment nodes green and tool nodes blue.

In accordance with the IPO model, each node describes a specific process that contains both inputs and outputs represented by the input and output ports respectively. The input ports are located on the left-hand side of the node and the output ports on the right-hand side. Depending on their task, the ports can be divided into parameterisation, dependency, environment, and stdin/stdout ports. Stdin and stdout ports refer to the standard input and standard output streams of the underlying program, respectively. Parameterisation ports are used to pass arguments and options necessary to execute the node, such as string or boolean values. The execution order of the nodes, including control mechanisms such as if-conditions and for-loops, can be defined using the dependency ports. Environment ports are used to set a prefix to a tool node to execute it in a specific environment, such as a secure shell (SSH), that enables the remote execution of tools. Piping the output of a node into another node can further be realised using stdin and stdout ports. The provision of nodes that have these defined

DOI: 10.5334/dsj-2022-016

inputs and outputs is the basis of workflow modelling in KadiStudio. This is in accordance to the identified structure illustrated in Figure 3. Connecting and parameterising the nodes using the named ports allows to add the data flow according to the APR model to the workflow as described in Chapter 2. This puts the added nodes into defined relations and allows the user to see at first glance which inputs a process uses and to which process its output is forwarded. This structures the data flow in a comprehensible manner.

When executing a workflow, the added nodes and their connections are processed and translated into command line interface (CLI) commands. Hence, the workflow editor can be understood as a graphical programming language with simple usability due to its intuitive character, also allowing inexperienced users to model their workflows. The use of this modelling mechanism will be presented in the following examples.

4.1 PARAMETERISATION AND USE OF NODES

As mentioned in the previous section, tool nodes represent CLI commands, structured according to the IPO model, thus possessing defined inputs and outputs. To parameterise the underlying CLI command, the node's input ports are used. A simple example of such a parameterisation is presented in Figure 7.

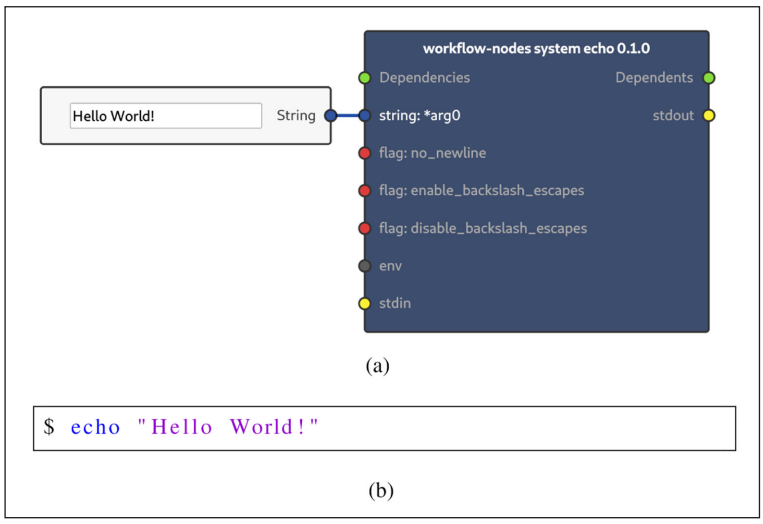

Figure 7 Overview of the node parameterisation. The parameterisation of an *echo* node using a string source node is shown in **(a)**. this is equivalent to the command shown in **(b)**.

The added tool node represents the 'echo' CLI command. Adding a source node of type string and connecting it to the tool node allows for the parameterisation of the echo command. In the presented example, 'Hello World!' is passed to the tool node, resulting in the command shown in Figure 7(b). To pipe a node's output into another node, the stdout and stdin ports are used. Adding a *File Output* node to the workflow example of Figure 7 and connecting it to the tool node as shown in Figure 8(a) activates this piping functionality. As can be seen in the resulting command shown in Figure 8(b), the standard output is forwarded to the second node.

Moreover, the parameterisation of the tool node in Figure 8 is realised using a *UserInput: Text* node that prompts the user for an input when executed. In addition, the dependency ports have also been connected in this example. Defining the dependencies of the workflow nodes supports the process engine in determining the execution order of the added nodes. In the shown example, this implies that the *File Output* node is not executed until the echo node has been called and executed successfully. In general, when modelling a workflow, it is strongly recommended that the dependency ports are connected, as undefined dependencies may result in a wrong execution order, possibly rendering the workflow inoperable.

Figure 8 also vividly illustrates the workflow modelling concept implemented in KadiStudio. The used tools are defined by certain inputs, outputs and a process thus complying with the IPO model. Depending on their purpose they can be assigned to the elements of the APR model. Specifically, the data acquisition consists of the *UserInput: Text* node that prompts the user for an input, which is forwarded to the data processing in form of the echo node by connecting

the ports. After processing this input in the echo node, the result is forwarded to the next node in the subsequent data routing. In this case, the received input is routed into a file using the *File Output* node. This demonstrates the idea of structuring the work steps of each task in the APR model according to their purpose in the present data flow. In the previous examples, the nodes were parameterised using different built-in nodes. On the one hand with source nodes, that provide the computational values *string, boolean, integer*, and *float*, which are set when the workflow is created and remain constant for each execution. On the other hand, by using *user input* nodes, that allow to formulate more generic workflows by granting the possibility to interactively redefine certain parameters during the workflow execution. These nodes pause the execution of the workflow and prompt the user for an input via a dialogue box before continuing the execution, as shown in Figure 9.

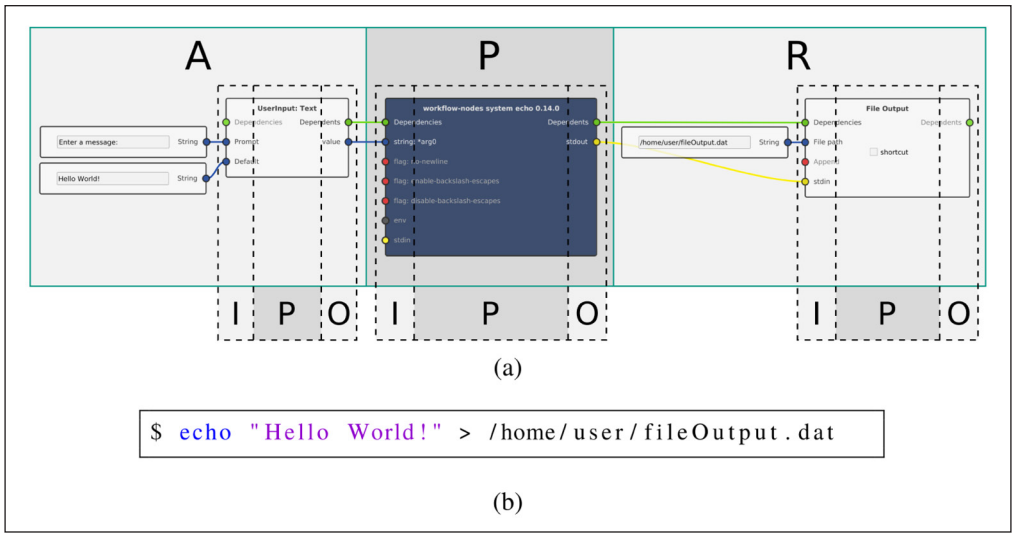

Figure 8 Visualisation of the workflow modelling concept implemented in KadiStudio. Multiple tools are connected according to the APR model forming a simple workflow. Connecting the stdout port to the stdin port as shown in **(a)** pipes the standard output stream of the tool node into the standard input stream of the *File Output* node that finally routes them into a file. This demonstrates structuring the tools of a task depending on their purpose in the data flow. This has the same effect as the command shown in **(b)**.

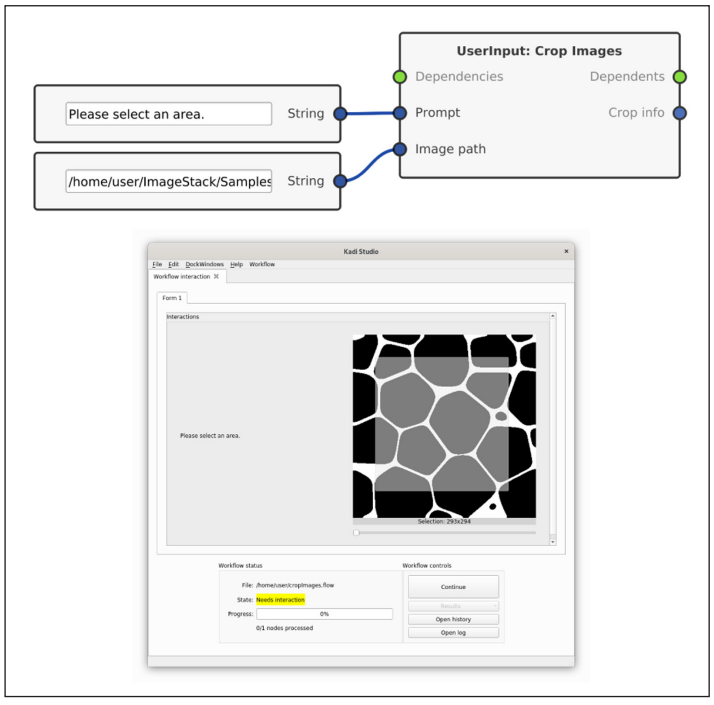

Figure 9 Usage of *UserInput* nodes. During the execution, the user is prompted for an input, such as to select an image area, as shown in this example. The selected area can then be used in further investigations.

Through the use of such user interactions the user is given control over the workflow execution allowing to adapt to different use cases. Workflows are therefore not just predefined scripts that can be applied under certain circumstances, but can be seen as generic tools adaptable to the current use case during execution. The interactively definable parameters are manifold and include not only the query of basic computational values but also the selection of files

and, for example, the cropping of images to a section to be examined, as depicted in Figure 9. Using this prompting mechanism also allows to model manual worksteps in KadiStudio. This is realised by requesting the user to conduct a workstep with certain inputs and querying the results obtained as shown in Figure 10. To ensure the reproducibility of the results obtained in workflows that use such user interactions, the interactively defined user inputs are saved within a log file. When uploading the results to a repository such as Kadi4Mat, the logged user inputs can be used as metadata for the generated data.

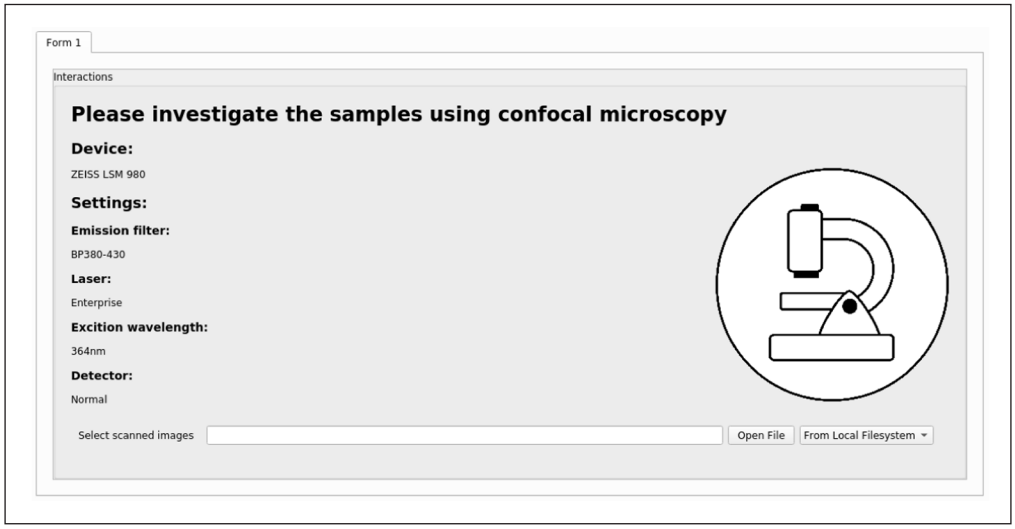

Figure 10 Integrating manual work steps by giving the user all necessary parameters and asking them to select the generated results.

Apart from the interactive nodes, KadiStudio offers various other built-in nodes such as *Variable*, *Loop* or *ifBranch* that allow to manipulate the workflow execution and facilitate the formulation of generic research workflows. The provision of these built-in nodes in the workflow system KadiStudio thus not only guarantees the reproducibility of any manual or digital research process, but also allows for the repurposability of the workflows.

4.2 ADDING NEW NODES

To guarantee the interoperability of modelled workflows and to model the heterogeneous tool landscape present in scientific research, the repertoire of nodes available in KadiStudio must be easily extendable. In this way, custom functionalities and data conversion nodes can be incorporated into the workflow editor, enabling formulation of arbitrary workflows and their application to different file formats for instance. The interface for integrating new tools was therefore kept as simple as possible. Prerequisites for a new node to be added to the editor are only that the underlying CLI-command is (1) executable and (2) provides the *--xmlhelp* option. The *--xmlhelp* option returns a machine-readable description of the command, that is needed by the workflow editor to create the visual representation of the command within the editor. In case the desired tool does not provide this option, it can be added retroactively, for example, with a wrapper script using the *xmlhelpy* Python library (Kadi4Mat Team & Contributors 2022d). Listing 1 of appendix A shows an abbreviated, exemplary implementation of such a Python wrapper for the *echo* command. The XML output generated by this wrapper is shown in Figure 11.

```
<?xml version='1.0' encoding='UTF-8'?>
<program name="workflow-nodes system echo" description="Wrapper node for echo (GNU coreutils)" version="0.11.0">
  <param description="Message to be echoed" type="string" name="arg0" positional="true" required="true"/>
  <param description="Do not output the trailing newline" type="flag" name="no-newline" char="n" default="false"/>
</program>
```

Figure 11 XML output of the wrapper script shown in Listing 1 of appendix A, that is printed when using the *--xmlhelp* option. The root element *program* specifies the command as a regular tool. Each of the *param* elements represents a configurable option of the wrapped command.

The structure of an xmlhelp output always follows the same pattern. After the declaration as an XML document, a root element of type *program* or *env* follows, which indicates a tool or an environment node respectively and is provided with the name, description and version attributes. Within this element, a *param* element is specified for each possible parameter of the command, which must contain the name, description and type attributes in order to

be rendered within the editor. Additionally, further attributes such as a default value can be defined. The definition of these *params* specifies the inputs of the final node and thus serves to represent the underlying process with respect to the IPO model. The tool node derived from the xmlhelp above is shown in Figure 12.

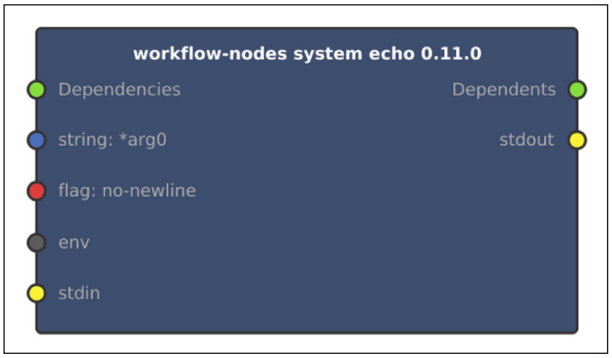

Figure 12 Created echo node. Echo node derived from the XML output shown in Figure 11. Each *param* is represented by an input port. Tool node specific ports such as *env* are added automatically.

The wrapper script shown in Listing 1 of appendix A is part of the workflow-nodes library (Kadi4Mat Team & Contributors 2022c), that already contains various Python-based nodes covering basic as well as some more specialized functions. The tool thus fulfills both prerequisites: (1) *executable*, (2) *--xmlhelp* option and can be added to the editor's tool list using its GUI as shown in Figure 13. In the dialogue every executable within the PATH environment variable is listed. Selecting a tool will permanently add it to the usable tools of the editor.

The provision of the described interface for adding new nodes to KadiStudio enables not only the formulation of arbitrary workflows but also allows for the simple adaptation of existing workflows to different file formats. This contributes to the generic character of KadiStudio and ensures the interoperability of the workflows modelled in it.

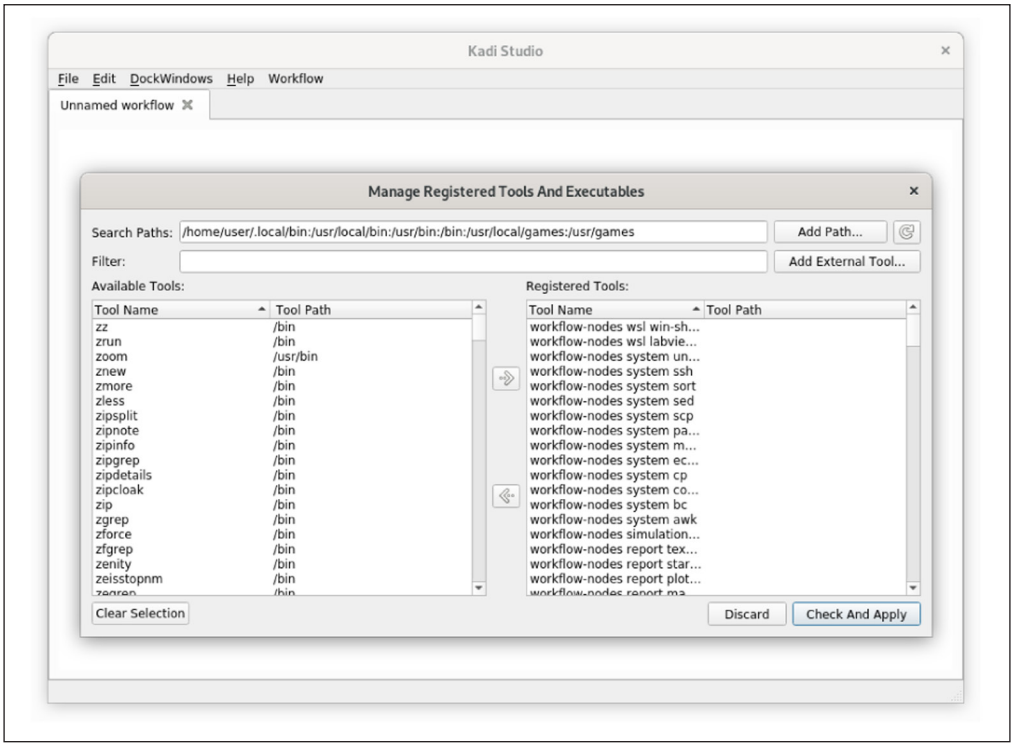

Figure 13 Dialogue for registering tools in the editor of KadiStudio. All executables in the PATH are listed. Commands can be queried and the default search path can be extended by the user.

4.3 LINK BETWEEN KADISTUDIO AND A REPOSITORY

As already mentioned, workflows created in KadiStudio can be stored directly in Kadi4Mat and provided with metadata. This permits the findable and accessible storage of workflows. Since the FAIR idea however does not only apply to workflows but to all scientific data, KadiStudio aims to provide FAIR documentation of the data created during a workflow. For this purpose, a link between the workflow editor and an arbitrary repository can be established. As a reference, this link has already been implemented for the repository of Kadi4Mat in form of tool nodes

collected in the kadi-apy library (Kadi4Mat Team & Contributors 2022a). These nodes access the repository and its functionalities via the application programming interface (API) provided by Kadi4Mat, which offers a set of defined functions and interfaces to interact with *KadiWeb*. Registering a repository to KadiStudio is realised using a graphical user interface as shown in Figure 14. In case of Kadi4Mat this requires its host address as well as a personal access token (PAT).

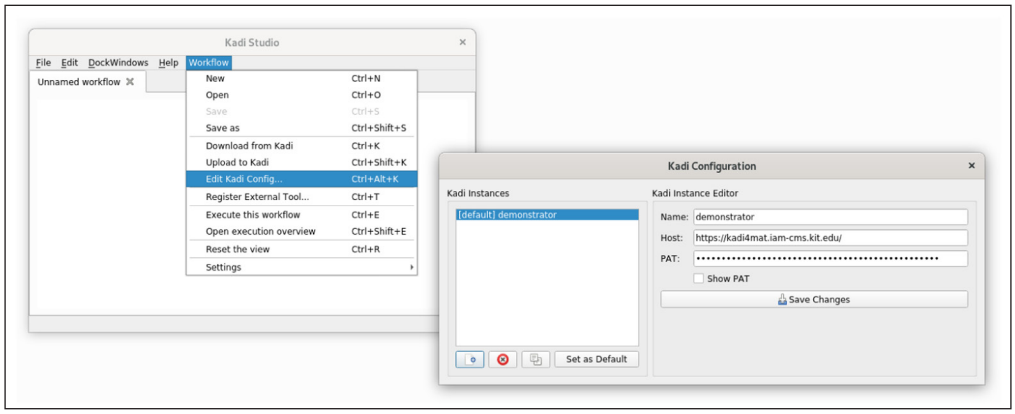

Figure 14 Dialogue for registering Kadi4Mat instances. Registered instances can be accessed during a workflow to use the functionalities of the repository and to exchange data.

Linking the local editor with the Kadi4Mat repository opens up various possibilities. Workflows saved in the repository can be loaded directly into the local editor to be edited. In addition, data stored in the repository can be loaded and processed within workflows and the corresponding results can be uploaded automatically to Kadi4Mat and provided with descriptive metadata. Establishing a link to a repository such as Kadi4Mat thus allows to holistically model research processes and the data used therein in a comprehensible manner. In summary, KadiStudio's access to the data allows to map the entire scientific work from the raw or source data, through its analysis and use up to the structured storage within the repository. Structuring the generated data in Kadi4Mat and defining their interrelationships further enables their provenance tracking. In this way KadiStudio provides the possibility for FAIR handling of data generated during a workflow.

5 TECHNICAL ASPECTS

The technical details of the workflow editor, its architecture and related software components are presented in the following. The focus of these developments was again on the objective of creating a generic and intuitive software system, that allows for the FAIR formulation of research processes.

5.1 XMLHELPY

The generic character of the workflow editor is partly implemented by supporting the simple extension of the available tools. To incorporate a new tool, it must provide the *--xmlhelp* option, that returns a machine-readable description of the tool. Adding this option to existing tools can be realised using the xmlhelpy library. Xmlhelpy is a Python library available on *PyPI*. It is based on the open source framework Click (The Pallets Projects 2014), and extends it with custom classes and functions. As other related libraries for argument parsing, xmlhelpy allows to conveniently specify a program's command line arguments and takes care of parsing and validating them. It then automatically adds the *--xmlhelp* option to the program allowing to obtain its specification in a machine-readable format.

5.2 ARCHITECTURE FOR WORKFLOW EXECUTION

Figure 15 gives an overview of the workflow system's architecture. As mentioned in the previous sections, two versions of the workflow editor exist, which both use the same JSON-based file format to load and save workflows. For execution, the workflow files are managed by the process manager, a software component dedicated to providing a unified interface for workflow execution. The process manager does not read or analyse the workflow files by itself but only acts as a thin layer of abstraction for the GUI components using its interface.

Instead, it passes the workflow files on to a suitable process engine and instructs it with the execution. This way, the process engine implementations can be exchanged easily, enabling the flexible adaptation of the workflow system to changing requirements, thus strengthening the generic character of the Kadi4Mat workflow system. For this, a variety of process engines with specific characteristics suitable for a certain use case—remote execution, parallel execution, distributed execution, or running on large computer clusters/high performance computers—or workflow file format can be registered in the process manager. It is therefore conceivable to integrate other programs for workflow execution, such as Fireworks (Jain et al. 2015), in form of a specialised process engine. As a reference implementation, a fully functional process engine is provided (Zschumme, Schoof, et al. 2022), which follows a sequential execution approach. A detailed description of this process-engine implementation and of the process-manager is given in the following sections.

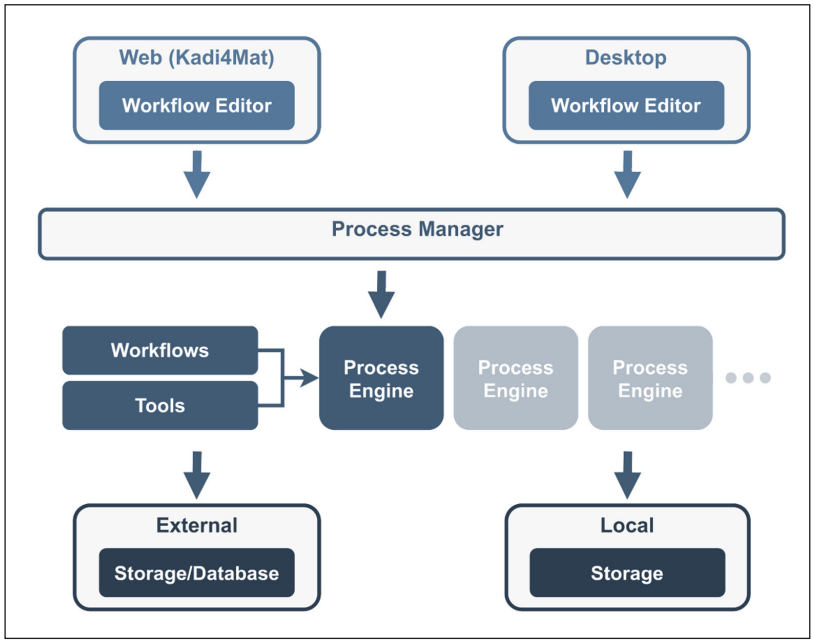

Figure 15 Architecture of the workflow system. Based on Figure 3 (Brandt et al. 2021) (CC BY 4.0). The process manager orchestrates and monitors the execution of workflows by delegating it to a suitable process engine and tracking its status.

5.3 PROCESS ENGINE

The term process engine refers to a software component dedicated to running workflows and performing all tasks formulated in the workflow. As shown in Figure 15, the general concept takes into account that several different implementations might be available to flexibly adapt the execution to different technical emphases or capabilities.

Since each process engine implementation is free in how it executes the workflow, there is a wide field of possibilities, ranging from executing the workflow locally to delegating the execution to another application or computer to just printing a workflow description in a format like common workflow language (CWL) (Crusoe et al. 2021) or workflow description language (WDL) (Frazer et al. 2012). Prototype implementations of process engines running Kadi4Mat workflows with Fireworks (Jain et al. 2015) and CWL (Crusoe et al. 2021) have already been implemented. These promising approaches are to be continued in the future and could expand the current possibilities.

At the moment there is one fully functional process engine implementation (Zschumme, Schoof, et al. 2022), which is a CLI-based application written in C++, published under the Apache-2.0 license. It serves as a reference implementation and allows local execution of workflows created with any of the workflow editors presented in Chapter 3. Its schematic functioning is outlined in the following.

When instructed to execute a workflow, the process engine first reads the workflow file and parses its JSON-structured content. During this step the process engine creates an internal in-memory representation of the workflow based on the unsorted lists of node descriptions and connections contained in the workflow file. An important aspect of this analysis is the determination of the execution order. For this, the data connections in the workflow as well as

the explicit dependency connections are considered. The dependency connections also help with identifying conditional execution paths implied by *If Branch* or *For Loop* nodes for example.

After determining the correct order of execution, the implemented process engine executes each node of the workflow sequentially. This sequential and separate execution requires the nodes to provide its own execution function, which can be called as soon as the process engine decides to run the corresponding node. In order to manage the workflow execution, the implemented process engine uses a number of control files that are stored in an execution folder unique to every execution. Within these files, information about the progress of the execution, log files, and other necessary information is stored. This central storage of all relevant information eases debugging and ensures the reproducibility of the obtained results. When all nodes have been processed successfully, the execution is completed.

5.4 PROCESS MANAGER

The process manager is a CLI-based application written in C++ published under the Apache-2.0 license (Zschumme, Steinhülb & Brandt 2022). As shown in Figure 15, it provides an interface for the GUI components to run workflows and to monitor and interact with already existing execution instances. It further administers the different process engines, which can be configured in a JSON-based configuration file that contains information on how to use the interface of each engine. When executing a workflow, the user can select an engine suitable for the current use-case from this list of available engines. In case the user omits this choice, the process manager instructs the default process engine with the workflow execution.

Once a workflow is started, the process manager will interact with the responsible process engine to obtain the status and log of the execution or to provide user input. For this, the process manager assigns a unique identifier to each execution instance and creates an **execution folder** which is used exclusively during this particular execution. The process engine is then free to use this provided, empty folder for storage. This eliminates the risk of overwriting important data from previous executions, which is why the execution folder is used as the base path for all executed programs by the current process engine implementation (Zschumme, Schoof, et al. 2022). Depending on the internals of the workflow, however, different save paths can be used as well.

6 CONCLUSION

In this paper, a concept for the FAIR formulation of research processes was presented and its concrete implementation in form of the workflow system KadiStudio introduced. The implemented software allows for the FAIR modelling of scientific research processes in form of automatable and reproducible workflows. Basis for this was the identification of a generic structure in research processes that can be used for their modelling. For this purpose, the input-process-output model was iteractively applied to research processes until a structure emerged, that enables the bottom-up modelling of any process. The found structure abstracts processes as the concatenation of various functions or programs that are defined by certain inputs, a process, and outputs. The interconnection and parameterisation of these programs subsequently allows to model the data flow within the research process comprehensibly. Stringing all work steps of a workflow together through the said connections finally models the complete workflow. Using this method, arbitrary workflows can be created. The atomistic process description on different levels further permits the identification of reusable elements in workflows. Thus, using workflows research results can not only be made reproducible, but the used methods can also be applied for other purposes. This reuse of certain procedures additionally minimises the susceptibility to errors in scientific workflows and thereby guarantees the quality assurance of scientific data.

In KadiStudio the identified process structure is implemented by providing different nodes, each of which represents a certain function or program. By adding and connecting them in a graphical user interface using an intuitive point-and-click mechanism, research processes can be modelled. To guarantee the FAIR formulation of workflows in KadiStudio, certain design choices have been made and specific functionalities have been included. The aspects findable and accessible are ensured by directly integrating KadiStudio into Kadi4Mat, thus allowing the

structured storage and management of workflows within its repository. This also applies to the scientific data generated in workflows, which can be automatically provided with descriptive metadata and stored in Kadi4Mat in a FAIR manner. The interoperability of workflows is further guaranteed by giving the user the possibility to add and adapt nodes in KadiStudio. In this way adaptions necessary to process different file formats can be easily implemented into existing workflows. Through the simple extension of the available tools the heterogeneous tool landscape present in science can additionally be represented in KadiStudio. Hence, the workflow editor can be used in any scientific domain to model the occurring processes. This also incorporates manually performed worksteps, which can be integrated through the provided user interaction nodes and prompts. These interactive elements additionally enable the formulation of generic workflows that can be applied to different use cases therefore ensuring the repurposablility of modelled workflows. Logging all inputs provided during the execution of a workflow finally realises the reproducibility of a workflow execution as well as of all data generated within it.

Summarising, KadiStudio offers an easily adaptable and intuitively usable solution to holistically model arbitrary research processes and the data generated therein in a FAIR manner. The implemented system is simple in use and can be flexibly adapted to any scientific domain, therefore also satisfying the requirements to a workflow system formulated by (Pizzi et al. 2016). KadiStudio thus makes a decisive contribution to promoting FAIR handling of data and scientific processes and therefore to the realisation of the fourth scientific paradigm.

APPENDIX A

Listing 1 Echo node example.

```
import subprocess
import sys

from xmlhelpy import argument
from xmlhelpy import option

# Decoration with @command defines the node type as 'tool'.
# Using @environment would set the node type to environment.
@command(version="0.1.0")
# @argument and @option add new ports to the tool node,
# which serve to parameterise the underlying funcion.
@argument("message", description="Message to be echoed")
@option(
    "no-newline",
    char="n",
    is_flag=True,
    description="Do not output the trailing newline",
)
# The echo function is executed when the node is called.
def echo(message, no_newline):
    """Wrapper node for echo (GNU coreutils)."""
    cmd = ["echo"]

    if no_newline:
        cmd.append("-n")

    cmd.append(message)
    sys.exit(subprocess.run(cmd).returncode)
```

ACKNOWLEDGEMENTS

This work is partly funded by the German Research Foundation (DFG) under Project ID 390874152 (POLiS Cluster of Excellence), by the German Federal Ministry of Education and Research (BMBF) in the project FB2 TheoDat (project number 03XP0435D), by the Ministry of Science, Research and Art Baden-Württemberg (MWK-BW) in the project MoMaF–Science Data Center, with funds from the state digitization strategy digital@bw (project number 57), by the Helmholtz association in the project INNOPOOL MDMC (program No. 43.35.01) and also funded by the BMBF and MWK-BW as part of the Excellence Strategy of the German Federal and State Governments in the project Kadi4X. We would also like to acknowledge the German Federal Ministry of Education and Research (BMBF) for its financial support within the project AQuaBP, under the grant number 03XP0315B. Some ideas presented in this paper are enhanced by the fruitful discussions in different working groups of the project NFDI4Ing.

AUTHOR AFFILIATIONS

Lars Griem orcid.org/0000-0002-8093-6356
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany

Philipp Zschumme orcid.org/0000-0002-4821-5719
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany

Matthieu Laqua orcid.org/0000-0003-2978-3160
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany

Nico Brandt orcid.org/0000-0002-3860-1376
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany

Ephraim Schoof orcid.org/0000-0001-6821-7263
Helmholtz Institute Ulm for Electrochemical Energy Storage (HIU), Helmholtzstraße 11, 89081 Ulm, Germany

Patrick Altschuh orcid.org/0000-0003-1373-492X
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany; Institute for Digital Materials Science (IDM), Karlsruhe University of Applied Sciences, Moltkestraße 30, 76133 Karlsruhe, Germany

Michael Selzer orcid.org/0000-0002-9756-646X
Institute for Applied Materials (IAM-MMS), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany; Institute for Digital Materials Science (IDM), Karlsruhe University of Applied Sciences, Moltkestraße 30, 76133 Karlsruhe, Germany

COMPETING INTERESTS

The authors have no competing interests to declare.

REFERENCES

Afgan, E, et al. 2018. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic acids research,* 46(W1): W537–W544. DOI: https://doi.org/10.1093/nar/gky379

Berthold, MR, et al. 2009. KNIME-the Konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1): 26–31. DOI: https://doi.org/10.1145/1656274.1656280

Brandt, N, et al. 2021. Kadi4Mat: A Research Data Infrastructure for Materials Science. *Data Science Journal* 20.1. DOI: https://doi.org/10.5334/dsj-2021-008

Crusoe, MR, et al. 2021. Methods included: standardizing computational reuse and portability with the common workflow language. *arXiv preprint arXiv, 2105.07028*. DOI: https://doi.org/10.1145/3486897

Crusoe, MR, et al. May 2022. Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language. *Commun. ACM,* 65(6): 54–63. DOI: https://doi.org/10.1145/3486897

Demšar, J, et al. 2013. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research,* 14: 2349–2353.

Di Tommaso, P, et al. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology,* 35(4): 316–319. DOI: https://doi.org/10.1038/nbt.3820

Dmitry, PEA. 2017. *Qt5 Node Editor*. https://github.com/paceholder/nodeeditor.

Draxl, C and **Scheffler, M.** 2020. Big data-driven materials science and its FAIR data infrastructure. *Handbook of Materials Modeling: Methods: Theory and Modeling*, 49–73. DOI: https://doi.org/10.1007/978-3-319-44677-6_104

Frazer, S, et al. 2012. *Workflow Description Language – Specification and Implementations*. https://libraries.io/github/openwdl/wdl.

Goel, A. 2010. *Computer fundamentals*. Pearson Education India.

Hey, AJ, Tansley, S, Tolle, KM, et al. 2009. *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. WA: Microsoft research Redmond.

Jain, A, et al. 2015. FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience,* 27(17): 5037–5059. DOI: https://doi.org/10.1002/cpe.3505

Kadi4Mat Team and Contributors. June 2022a. *IAM-CMS/kadi-apy: Kadi4Mat API Library*. Version 0.23.0. DOI: https://doi.org/10.5281/zenodo.6623518

Kadi4Mat Team and Contributors. June 2022b. *IAM-CMS/kadi: Kadi4Mat*. Version 0.25.1. DOI: https://doi.org/10.5281/zenodo.6623521

Kadi4Mat Team and Contributors. July 2022c. *IAM-CMS/workflow-nodes*. Version 0.15.0. DOI: https://doi.org/10.5281/zenodo.6806747

Kadi4Mat Team and Contributors. February 2022d. *IAM-CMS/xmlhelpy*. Version 0.9.2. DOI: https://doi.org/10.5281/zenodo.5971732

Kadi4Mat Team and Contributors. July 2022e. *kadistudio: 0.1.0.alpha1*. Version 0.1.0.alpha1. DOI: https://doi.org/10.5281/zenodo.6810891

Kluyver, T, et al. 2016. Jupyter Notebooks? A publishing format for reproducible computational workflows. In: Loizides, F and Scmidt, B (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press. pp. 87–90. DOI: https://doi.org/10.3233/978-1-61499-649-1-87

Mölder, F, et al. 2021. Sustainable data analysis with Snakemake. *F1000 Research*, 10. DOI: https://doi.org/10.12688/f1000research.29032.1

Pizzi, G, et al. 2016. AiiDA: automated interactive infrastructure and database for computational science. *Computational Materials Science*, 111: 218–230. DOI: https://doi.org/10.1016/j.commatsci.2015.09.013

The Pallets Projects. 2014. *Click – The Pallets Projects*. https://palletsprojects.com/p/click/.

Wilkinson, MD, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3(1): 1–9. DOI: https://doi.org/10.1038/sdata.2016.18

Zelle, JM. 2004. *Python programming: an introduction to computer science*. Franklin, Beedle & Associates, Inc.

Zschumme, P, Schoof, E, et al. July 2022. *IAM-CMS/process-engine*. Version 0.5.0. DOI: https://doi.org/10.5281/zenodo.6806707

Zschumme, P, Steinhülb, J and **Brandt, N.** February 2022. *IAM-CMS/process-manager*. Version 0.2.0. DOI: https://doi.org/10.5281/zenodo.5972885

TO CITE THIS ARTICLE:
Griem, L, Zschumme, P, Laqua, M, Brandt, N, Schoof, E, Altschuh, P and Selzer, M. 2022. KadiStudio: FAIR Modelling of Scientific Research Processes. *Data Science Journal*, 21: 16, pp. 1–17. DOI: https://doi.org/10.5334/dsj-2022-016

Submitted: 08 July 2022
Accepted: 06 September 2022
Published: 23 September 2022

Data Science Journal is a peer-reviewed open access journal published by Ubiquity Press.