**AiiDA**

AiiDA is a robust open-source high-throughput infrastructure addressing the challenges arising from the needs of automated workflow management and data provenance recording. AiiDA’s workflow language provides advanced automation, error handling features and a flexible plugin model to allow interfacing with external simulation software. The associated plugin registry enables seamless sharing of extensions, empowering a vibrant user community dedicated to making simulations more robust, user-friendly and reproducible.

Architecture Overview

AiiDA aims to provide a framework that enables designing and running complex high-throughput computational workflows with full automatic provenance and built-in support for high-performance computing on remote supercomputers. The architecture, as shown in Fig. 1, is designed with these goals in mind.

![Diagram

Description automatically generated]()

Figure 1 Schematic overview of the architecture of AiiDA 1.

One of the core components, the engine, is responsible for running all calculations and workflows that are submitted by the user Calculations and workflows can be implemented in the custom language provided by AiiDA’s core API which is implemented in Python.

Any calculation or workflow that is run by the engine will be automatically recorded in the provenance graph in order to enable the reproducibility of the results. Besides the workflow language, the ORM also provides the tools to interact with the nodes of the provenance graph and inspect their content. The QueryBuilder is the tool that allows efficient traversal of the provenance graph to select (sets of) nodes of interest.

The contents of the provenance graph are stored in a file repository on the local file system and a relational database. The mapping between the database and the Python API is performed by an Object Relational Mapper (ORM): currently the user can choose between the Django (djangoproject.com) or SQLAlchemy (sqlalchemy.org) library.

From the outside, users can interact with AiiDA through a command line interface called verdi, an interactive Python shell or normal Python scripts. The REST API allows one to query the provenance graph through HTTP calls (see section “The REST API” for more details). AiiDA itself can communicate with computing resources either locally or over SSH to run calculations on those resources and comes with built-in support for most well-known and used job schedulers.

Engine and Workflow Language

The engine is the component of AiiDA in charge of automating the execution of calculations and workflows. AiiDA provides a workflow language to define the logic to run complex sequences of steps, with potentially nested subworkflows and calculations.

The engine consists of runners that are executed in parallel as different operating system processes, supervised by a daemon that monitors and relaunches them if they were to die. Each runner can process tasks independently and concurrently, distributing the workload involved in workflow and calculation execution. Task distribution is achieved via a task queue implemented using the AMQP protocol through RabbitMQ, to guarantee reliable scheduling and almost instantaneous reaction to events such as the request for the submission of a new calculation or workflow, or continuing a workflow when the calculations or subworkflows it depends upon are completed.

Thanks to this scalable architecture, the AiiDA engine is able to sustain high-throughput workloads involving tens of thousands of concurrent tasks every hour distributed on multiple computational resources. Additionally, during execution AiiDA automatically stores all data and actions in the provenance graph, including the workflows, the calculations and their inputs and outputs, to provide full traceability.

.

Processes in AiiDA

In AiiDA, any entity that handles input data to produce output data, and that is run by the engine, is called a process. Processes come in two flavours: calculations and workflows In particular, calculations are defined as processes that create new data as output, given certain data as input. A typical case is the execution of a simulation code on a remote computer. In contrast, workflows in AiiDA are solely tasked with the orchestration of subprocesses, calling calculations and/or other workflows in a certain logical sequence. Consequently, workflows are not allowed to generate new data, but can only return existing data.

This distinction is critical in the design of the provenance model of AiiDA, allowing to differentiate the part of the provenance graph that represents exactly how data was generated from the logical provenance that captures the reason behind the data creation.

**The Provenance Model**

AiiDA’s engine automatically represents the execution of processes, along with their inputs and outputs, as nodes of a directed graph. Links connect the nodes; process nodes. Since output data can in turn be used as input to new processes, extensive graphs are generated. They are called AiiDA provenance graphs, as they allow to retrace the exact steps that led to the creation of a piece of data. An example of a simple provenance graph is shown in Fig. 2.

![Diagram, shape

Description automatically generated]()

Figure 2

(a) A schematic provenance graph representing the execution of a workflow W1 receiving three data nodes D1, D2 and D3 as input, containing the values x, y and z respectively. W1 computes the expression (x + y) · z by calling two calculations C1 (to perform the sum) and C2 (to perform the product), forwarding the correct inputs to them. C1 creates the intermediate node D4 (with the value x + y) and C2 then creates the node D5 with the final result, that is then also returned by W1. While this simplified example is purely for illustrative purposes, it demonstrates that by storing execution information as a graph, the provenance of all data is fully recorded.

(b) Data-provenance layer: it includes calculation and data nodes only, showing the exact sequence of steps that led to the creation of the data nodes.

(c) Logical-provenance layer: it hides the details of all intermediate results and focuses only on how the workflow produced the final results from a given set of inputs.

Node Types

While all nodes of the graph share a set of common properties, there is a need to define custom properties based on what the node represents. Therefore, the various AiiDA processes as well as data are represented by different node subtypes. This makes it possible to implement functionality specific to each of them and to explicitly target nodes of a certain type when querying the graph.

In AiiDA, the Node class is the base class to represent any node in the graph. The common properties of any node include the user who created it, the creation and last modification times, an optional computer on which it was run or stored, and a human-readable label and description. Node classes are subclassed to build a hierarchy of node types, schematically represented in Fig. 3. In particular, data and process nodes are represented by the Data and ProcessNode subclasses, respectively.

![Diagram

Description automatically generated]()

Figure 3 Hierarcy of node types in AiiDa

The different node classes allow to implement custom functionality for each subtype. Additionally, the subclass hierarchy allows to query for specific node types, or a set thereof.

The hierarchy of *ProcessNode* subclasses reflects the distinction in AiiDA between calculations and workflows, represented by subclasses of *CalculationNode* or *WorkflowNode*, respectively. In practice, in a provenance graph one finds instances of CalcJobNode, *CalcFunctionNode*, *WorkChainNode* or *WorkFunctionNode*, representing executions by the engine of the corresponding process classes. The intermediate classes in the hierarchy serve mostly as a taxonomic classifier and they are useful when querying the provenance graph.

Link Types

All links have a type to indicate the semantic meaning of the relationship. In addition, links have a label that can be used, given a node, to distinguish nodes connected to it with the same link type. For example, labels identify the different input nodes to a process. A summary of all link types in AiiDA is shown in Fig. 4.

Diagram, schematic

Description automatically generated

Figure 4 Link types allowed in the AiiDa provenance graph

Rectangles represent node types and arrows connecting them indicate the direction and the type of each link. The symbols at the start and end of each arrow indicate the cardinality of the corresponding link types: 0..1 means that at most one node is allowed on that link endpoint for a given node on the opposite endpoint. Additionally, a dagger (†) indicates that link labels must be unique for a given node on the opposite endpoint.

Process nodes can have input and output links to data nodes, representing their inputs and outputs. More specifically, in the implementation, input links can be of type *INPUT\_CALC* and *INPUT\_WORK* depending on the type of the linked process node. Similarly, output links can either be of type *CREATE* or *RETURN* explicitly highlighting the difference between calculation and workflow processes.

Node properties

AiiDA provides two data stores: a filesystem repository and a relational database (where any JSON-serialisable key-value pair can be saved. Properties that are stored in the database are named attributes, which are fully and efficiently queryable. In contrast, properties that do not require querying and/or are very large in size, such as large arrays or raw files, are better stored in the repository so as not to overburden the database. Attributes and the files in the repository are immutable once the node is stored, since together they define the “content” of the node and allowing them to be changed would invalidate the provenance of descendants. Mutable properties are also allowed and are called extras that, like attributes, are stored as key-value pairs in the database. However, in stark contrast to attributes, extras can be added and/or modified at any time. A typical use case is to tag nodes with custom properties that can, for example, be used for more selective querying.

The FAIR Guiding Principles

The FAIR Guiding Principles describe distinct considerations for contemporary data publishing environments with respect to supporting both manual and automated deposition, exploration, sharing, and reuse. In principal, FAIR describes concise, domain-independent, high-level principles that can be applied to a wide range of scholarly outputs. [1]

The FAIR Guiding Principles describe distinct considerations for contemporary data publishing environments with respect to supporting both manual and automated deposition, exploration, sharing, and reuse.

The elements of the FAIR Principles are related, but are independent and separable. Moreover, the Principles define characteristics that contemporary data resources, tools, vocabularies and infrastructures should exhibit to assist discovery and reuse by third-parties, such as data publishers or stewards.

Hence, they are not, themselves, a standard or a specification. They act as a guide to data publishers and stewards to assist them in evaluating whether their implementation choices are rendering their digital research artefacts Findable, Accessible, Interoperable, and Reusable.

|  |  |
| --- | --- |
| **Findable** | (meta)data are assigned a globally unique and persistent identifier  data are described with rich metadata (defined by R1 below)  metadata clearly and explicitly include the identifier of the data it describes  (meta)data are registered or indexed in a searchable resource |
| **Accessible** | (meta)data are retrievable by their identifier using a standardized communications protocol  the protocol is open, free, and universally implementable  the protocol allows for an authentication and authorization procedure, where necessary  metadata are accessible, even when the data are no longer available |
| **Interoperable** | (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.  (meta)data use vocabularies that follow FAIR principles  (meta)data include qualified references to other (meta)data |
| **Reusable** | meta(data) are richly described with a plurality of accurate and relevant attributes  (meta)data are released with a clear and accessible data usage license  e(meta)data are associated with detailed provenance  (meta)data meet domain-relevant community standards |

[1]. Wilkinson et al 2018

[2] Jens Bröder, Daniel Wortmann, Stefan Blügel, JuCLS database of core-level shifts from all-electron density functional theory simulations for chemical analysis of X-ray photoelectron spectra, Materials Cloud Archive 2020.139 (2020), doi: 10.24435/materialscloud:3j-p3.