The logo for 'Open Model' features the word 'Open' in a blue circle with a white 'O', followed by 'pen Model' in a grey, rounded font. Above the text are three colored circles (green, yellow, orange) connected by a blue arc. The entire graphic is framed by two blue curved lines at the top and bottom.

# Open Model

LOUIS PONET (EPFL)

**WP4-D4.4-EXEFLOW DEPLOYMENT BASED ON  
VANILLA AIIDA INTEGRATION OF CUDS**

# D4.4:EXEFLOW DEPLOYMENT BASED ON VANILLA AIIDA INTEGRATION OF CUDS

## DOCUMENT CONTROL

Document Type	Deliverable Report
Status	Final
Version	1.0
Responsible	Louis Ponet (EPFL)
Author(s)	Louis Ponet (EPFL)
Release Date	2022-02-14, 2023-01-24

## ABSTRACT

OpenModel will initially be focused on using AiiDA as a workflow executor. To this end an interface between OpenModel and AiiDA is being developed. Since AiiDA has an internal data representation of data and processes that mutate and create data, the first and maybe most important part of the interface is to establish mapping between the OpenModel Common Universal DataStructure (CUDS) and AiiDA DataNodes. The final architecture for the OpenModel CUDS is not yet in place, making it necessary to develop a way to generate a semantic representation of any AiiDA DataNode, which can subsequently form the basis of further OpenModel CUDS architecture decisions. For the time being, DLite was chosen as the library to semantically represent AiiDA DataNodes.

## CHANGE HISTORY

Version	Date	Comment
0.1	2022-02-14	First Draft, Louis Ponet
0.2	2022-02-20	Louis Ponet
1.0	2022-02-28	Final, Louis Ponet
1.0	2022-02-28	Review by technical Manager
1.0	2023-01-24	Added Request from EC

**DISSEMINATION LEVEL**

PU	Public	<b>X</b>
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

## TABLE OF CONTENT

Document Control.....	1
Abstract.....	1
Change History .....	1
Dissemination level .....	2
Table of Content .....	3
List of Figures .....	4
1 INTRODUCTION .....	5
1.1 DLITE [1].....	5
1.2 Aiida [3] .....	6
2 Aiida-cuds [4].....	6
2.1 DEMONSTRATING EXAMPLES.....	7
2.1.1 EXAMPLE 1: KPOINTS DATA .....	7
2.1.2 EXAMPLE 2: ORBITAL DATA.....	8
3 BIBLIOGRAPHY.....	11
4 ACKNOWLEDGMENT .....	12

## LIST OF FIGURES

Figure 1: A screenshot from of the aiida-cuds GitHub repository. The actual AiiDA-CUDS interface code can be found in the aiida-cuds subdirectory, static datamodels in the entities subdirectory and scripts for running and testing the interfaces in the scripts subdirectory. .... 7

# D4.4-EXECFLOW DEPLOYMENT BASED ON VANILLA AIIDA INTEGRATION OF CUDS

## 1 INTRODUCTION

The goal of this deliverable is to develop an interface between semantically annotated OpenModel CUDS and AiiDA DataNodes. These DataNodes are stored in an internal AiiDA DataBase which is usually a Postgres DataBase. These nodes not only hold the actual data attributes, but also some metadata that AiiDA uses to track the data provenance. This metadata, however, does not semantically describe the contents of the DataNodes, but rather their connectivity and unique identifier. The initial goal is to simply be able to translate between semantically defined DLite data and AiiDA DataNodes.

### 1.1 DLITE [1]

DLite employs a semantic datamodel that can be supported by an underlying ontology. Data in DLite is represented as an Instance, identified by a UUID and described by its MetaData. This description contains a set of named properties and dimensions. Each property has a given basic type, e.g. float, and potentially the dimensions that describe the amount of this basic type can be found in the property. If no dimensions are present, the property contains a single entry of its type, i.e. it is a scalar. The datamodel used by DLite is an example of a [StructOfArrays](#) [2] design.

Upon creating an Instance that is described by some metadata, the extent of each of the dimensions must be set, which allows to create an empty Instance with the correct sizes of its properties. These can then be filled out by the actual data that will be contained by the Instance.

The fact that the datamodel of an Instance is fully described by the attached metadata forms the basis for easily converting certain data between different datamodels. It allows for code to reason about the layout and types of the data contained in certain Instances.

This conversion process can be further aided with an accompanying Ontology that further describes what each of the properties are semantically. For example, one datamodel of a crystalline structure could store the positions of atoms in a property with name **atomic\_positions**, while another might store it in a property simply named **positions**. Even though these two properties are designated with different names, they *mean* the same thing. If an ontology is then used to *describe* the meaning of each of the properties inside the datastructures, automatic conversion between the two becomes feasible.

The current deliverable establishes an interface between DLite Instances and AiiDA DataNodes, which can be used inside AiiDA WorkChains to perform calculations with. This forms the basis for future work, as it signifies to some extent the *exit point* of the data used in OpenModel, and the execution software represented by AiiDA.

## 1.2 [AiiDA \[3\]](#)

AiiDA uses the concept of Nodes. These are entries in an underlying database which can represent Data (DataNodes) or Processes (ProcessNode). In this report we focus on the DataNodes. Each node is represented by some metadata that is identical for each node, and a blob of data that holds the actual data of said node. There are many different DataNodes, each a subclass of DataNode class, which holds their respective data, and usually define validation and storage methods. The blobs of data that are of interest can be accessed through the **node.attributes** dictionary of the DataNodes.

The main difficulty with respect to developing an interface layer between DLite Instances and AiiDA DataNodes, is that the latter hold arbitrarily structured data. This means that the *attributes* of a DataNode can be an arbitrary nesting of dictionaries and lists, thus being a mix of both the [StructOfArrays \[2\]](#) and [ArraysOfStructs \[2\]](#) data paradigms. The unpacking and restructuring of the DataNodes attributes into the pure StructOfArrays design of DLite, while maintaining the former's structure, is the main difficulty that has to be overcome, and will be the main topic of the remainder of this report.

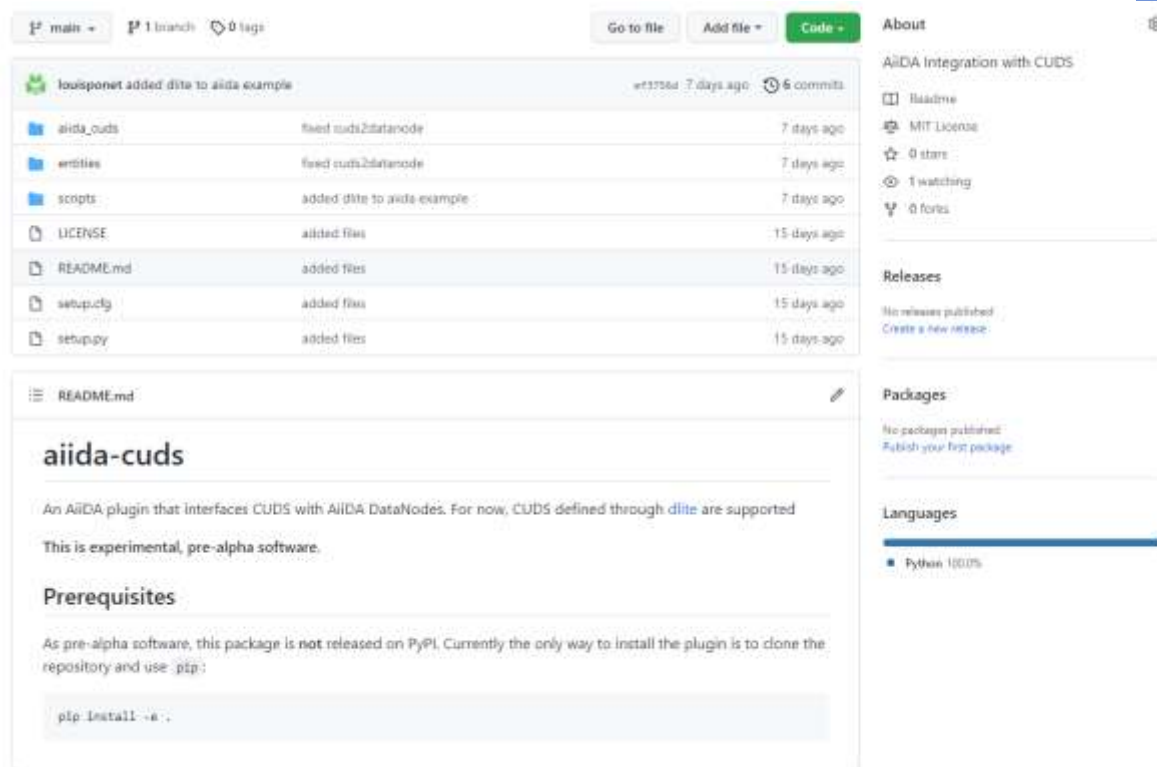
## 2 [AiiDA-CUDS \[4\]](#)

AiiDA provides a way of extending the core library with plugins. These usually define a set of WorkChains, CalcJobs, CalcFunctions and Data. This means that there is no concrete set of possible AiiDA DataNodes, thus requiring a generic way of generating DLite metadata for a given DataNode.

DLite employs a strict model of StructOfArrays entities rather than ArraysOfStructs. An ArrayOfStructs, or any attribute of an AiiDA DataNode that encompasses another class or dict will be referred to as nested data. AiiDA DataNodes in general do contain nested data, making it not a priori clear how to map to DLite datastructures.

The solution to this problem is to unpack the AiiDA DataNodes into a strict StructOfArrays, while annotating the property names to capture the level of nesting. The exact annotation method might subject to change, but the general concept will remain the same.

The developed code can be found in the [github repository \[5\]](#).



The screenshot shows the GitHub repository for 'aiida-cuds'. At the top, there are navigation buttons for 'main', '1 branch', and '0 tags'. Below this is a commit history table:

Commit	Author	Time
louisponet added dilite to aiida example	wt37864	7 days ago
aiida_cuds	fixed cuds2datanode	7 days ago
entities	fixed cuds2datanode	7 days ago
scripts	added dilite to aiida example	7 days ago
LICENSE	added files	15 days ago
README.md	added files	15 days ago
setup.cfg	added files	15 days ago
setup.py	added files	15 days ago

The README file is displayed below, titled 'aiida-cuds'. It describes the package as an AiiDA plugin for CUDS and includes a 'Prerequisites' section with a code block for installation:

```
pip install -e .
```

The right sidebar contains metadata: 'About' (AiiDA Integration with CUDS), 'Releases' (No releases published), and 'Languages' (Python 100%).

Figure 1: A screenshot from of the aiida-cuds GitHub repository. The actual AiiDA-CUDS interface code can be found in the aiida-cuds subdirectory, static datamodels in the entities subdirectory and scripts for running and testing the interfaces in the scripts subdirectory.

## 2.1 DEMONSTRATING EXAMPLES

To explain the design decisions that were taken it is most instructive to look at some examples. The first example demonstrates the process for a trivial case where the AiiDA DataNode does not have any nested data. The second example is more involved and highlights the issues that arose during the development.

### 2.1.1 EXAMPLE 1: KPOINTS DATA

The attributes of a certain KpointsData node are as follows:

```
{'mesh': [4, 4, 4], 'offset': [0.0, 0.0, 0.0]}
```



Using the translation, this generates the following DLite metadata:

```
{
  "uuid": "93c23afc-8552-5cdc-a0e6-d254a71fbaca",
  "meta": "http://onto-ns.com/meta/0.3/EntitySchema",
  "uri": "onto-ns.com/meta/1.0/core.array.kpoints",
  "name": "core.array.kpoints",
  "version": "1.0",
  "namespace": "onto-ns.com/meta",
  "description": "Inferred metadata for 656518e7-63c2-40ea-b3ee-9dcc213dd177 of type KpointsData",
  "dimensions": [
    {
      "name": "mesh_1",
      "description": "Dimension 1 of attribute mesh"
    },
    {
      "name": "offset_1",
      "description": "Dimension 1 of attribute offset"
    }
  ],
  "properties": [
    {
      "name": "mesh",
      "type": "int32",
      "dims": [
        "mesh_1"
      ]
    },
    {
      "name": "offset",
      "type": "float64",
      "dims": [
        "offset_1"
      ]
    }
  ]
}
```

We can see that the correct dimensions and properties were generated. The attributes of the KpointsData are then trivially mapped to the properties of the corresponding DLite datastructure.

### 2.1.2 EXAMPLE 2: ORBITALDATA

Here we consider a more involved AiiDA DataNode that represents the information of the atomic orbitals to be used in a [Wannier90 workchain](#) [6]. The structure of such a node is as shown on the right. We see here that the top-level attribute **orbital\_dicts** is an array of dictionaries, i.e. an array with nested data or an `ArrayOfStructs`. To construct a corresponding DLite datastructure we thus have to convert this to a `StructOfArrays` representation, while encoding the nesting level in order to allow for a reverse conversion from the DLite datastructure back to an AiiDA `OrbitalData` node.

```
{'orbital_dicts': [{'spin': 0,
  'position': [0.0, 0.0, 0.0],
  'kind_name': 'Ni',
  'diffusivity': None,
  'radial_nodes': 0,
  '_orbital_type': 'realhydrogen',
  'x_orientation': None,
  'z_orientation': None,
  'magnetic_number': 0,
  'angular_momentum': 2,
  'spin_orientation': None}]}
```

The generated metadata looks as follows:

```
{
  "uuid": "ce35b9a6-5a9b-5fe2-9acf-22dcdd673c26",
  "meta": "http://onto-ns.com/meta/0.3/EntitySchema",
  "uri": "onto-ns.com/meta/1.0/core.orbital",
  "name": "core.orbital",
  "version": "1.0",
  "namespace": "onto-ns.com/meta",
  "description": "Inferred metadata for c5fa17b5-c852-436d-957b-260f755a6418 of type OrbitalData",
  "dimensions": [
    {
      "name": "orbital_dicts_1",
      "description": "Dimension 1 of attribute orbital_dicts"
    },
    {
      "name": "orbital_dicts_dict_position_2",
      "description": "Dimension 2 of attribute orbital_dicts_dict_position"
    }
  ]
},
```

```

"properties": [
  {
    "name": "orbital_dicts_dict_spin",
    "type": "int32",
    "dims": [
      "orbital_dicts_1"
    ]
  },
  {
    "name": "orbital_dicts_dict_position",
    "type": "float64",
    "dims": [
      "orbital_dicts_1",
      "orbital_dicts_dict_position_2"
    ]
  },
  {
    "name": "orbital_dicts_dict_kind_name",
    "type": "string",
    "dims": [
      "orbital_dicts_1"
    ]
  },
  {
    "name": "orbital_dicts_dict_radial_nodes",
    "type": "int32",
    "dims": [
      "orbital_dicts_1"
    ]
  },
  {
    "name": "orbital_dicts_dict__orbital_type",
    "type": "string",
    "dims": [
      "orbital_dicts_1"
    ]
  },
  {
    "name": "orbital_dicts_dict_magnetic_number",
    "type": "int32",
    "dims": [
      "orbital_dicts_1"
    ]
  },
  {
    "name": "orbital_dicts_dict_angular_momentum",
    "type": "int32",
    "dims": [
      "orbital_dicts_1"
    ]
  }
]
}

```

As can be seen from the above, every attribute from the nested dictionary is translated into a DLite property with name **orbital\_dicts\_dict\_<attribute>**, e.g. **orbital\_dicts\_dict\_angular\_momentum** will be an array with the same length as the original array, with the values corresponding to the **angular\_momentum** attribute of each of the dictionaries that were in the **orbital\_dicts** attribute of the AiiDA OrbitalData node.

Using the **\_dict\_** in a property name thus allows us to encode the level of the nested data, which is then used when the AiiDA DataNode is reconstructed.

[A more complete example](#) [7] to demonstrate how to run an AiiDA [Quantum-Espresso](#) [8] [pw CalcJob](#) [9] starting from DLite data has also been developed.

### 3 BIBLIOGRAPHY

- [1] <https://github.com/sintef/dlite>
- [2] [https://en.wikipedia.org/wiki/AoS\\_and\\_SoA](https://en.wikipedia.org/wiki/AoS_and_SoA)
- [3] <https://aiida.readthedocs.io/projects/aiida-core/en/latest/index.html>
- [4] <https://github.com/H2020-OpenModel/aiida-cuds>
- [5] [https://github.com/H2020-OpenModel/aiida-cuds/aiida\\_cuds/cuds.py](https://github.com/H2020-OpenModel/aiida-cuds/aiida_cuds/cuds.py)
- [6] <https://github.com/aiida-team/aiida-wannier90>
- [7] [https://github.com/H2020-OpenModel/aiida-cuds/blob/main/scripts/dlite\\_aiida\\_example.py](https://github.com/H2020-OpenModel/aiida-cuds/blob/main/scripts/dlite_aiida_example.py)
- [8] <https://www.quantum-espresso.org/>
- [9] [https://aiida-quantumesspresso.readthedocs.io/en/latest/module\\_guide/calculations.html#module-ai-ida\\_quantumesspresso\\_calculations\\_pw](https://aiida-quantumesspresso.readthedocs.io/en/latest/module_guide/calculations.html#module-ai-ida_quantumesspresso_calculations_pw)

## 4 ACKNOWLEDGMENT



*This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 953167.*

*This document and all information contained herein is the sole property of the OpenModel Consortium. It may contain information subject to intellectual property rights. No intellectual property rights are granted by the delivery of this document or the disclosure of its content.*

*Reproduction or circulation of this document to any third party is prohibited without the consent of the author(s).*

*The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed herein lies entirely with the author(s).*

*All rights reserved.*

---