2022

Thomas F. Hagelien (SINTEF), Casper W. Andersen (SINTEF), Jesper Friis (SINTEF), Francesca L. Bleken (SINTEF), Sylvain Gouttebroze (SINTEF), Louis Ponet (EPFL), Anders J. Eklund (SINTEF)

SINTEF

D3.1-WRAPPER SDK

D3.1: WRAPPER-SDK

DOCUMENT CONTROL

Document Type	Other
Status	Final
Version	1.0
Responsible	Francesca L. Bleken, Jesper Friis (SINTEF)
Author(s)	Thomas F. Hagelien (SINTEF), Casper W. Andersen (SINTEF), Jesper Friis (SINTEF), Francesca L. Bleken (SINTEF), Sylvain Gouttebroze (SINTEF), Louis Ponet (EPFL), Anders J. Eklund (SINTEF)
Release Date	2022.06.30, 2023.01.24

ABSTRACT

Design considerations and current development status of the ExecFlow Software Development Kit (ExecFlowSDK).

CHANGE HISTORY

Version	Date	Comment
0.1	2022-05-30	First Draft. Thomas Hagelien, Jesper Friis, Casper Andersen
0.2	2022-06-05	Updates. Thomas Hagelien, Jesper Friis
0.3	2022-06-28	Updates. Thomas Hagelien
0.4	2022-06-29	Updates. Thomas Hagelien, Jesper Friis, Casper Andersen, Francesca Bleken, Sylvain Gouttebroze
0.5	2022-06-30	Revision. Anders Eklund, Jesper Friis, Sylvain Gouttebroze, Louis Ponet
0.6	2022-06-30	Revision. Finalized by Welchy Cavalcanti and submitted.
1.0	2023-01-24	Added request from EC

DISSEMINATION LEVEL

PU	Public	X
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

LIST OF FIGURES

LIST OF TABLES

1 MOTIVATION

OpenModel will develop a body of interfaces to third-party models as described in the roadmap for implementation of wrappers and solvers and needed supporting tools for pre- and post-processors, MCO and analytics tools described (deliverable 3.2).

The list of wrappers is long, so to realistically be able to implement and maintain these wrappers it is essential with a good development environment and tooling for simplifying and streamlining the development and maintenance processes. It is the first version of this tooling set that is provided in this deliverable.

2 INTRODUCTION

Table 1. Different pathways and use cases for the SDK to fulfill.

Software status	Steps toward ExecFlow plugin	SDK should provide
Existing AiiDA plugin	Generate DLite/SOFT data models and mappings for the plugin data node	Guidelines and examples
DLite/SOFT compatible software	Generate AiiDA plugin (data nodes, parser, calcfunction)	Code generator, best practices, adapted cookie cutter template
OSP-core wrapper	Generate DLite/SOFT data models, AiiDA plugin (data nodes, parser, calcfunction)	Guidelines and examples, code generator, best practices, adapted cookie cutter template
Independent software	Generate DLite/SOFT data models and mappings, AiiDA plugin (data nodes, parser, calcfunction)	Guidelines and examples, code generator, best practices, adapted cookie cutter template

An important requirement is that we want to be able to reuse already existing wrappers implemented as (i) traditional AiiDA plugins, (ii) as DLite/SOFT plugins or as (iii) OSP-CORE wrappers. Hence, the tooling developed in this deliverable will focus on integrating these existing types of wrappers into the OpenModel framework. Hence, the name Wrapper Software Development Kit (SDK) is slightly misleading. For this reason, we will hereafter refer to the tools provided in this deliverable as ExecFlowSDK.

The main objective of the ExecFlowSDK is to accelerate the development process through best practises and providing reusable and customizable templates and specialized tools. The ExecFlowSDK facilitates a fully semantic and seamless integration of third-party tools with ExecFlow, such as external data sources, simulation tools and other knowledge sources. Since ExecFlow is based on AiiDA and exploits the provenance capabilities that AiiDA, the ExecFlowSDK must facilitate storage of data provenance as part of the plugin development (e.g., configuration parameters of a transformation pipeline).

The current version of the SDK will include the following main components:

- Integration of OTEAPI pipelines as AiiDA processes.

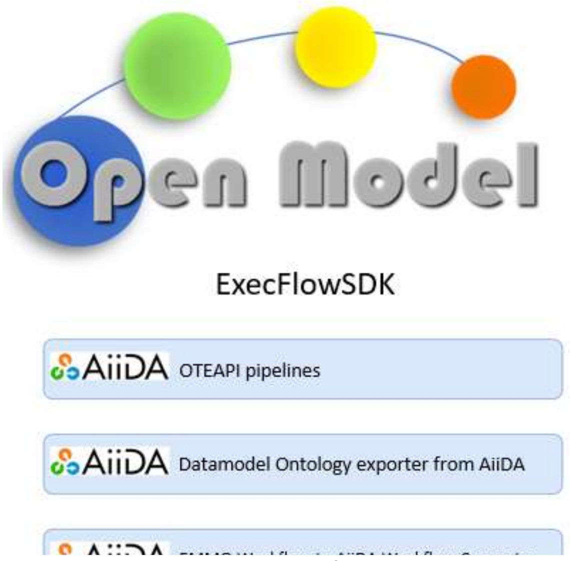


Figure 1. Main components of the ExecFlowSDK.

- Code generation for exporting AiiDA Data Nodes to DLite/SOFT data models to support semantic mappings.
- Workflow generator utility which, based on EMMO Workflows, can generate AiiDA workflows on-the-fly.

The ExecFlowSDK is an expert tool for developers and enrich AiiDA workflows, processes and data with semantics based on EMMO. The SDK is specialized and geared towards working with EMMC embraced technologies for semantic representation, interoperability, and interfaces. The SDK extends the current AiiDA development process enabling OpenModel to fully utilize the build-in provenance system.

The work carried out in this deliverable anticipates some of the design of ExecFlow that is the topic of task 4.2 lead by EPFL. So, even though mainly SINTEF have been behind the implementation of this deliverable, the design has been discussed broadly among several partners in the biweekly WP4 meetings organized by EPFL. This mitigates the risks that the implementation in this deliverable become misaligned with contributions from other partners in task 4.2. The ExecFlowSDK also builds on and further extend the AiiDA-CUDS integration delivered in D4.4.

3 WORK CARRIED OUT / STATUS

3.1 DESIGN PROCESS

3.1.1 CONSTRAINTS & CONSIDERATIONS

Automatic provenance tracking is one of the key features of AiiDA, which OpenModel will benefit from taking full advantage of. AiiDA requires all internal data representations and workflow descriptions to be developed as part of its plugin system. Once all data nodes, calculation functions, work chains and parsers are defined, AiiDA will be able to generate the full provenance graphs during the execution of the workflows.

The benefits of using AiiDA is achieved when AiiDA is used idiomatically, i.e., the way AiiDA is designed to operate. Therefore, integrating AiiDA intimately into the OpenModel framework is the most optimal solution. This diverts slightly from the DoA, but the gains far outweigh the disadvantages in this approach. The challenge of OpenModel is thus how to take advantage of the provenance system, existing integrations, and features of AiiDA while still incorporating semantic interoperability.

A clear alignment and integration with sister EU projects play a large role here as well. OntoTrans shows the way forward for how to implement an Open Translation Environment (OTE). Integrating this technology in OpenModel will dramatically improve the cross-platform interoperability as well as the intrinsic semantic interoperability offered by the OTE software developed in the OntoTrans project called OTEAPI.

It is furthermore expected that other EU projects, e.g., DOME 4.0 and VIPCOAT, will implement a similar support for OTEAPI, making the OpenModel platform both competitive among its peers as well as interoperable between themselves.

3.1.1.1 LESSONS LEARNED (AIIDA AS OTE DATA SOURCE)

The OTEAPI pipelines are typically defined as a set of processing steps starting with a data source and ending up with a transformation/simulation step. One initial idea was to take advantage of the fact that AiiDA stores all its data in a PostgreSQL database. Each data point stored in the database is assigned a unique identifier, allowing external systems to refer to the data. By using the persistent data nodes from AiiDA as data sources it would be possible to define specific (DLite/SOFT) data-models mapped to ontological concepts as semantic data documentation. Using the AiiDA PostgreSQL database directly as a data source turned out to be difficult, as the database is dependent on a local AiiDA repository filesystem. Attempts to distribute information to recreate the local configuration violates assumptions made by AiiDA and the development team concluded that using the AiiDA PostgreSQL databases as a shared resource was not optimal. However, AiiDA provides its own module for exposing data through a REST API. This allows specific installations of running a local web-service that can expose contents to external users. Another option for sharing data between AiiDA installations are through AiiDA Archives. Data and provenance graphs may be shared as artifacts on public research data repositories such as Materials Cloud Archive[1], Zenodo[2] and Open Science Framework[3].

3.1.1.2 FOLLOW FUTURE DEVELOPMENT OF AIIDA

As ExecFlow is based around AiiDA, there is a strong dependency to AiiDA as a software component. AiiDA is rapidly evolving, and the risk of having to deal with breaking changes is not unlikely. On the other hand, AiiDA will develop new features that ExecFlow might benefit from. There are two main strategies on how to deal with external dependencies like this. One is to freeze the dependency to a given major version (for instance v2.x). This will ensure that ExecFlow will not break due to future changes. However, there is a risk that ExecFlow may become obsolete due to dependencies to “legacy” software and the task of upgrading may be significant. The other strategy is to adapt ExecFlow (and the SDK) to the latest version of AiiDA and make continuous adjustments when needed. This might decrease the efficiency of the development process, or force users of OpenModel to upgrade the system more often but ensures that ExecFlow stays relevant.

AiiDA is developing a REST API that eventually might be utilized in a way that removes the need for having local AiiDA repositories and installations. This might be a good future option for OpenModel/ExecFlow as the workflow orchestrations can then be maintained as a single service and part of the OpenModel service stack.

3.1.1.3 USE OF EXISTING RESULTS FROM OTHER WORK PACKAGES

Figure 2 shows some important components that this deliverable uses and builds further on, in addition to the OTEAPI and DLite/SOFT frameworks. Including the AiiDA CUDS in D4.4, the declarative workflow language developed in WP4 and internally at EPFL and the DLite/OSP-CORE integration developed in WP2 and in OntoTrans.

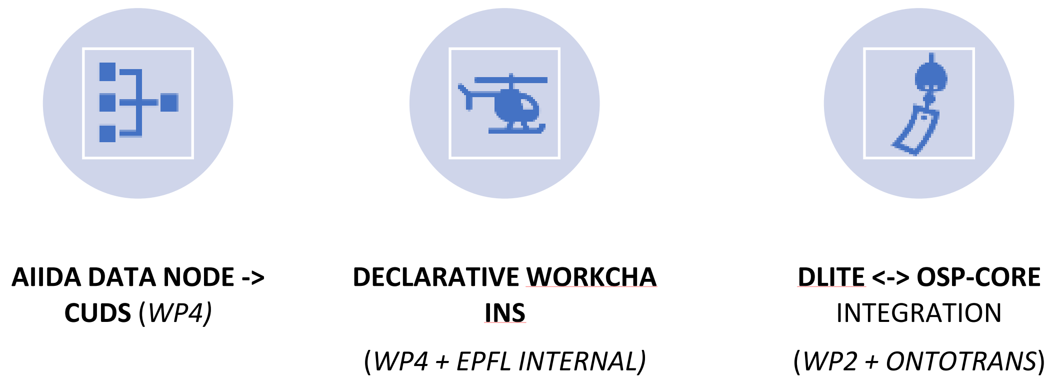

AIIDA DATA NODE -> CUDS (*WP4)*
DECLARATIVE WORKCHAINS
(*WP4 + EPFL INTERNAL)*
DLITE <-> OSP-CORE INTEGRATION
(*WP2 + ONTOTRANS*)

Figure 2. Important components of this deliverable.

3.1.2 PROVENANCE + SEMANTIC INTEROPERABILITY (AIIDA + OTEAPI)

The major benefit of using AiiDA is its integrated provenance system. A core feature of AiiDA is to keep track of all calculations and workflows, as well as their inputs and outputs, with a meaningful linking between all of these. OpenModel will take full advantage of this provenance and granularize its parts appropriately to ensure a meaningful provenance can be kept of every ExecFlow run, with the proper amount of detail. This will have a

[1] Materials Cloud Archive
[2] Zenodo
[3] Open Science Framework

fundamental impact on the quality of data produced by the OpenModel platform in terms of accountability, sovereignty, licensing, and general provenance information available.

A major detraction from AiiDA is its data model rigidity and intrinsic inability to handle semantic data. This will be mitigated by utilizing OTEAPI developed as part of the EU project OntoTrans. OTEAPI provides a framework for handling data through data models, with support for mapping these data models to ontologies. OTEAPI allows for any specific integration to specify the data models and data model system one wants to use, e.g., DLite/SOFT, SimPhoNy, or OSP-Core, while providing the necessary semantic interoperability between data models and ontologies that AiiDA is lacking.

Together, these two software frameworks will become the basis for ExecFlow, allowing it to provide semantic interoperability using one's desired data model implementation software, while keeping the stability of AiiDA in terms of data provenance and workflow execution.

3.1.3 OTEAPI

The Ontology-based Translation Environment Application Programming Interfaces (OTEAPI) is a framework for connecting heterogeneous data resources, semantic interoperability frameworks, open simulation platforms, and standalone simulation tools. The OTEAPI allows for building complex use-case representations by combining a set of simple reusable functions.

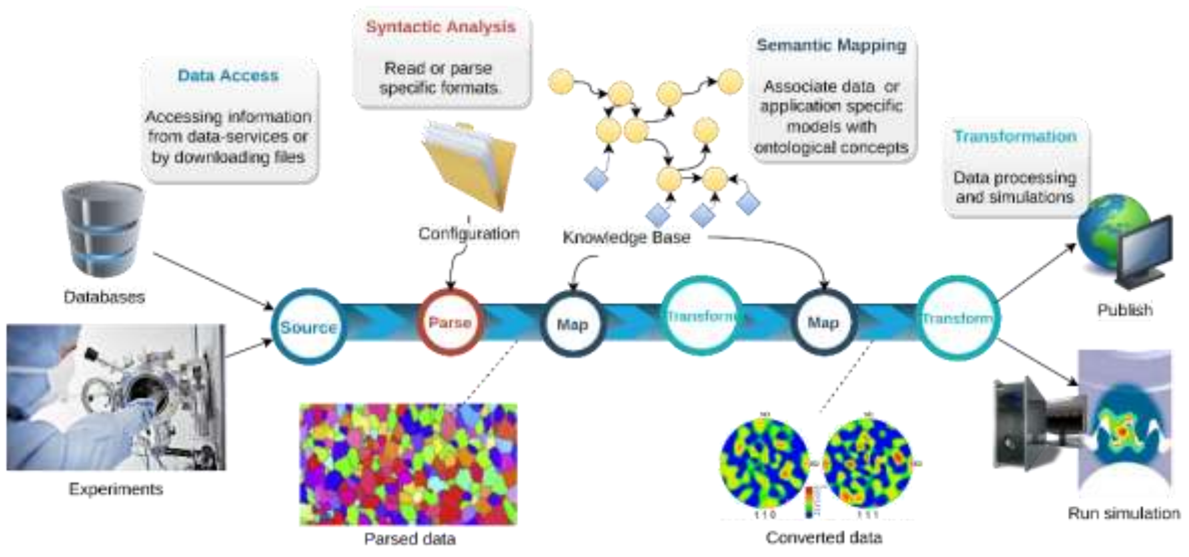

Syntactic Analysis
Read or parse specific formats
Data Access
Accessing information from data-services or by downloading files
Semantic Mapping
Associate data or application specific models with ontological concepts
Transformation
Data processing and simulations
Databases
Configuration
Knowledge Base
Source
Parse
Map
Transform
Map
Transform
Publish
Experiments
Parsed data
Converted data
Run simulation

Figure 3. Example of an OTEAPI pipeline.

Figure 3 presents an example of an OTEAPI pipeline. The end user, in this case a modeler running a simulation software, needs texture data in each data representation (e.g., as pole figures) and format. He/she finds some experimental data for the relevant alloy and condition. With OTEAPI the modeller can select the input data source without the need of taking into consideration how the data is represented (e.g., as an orientation distribution function (ODF)) and formatted. If both the data provider and modeller map their data models to a common ontology the OTEAPI can seamlessly transform the ODF data representation to the expected pole figure data representation.

OTEAPI is an implementation of the pipe and filter architectural pattern.[4] This simply describes a set of connected components that process a stream of data from an input source to a receiver such as a data sink. Once a pipeline is constructed, it is possible to execute data processing as if it were a single component. The pipeline

[4] Bass, Len, Paul Clements, and Rick Kazman. Software Architecture in Practice: Software Architect Practice_c3. Addison-Wesley, 2012.

can further be embedded in a larger workflow system. It provides a unified way to manage complex software processing. By combining reusable elements for accessing or downloading data, parsing information, transforming information, and performing business logic, the OTEAPI is highly adaptable.

A complex processing scenario is defined by constructing a "pipeline" from reusable and interchangeable parts. OTEAPI supports the 6 main categories of tasks (aka. filters) listed in Table 2.

Table 2. The main categories of tasks (filters) implemented in OTEAPI.

Category	Description
Resource Strategy *Data access (information retrieval)*	Accessing data will typically involve authorization, transportation protocols and query languages. A simple data access module will be able to initiate a transport protocol (for instance http) and access an artifact given a unique URL.
Parser Strategy *Syntactic Analysis*	Heterogenous data sources have different syntactic ways of defining the structure and contents. Some data source will provide schemas and metadata (SQL, HDF5), some are documented file formats (CSV, XLSX, etc.). To manage the rich variety of formats, a parser or syntactic analyser is needed for further processing of the data.
Mapping Strategy *Semantic Mapping*	A data source associated with a semantic data model allows for semantic interoperability. The semantic data model is constructed by mapping a (physical) data model representing the business data to (logical) ontological concepts. This can be performed by defining relations between the physical properties define in the data model, to domain ontology concepts that can describe the meaning of the physical quantity, the measurement technique, etc. This allows for reasoning about data and ensuring unambiguous interpretations.
Filter Strategy *Data Filtering*	Data filtering allows for extracting a subset of the available data, by defining a view or a size limitation. The data filter is usually very closely related with the data accessor and may include data source specific query languages.
Function Strategy *Synchronous Information Processing*	OTEAPI defines two different methods of generating new information from a stream of data: synchronous and asynchronous processing. With synchronous information processing an operation or *function* will be performed, and when completed the next step in the pipeline is called.
Transformation Strategy *Asynchronous Processing*	Asynchronous processing differs from the synchronous processing in that it will start a process as background task and return a handle to the process. The handle can be used to inspect the status of the background task. Asynchronous processing is useful for instance when starting long-running computations.

A pipe is simply a source that consumes data by an input filter. A "filter" can be any transformation of the data or operations that receives the stream of data from an input pipe and delivers data as a stream to an output pipe. There can be many distinct types of filters, but in general they will share the same generic interfaces and can in principle be connected to any pipe.

A key advantage of the pipe & filter pattern is that it allows for loose and flexible coupling of interchangeable filters, the filters are re-usable, a set of pipelines can be run parallel. Another principal factor is that filters can be "anything" if it honors the interface specification and is treated as "black-boxes" by the system.

A common design pattern for managing a set of (run-time) interchangeable modules with different implementations is the strategy pattern. A module will run the same named function, but the instance of the object that is called will be changed depending on the context.

In the scope of the Ontology based Translation Environment, there are four main categories of operations:

- Resource operations administrates data sources and target sources. This includes downloading data using various protocols, parsing specific file formats, interacting with web services or database management systems, generating target data and uploading information.
- Mapping between source specific metadata and general/-or domain specific semantic representations.
- Defining filtering operations that specify a subset of information .
- Transformations that operate on data and produce new information.

In the execution of a pipeline, the session object allows for sharing information between the different filters (implemented as strategies).

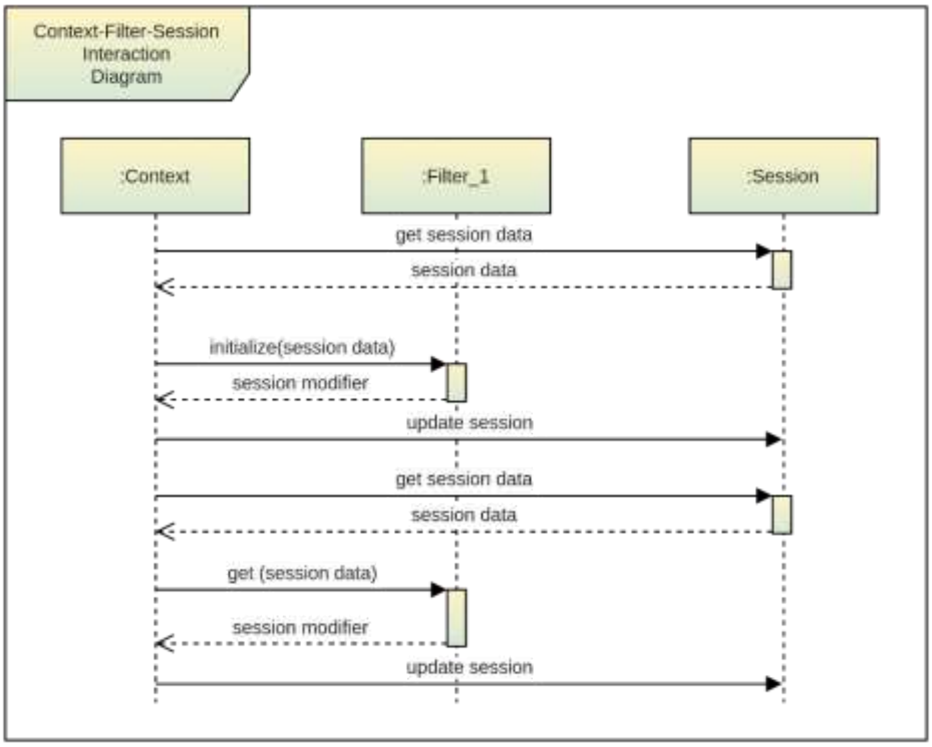


Figure 4. Context-filter-session diagram showing the communication between a context class, a filter, and the session.

Figure 4 shows the context-filter-session interaction diagram, illustrating the communication between a context class, a filter, and the session. The session is simply an in-memory key-value storage that is used as temporal memory management in the communication between different filter types. The session data will first be fetched into the context class and sent as argument to the filter's `initialize()` method. Note, the initialize function is stateless, and its only purpose is to create an object that will be added or updated in the current session.

3.1.4 ARCHITECTURE DESIGN FOR THE SDK

The SDK will be made up of several code modules to facilitate the creation of ExecFlow plugins. The main modules include code generation for AiiDA-specific classes (Data, Calculation, Workflow, Parser). However, another major part of the SDK will not be code modules, but rather documentation and tutorial guidelines for how to use the code modules, including examples based on use cases and possibly Success Stories.

3.2 DEVELOPMENT OF PARTS OF THE SDK

3.2.1 USE CASES

The use cases are the core basis for developing the SDK.

Table 3. A simple use case to motivate the SDK development.

Case	Thermodynamic modelling of solidification (part of SS3)
Status	**Description:** Function that returns the phases amount and composition after the solidification of a liquid metal under normal casting conditions. **Input:** Alloy chemical composition, thermodynamic database, model parameters (default values available). **Output:** Phases amount and chemical composition. **Implementation:** Python script calling a commercial package. **Data models:** Available for alloy chemical composition and outputs. **Ontology:** Microstructure domain ontology under development in EMMC Task Force, beta version available. **AiiDA:** No AiiDA plugin available for the software.
Needs	Tools to facilitate building the AiiDA plugin. Access to AiiDA Data Nodes to store and share data in external databases. Import microstructure description from external sources (using the same data model). Tool (or standardized format) to facilitate semantic description of the function and inputs/outputs. Facilitate interoperability with software computing solid state precipitation and possibly corrosion model.
SDK components	AiiDA Plugin Cookiecutter. AiiDA Data Node class generation from available data models. Semantic mapping generation functionality. AiiDA Calculation function or class code generation.

3.2.2 CODE GENERATORS

The ExecFlowSDK code generators are based on the Jinja[5] template engine for Python. The code generators can generate executable source code from a set of configurations and/or data models. In some use cases, there is a need to generate AiiDA Data Nodes from existing DLite/SOFT or OSP-Core representations. The code generators will be able to generate the necessary AiiDA classes for integrating these data models in an AiiDA plugin. In other cases, there is a need go from existing AiiDA Data Nodes to an external representation, e.g., for external data processing or data documentation, in which case the opposite process is needed.

Constructing workflows from a declarative language, i.e., statements indicating *what* is to be done, contrary to imperative statements describing *how* a workflow is to be constructed, is a powerful and effective paradigm. Jinja is employed for extracting declarative statements written in JSON or YAML formatted files and producing AiiDA workflows, specifically WorkChain Python classes, which will be a part of the AiiDA plug-in.

[5] Jinja is a fast, expressive, extensible templating engine. https://jinja.palletsprojects.com/en/3.1.x/

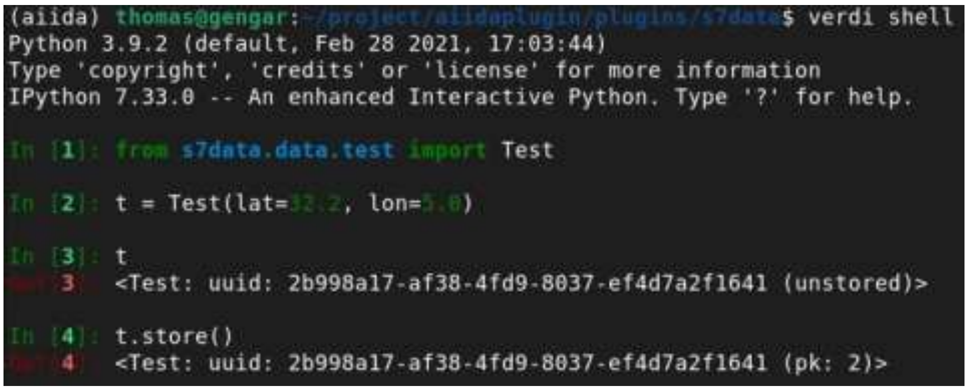
The product of the ExecFlowSDK code generators is code included in an AiiDA plugin. AiiDA plugins are generated using the AiiDA Plugins Cookiecutter[6] hosted on GitHub. The generated plugin supports basic regression tests based on pytest, includes a “Read The Docs" compliant documentation template engine, GitHub actions to be executed as part of the CI/CD steps to run tests and check test coverage at every commit, syntax and coding style checking and setup for making the plugin pip installable (prepared for submission to PyPI). For the OpenModel developers, the new generated or hand-written AiiDA Node classes need to be added. In addition, the Entry Points need to be updated for AiiDA to be able to find the nodes.

In the example below in Figure 5, the code generator was used to generate an AiiDA Data Node from a DLite/SOFT entity. The plugin was then prepared and installed. In the screenshot of the `verdi shell` in Figure 6 the generated Data Nodes are instantiated and stored in the AiiDA storage backend.

```
import yaml
from typing import TextIO
from jinja2 import Template, Environment
from s7.pydantic_models.soft7 import SOFT7Entity, SOFT7EntityPropertyType



with open('<path to entity declaratation (in yaml)>') as file:
    entity = yaml.safe_load(file)
    softEntity = SOFT7Entity(**entity)


# Generate the AiiDA DataNode to be included in the AiiDA plug-in

env = Environment(
    autoescape=False,
    optimized=False )

template_file = "aiida_data_gen.txt.j2"
output_file = "aiida_exampledatanode.py"

# Read jinja2 template file

with open(template_file, "r") as f:
    template = env.from_string(f.read())


# Generate new AiiDA DataNode class definition.

with open(output_file, "w") as f:
    template.stream(
        class_name = 'ExampleDataNode',
        entity=softEntity).dump(f)
```

Figure 5. Example: Generating AiiDA Data Node from DLite/SOFT type Entities.

1 [6] **AiiDA plugin cutter:** https://github.com/aiidateam/aiida-plugin-cutter



```
(aiida) thomas@gengar:~/project/aiidaplugin/plugins/s7data$ verdi shell
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.33.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from s7data.data.test import Test

In [2]: t = Test(lat=32.2, lon=5.0)

In [3]: t
Out[3]: <Test: uuid: 2b998a17-af38-4fd9-8037-ef4d7a2f1641 (unstored)>

In [4]: t.store()
Out[4]: <Test: uuid: 2b998a17-af38-4fd9-8037-ef4d7a2f1641 (pk: 2)>
```

Figure 6. Example: Using a generated AiiDA plugin to instantiate a Data Node generated from a DLite/SOFT Entities.

3.2.3 DATA MODEL TRANSFORMATIONS

The generated data model can be utilized by DLite/SOFT or OSP-CORE, for the latter an (EMMO-based) ontology and a mapping are required to generate the CUDS. DLite/SOFT data models also need mappings to EMMO ontologies for full semantic interoperability.

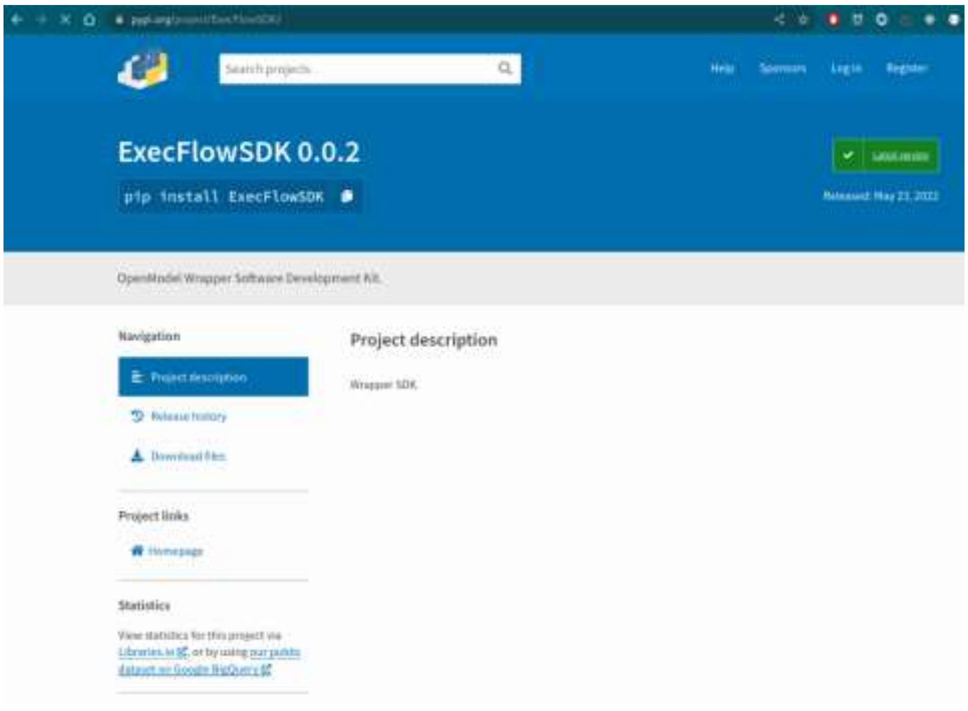
3.2.4 DECLARATIVE WORKCHAINS (AIIDA)

A specification of the AiiDA workflows (called WorkChains) can be described using simple statements in a text file (does not require programming). The wrapper SDK includes the AiiDA workflow generator for rapid onboarding of external simulation tools. The workflow generator is based on the ontological workflow representation (described by EMMO Workflows middle level ontology in WP1), from which the declarative WorkChain description eventually will be generated from. The generator will be able to generate AiiDA Workflows on-the-fly and will support composing workflows from already existing AiiDA workflows, completely new elements, or a hybrid. The declarative WorkChain description is crucial for the whole OpenModel it is outside the scope of deliverable D3.1 as WorkChains are part of WP4.

3.2.5 CI/CD FOR PYTHON PACKAGE CREATION AND DEPLOYMENT

As described in D2.1 – OpenModel Agile Manifesto, CI/CD systems are of high importance already at early stages of development to ensure good and continuous progress.

A first (empty) release of the package has already been published on PyPI (see Figure 7) and automatic deployment for further releases has been set up to ensure streamlined flow of updates to the users. The code is setup such that releases will be automatically pushed to PYPI, which is a python package service and allows developers to install the SDK using the python package installer called `pip`.



Search projects

Help Sponsors Log in Register

ExecFlowSDK 0.0.2

```
pip install ExecFlowSDK
```

Latest version

Released: May 23, 2022

OpenModel Wrapper Software Development Kit.

Navigation

Project description

Release history

Download files

Project links

Homepage

Statistics

View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery

Project description

Wrapper SDK

Figure 7. View of ExecFlowSDK Python package page on PyPI.

3.2.6 OTEAPI ADOPTION FOR AIIDA

The AiiDA OTEAPI pipeline plugin implement a semantic ETL (Extract, Transform, Load) system for integrating or ingesting data from heterogeneous data source into AiiDA. OTEAPI pipelines consist of connectors to heterogeneous data sources and ontology-mapped data models which allows external representations to connect to native AiiDA Data Nodes through semantic interoperability. The plugin allows running OTEAPI pipelines natively within AiiDA. This will ensure a close and direct interoperability with knowledge sources within external platforms such as OntoTrans, MarketPlace, DOME 4.0, VIMMP and VIPCOAT.

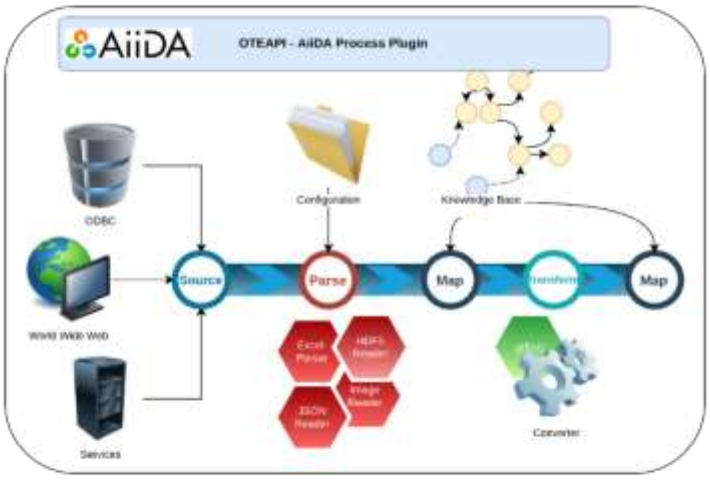


Figure 8. Use of OTEAPI in AiiDA.

The OTEAPI Integration for AiiDA involves

- Strategy Configurations as Data Nodes
- Strategies as CalcFunctions

- Pipelines as WorkChains

And is further detailed in the following subsections.

3.2.6.1 STRATEGY CONFIGURATIONS

The data resource configuration specifies an external information resource. The resource can either be a service (query endpoint, REST API, ODBC, etc.), or a static data file that can be retrieved. If the `downloadUrl` and a `mediaType` is specified, then OTEAPI can reason about the proper download protocol and deserialization of the information. If an `accessService` is specified, then OTEAPI will need to provide a specialized plugin for the particular service.

Table 4. Generic specifications of a resource.

Property	Type	Description
`downloadUrl`	string	Definition: The URL of the downloadable file in a given format. E.g., CSV file or RDF file. Usage: downloadURL *SHOULD* be used for the URL at which this distribution is available directly, typically through a HTTPS GET request or SFTP.
`mediaType`	string	The media type of the distribution as defined by IANA [IANA-MEDIA-TYPES]. Usage: This property *SHOULD* be used when the media type of the distribution is defined in IANA [IANA-MEDIA-TYPES].
`accessUrl`	string	A URL of the resource that gives access to a distribution of the dataset. E.g. landing page, feed, SPARQL endpoint. Usage: `accessURL` *SHOULD* be used for the URL of a service or location that can provide access to this distribution, typically through a Web form, query or API call. `downloadURL` is preferred for direct links to downloadable resources.
`accessService`	string	A data service that gives access to the distribution of the dataset.
`license`	string	A legal document under which the distribution is made available.
`accessRights`	string	A rights statement that concerns how the distribution is accessed.
`description`	string	A free-text account of the distribution.
`publisher`	string	The entity responsible for making the resource/item available.
`configuration`	object	Resource-specific configuration options given as key/value-pairs.

The configuration can include a reference to the DataCache. The DataCache is an internal storage mechanism that allows for temporarily storing artifacts for an active period to avoid the need to access the same resource multiple times over the network.

Mapping operations

Mapping allows for extending the data documentation with relations to knowledge concepts defined in ontologies or vocabularies. When using DLite/SOFT, the mappings are defined between data model properties to

EMMO based domain ontologies or other knowledge base concepts. Mappings are usually expressed using triples, or knowledge statements, defining the explicit relation between one concept and another. The mappings ontology <https://github.com/emmo-repo/domain-mappings> is the preferred choice.

Table 5. Generic specifications for a mapping filter.

Property	Type	Description
`mappingType`	string	Type of registered mapping strategy. E.g., mapping/demo.
`prefixes`	object	List of shortnames that expands to an IRI given as local value/IRI-expansion-pairs.
`triples`	array	List of semantic triples given as (subject, predicate, object).
`configuration`	object	Mapping-specific configuration options given as key/value-pairs.

Filtering operations

Often the data source contains more data than needed, and there is a need to extract a specific piece of information. Filters define a particular view into a subset of the data. Filters define either constraints or express a specific query.

Table 6. Generic specifications for a filter strategy.

Property	Type	Description
`filterType`	string	Type of registered filter strategy. E.g., filter/sql.
`query`	string	Define a query operation.
`condition`	string	Logical statement indicating when a filter should be applied.
`limit`	integer	Number of items remaining after a filter expression.
`configuration`	object	Filter-specific configuration options given as key/value-pairs.

Transformations

Transformations are asynchronous processes used to run simulations or any background job. Since transformations can take an arbitrary amount of time, they are not expected to return results further down a pipeline.

Table 7. Generic specifications for a transformation strategy.

Property	Type	Description
`transformation_type`	string	Type of registered transformation strategy. E.g., celery/remote.
`name`	string	Human-readable name of the transformation strategy.
`description`	string	A free-text account of the transformation.

`due`	string	Optional field to indicate a due data/time for when a transformation should finish.
`priority`	enum	Define the process priority of the transformation execution.
`secret`	string	Authorization secret given when running a transformation.
`configuration`	object	Transformation-specific configuration options given as key/value-pairs.

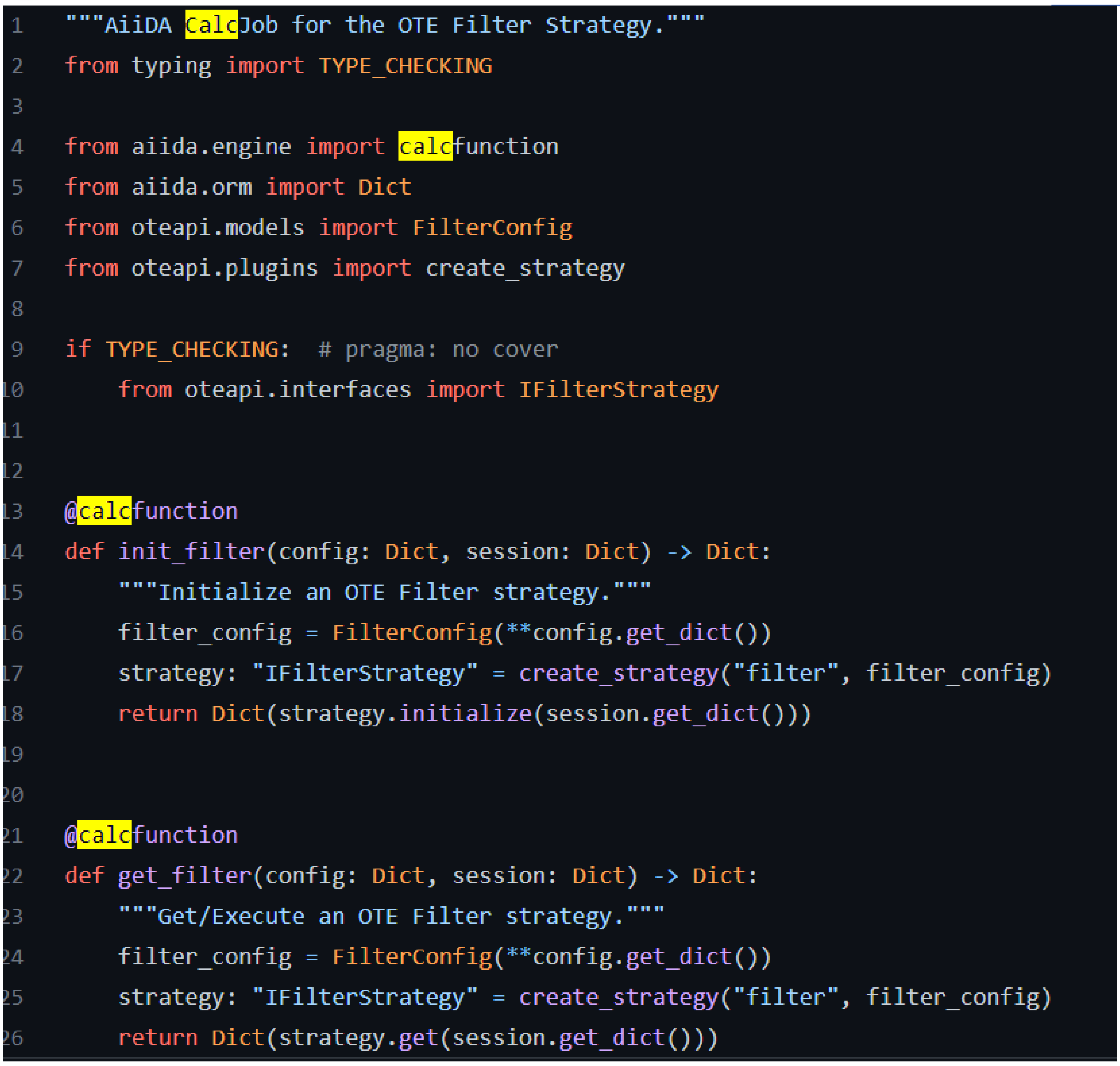
Transformation status

Since the transformations does not return a response when the task is finished, a client needs to use the API to (regularly) check the status of a running job. The transformation status model describes the expected payload sent from the OTEAPI and back to the client.

Table 8. General specifications for the status of a transformation strategy.

Property	Type	Description
`id`	string	ID for the given transformation process.
`status`	string	Status for the transformation process.
`messages`	array	Messages related to the transformation process.
`created`	string	Time of creation for the transformation process. Given in UTC.
`startTime`	string	Time when the transformation process started. Given in UTC.
`finishTime`	string	Time when the transformation process finished. Given in UTC.

3.2.6.2 STRATEGIES AS CALCFUNCTIONS



```python
"""AiiDA CalcJob for the OTE Filter Strategy."""
from typing import TYPE_CHECKING

from aiida.engine import calcfunction
from aiida.orm import Dict
from oteapi.models import FilterConfig
from oteapi.plugins import create_strategy

if TYPE_CHECKING:  # pragma: no cover
    from oteapi.interfaces import IFilterStrategy


@calcfunction
def init_filter(config: Dict, session: Dict) -> Dict:
    """Initialize an OTE Filter strategy."""
    filter_config = FilterConfig(**config.get_dict())
    strategy: "IFilterStrategy" = create_strategy("filter", filter_config)
    return Dict(strategy.initialize(session.get_dict()))


@calcfunction
def get_filter(config: Dict, session: Dict) -> Dict:
    """Get/Execute an OTE Filter strategy."""
    filter_config = FilterConfig(**config.get_dict())
    strategy: "IFilterStrategy" = create_strategy("filter", filter_config)
    return Dict(strategy.get(session.get_dict()))
```

Figure 9. How OTE strategies are implemented as AiiDA CalcFunctions. Here, Filter strategies are shown.

3.2.6.3 DECLARATIVE PIPELINES (OTEAPI)

Data pipelines are constructed connecting data sources to a set of processing steps, where the output from one step is the input to the next. In OTEAPI, the pipelines are constructed connecting a sequence of strategies which can be individually configured. A declarative way of expressing the OTEAPI pipeline is shown in the following figure, where YAML is used for defining the declarative pipeline.

```
version: "1"
strategy:
    - dataresource: mydata_resource
      downloadUrl: "http://example.com/resource/image.jpg"
      mediaType: image/jpeg
    - mapping: mymapping
      mappingType: mapping/demo
    - transformation: mytrans
      transformation_type: script/dummy
pipeline:
    mypipe: [mydata_resource | mymapping | mytrans]
```

Figure 10. Example: Declarative OTE Pipeline.

Each element in the list of strategies with configurations is used to instantiate each strategy programmatically. The pipeline orchestration can either be generated specific AiiDA WorkChains. In addition, a more generic approach where the declarative pipeline is encoded as an AiiDA Data Node and interpreted by a generic WorkChain class.

4 NEXT STEPS

4.1 FULL USE CASE DEMONSTRATION

In order to provide tests of the OpenModel tools before the success stories have their modelling workflows in place, both DCS and GCL have kindly provided already working use cases that during the next months will be used for a first end-to-end testing of the demonstration of the ExecFlowSDK.

4.2 WORKFLOW ONTOLOGY

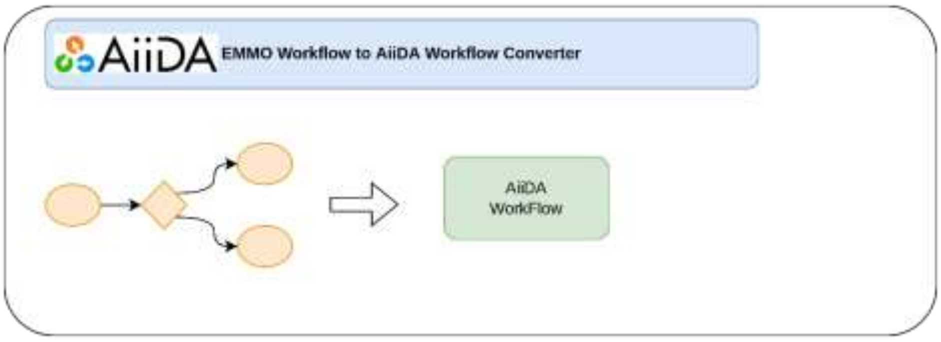


Figure 11. Conversion from an EMMO-based ontology workflow to an AiiDA Workflow.

The wrapper SDK will include the AiiDA workflow generator for rapid onboarding of external simulation tools. The workflow generator is based on the ontological workflow representation described by EMMO Workflows Midlevel ontology, as described in WP1. The generator will be able to generate AiiDA Workflows on-the-fly and will support composing workflows from already existing AiiDA workflows, completely new elements, or a hybrid.

4.3 DOCUMENTATION

Guidelines for writing an OpenModel ExecFlow Plugin will be added.

5 CONCLUSION

The ExecFlow SDK is accelerating the development of AiiDA plugins for the execution of semantic workflows with full provenance support.

The scope of D3.1 has evolved to take full advantage of AiiDA and integrate more closely with the Translation Services using the same standard interfaces and semantic interoperability platforms.

The software ecosystem is a collaborative effort between related EU projects. The OTEAPI is incorporated in OpenModel as a joint effort between the projects OntoTrans, VIPCOAT and OpenModel.

6 APPENDIX

6.1 NOMENCLATURE

- **Adapter pattern** is a software design pattern that allows to reuse existing (external) resources, the *adaptee*, that does not conform to the OntoTrans REST API by defining a separate *adapter* that provides it with a conforming REST API.
- **AiiDA** an open-source Python infrastructure to help researchers with automating, managing, persisting, sharing, and reproducing the complex workflows.
- **API – application programming interface –** is a software interface that allows applications to talk to each other.
- **APP** is the software module of the OTE (one for each Innovation Challenge) used to declare the components of a specific User Case, to build the workflow by selecting available solutions (e.g., models, database, other cases data, machine learning approaches) and to connect the available tools together (passive or active).
- **Authentication** a process of verifying the identity of a person (or thing).
- **Authorisation** the controlling of access rights to resources (access policy).
- **BDSS - business decision support system** is a tool for aiding decision-making.
- **Container** is an isolation of software package of applications and dependencies, allowing for reproducibility and simplify deployment.
- **Conversion** is a special filter that converts data from one representation to another.
- **CRUD** read, write, update, and delete - the basic operations for persistent storages.
- **CSV – Comma-separated values** is a simple file format used to store tabular data, such as a spreadsheet or database. The actual column separator and decimal point may depend on application and location.
- **CUDS - Common Universal Data Structures** is the ontology compliant data format of OSP-core. A CUDS is at the same time an ontological individual, an API, a container, RDF, and a node in a graph.
- **Dataresource** [in OntoTrans interface API] refers to the location of a piece of data. It can either be a data source or a data sink.
- **DataSpace** is a metadata storage for the Semantic Data Models, which are data models describing the low-level representation of a dataset.
- **Data sink** is something that designed to receive data.
- **Data source** is anything which produces digital information, from the perspective of systems which consume this information.
- **DBMS – database management system** is a software package designed to define, manipulate, retrieve, and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

- **DevOps** is a methodology that merges the disciplines of software development (dev) and operations (ops).
- **DLite** is a MIT-licensed data-centric interoperability framework for working with scientific data. Data is described with simple, yet flexible data models. DLite is written in C, with bindings to Python and Fortran. It is available at https://github.com/SINTEF.
- **Docker** is a technology that enables containerization of applications and dependencies and provide a portable standardized execution environment.
- **Filter** [in pipes and filter pattern] represent and algorithm to be performed on a specific type of data. It typically receives input from a pipe and delivers processed information to another pipe but may also read from a data source or write a data sink.
- **GUI – graphical user interface –** is a type of user interface that allows users to interact with and application through graphical elements.
- **Interface** the signature (codes and messages) of a set of software methods that allows for communication between different software elements.
- **JSON** is a popular specific file format for expressing information in terms of data objects expressed as attribute-value statements and collections of data objects.
- **Knowledge base** a technology used to store complex unstructured information. The OpenModel knowledge base is implemented in OntoFlowKB.
- **Kubernetes** is a platform for managing and automating container orchestration. See Container.
- **Mapping** (in the OTEAPI) is a special type of filter that maps data, typically read from a data source to concepts in an ontology.
- **MCO** - multicriteria optimisation is mathematical optimisation involving more than one objective function to be optimised simultaneously.
- **MCDM** - multicriteria decision making is decision-making based on MCO.
- **MODA – materials MOdelling DAta** – provides a systematic description and documentation of simulations including the user case, model, solver, and post-processor. It seeks to organize the information so that even complex simulation workflows can be conveyed more easily and key data about the models, solvers and post-processors and their implementation can be captured. EMMC provides MODA templates for physics-based modelling workflows.
- **OAuth2.0** is an industry standard protocol for authorization.
- **OntoFlow** a core component of OpenModel that provide workflow specifications based on technical and business requirements.
- **OntoFlowKB** is the OpenModel knowledge base implemented as a triple store.
- **MoDS – CMCL's Model Development Suite** is a set of tools for generation of surrogate models, multi-objective optimisation, uncertainty propagation, classification/clustering.
- **OpenAPI** is a standard for self-documenting REST APIs that allow both humans and computers to discover and understand the capabilities of the service.
- **OntoKB** is the OntoTrans knowledge base implemented as a triple store.
- **OntoRec** is an ontology-based recommender system in OntoTrans.
- **OpenID Connect** extends the OAuth 2.0 authorization framework with an authentication layer.
- **OpenModelDS** is a data service that will store the (potentially big) data needed by the MCO/ML algorithms. It is used by OpenModelDM.
- **OpenModelDM** is a collection of decision-making tools build on top of the MCO & BDSS Services.
- **OSP – Open Simulation Platform**. Allow for building an ecosystem of models and resources.
- **OSP-Core** is an OSP implementation originally developed in the SimPhoNy project and used in several other projects, like MarketPlace.
- **OTE – Open Translation Environment –** is a platform that provides search, decision mechanisms, tools to link models and databases, tools to link translators with modellers and industry and tools to share best practices.
- **Payload** is the part of transmitted data that is the actual intended message. Headers and metadata are sent only to enable payload delivery. In OntoTrans
- **Pipe** [in pipes and filters pattern] is a software component represent data with a type and connects the output from one *filter* into the input of another *filter*.

- **Pipeline** is a series of processes, usually linear, which filter or transform data. Pipelines are normally quick, and the data flow does not normally loop. The first process takes raw data as input, does something to it, then sends its results to the second process, and so on, eventually ending with the final result being produced by the last process in the pipeline. Branching is possible. The processes may be running concurrently (using streams).
- **Pipes and filters pattern** is a software design pattern that decompose a task that performs complex processing into a series of separate elements that can be reused.
- **Proxy** a proxy server acts as an intermediary between a client and a resource.
- **PostgreSQL** is an Open Source, SQL compliant, relational database management system
- **REST API** is used to describe a web service that allows different computer systems to communicate over HTTP.
- **SFTP – SSH File Transfer Protocol** is a secure file transfer protocol. It runs over the SSH protocol. It supports the full security and authentication functionality of SSH.
- **SimPhoNy** - An EU project as well as the native implementation of the core CUDS object and class generation for OSP-core.
- **SOFT - SINTEF Open Framework and Tools** is a set of concepts and tools for interoperability. Several implementations exist, of which SOFT7 is the development version where new ideas are tested out while DLite is more a working horse. The code base It is available at https://github.com/SINTEF/soft7.
- **SQL – Structured Query Language –** is a standard language for communicate with relational databases.
- **SQLite** is a library that implements a light-weight self-contained SQL database engine. The database is a file on the local file system.
- **SSH – Secure Shell** is a cryptographic network protocol for operating network services securely over an unsecured network.
- **Strategy pattern** is a software design pattern that enables selecting an algorithm at runtime.
- **Transformation** [in OntoTrans interface API] are processes that create new knowledge or data. This includes both numerical models/machine learning and things like post processing.
- **Verification** is the process that determines the quality of the software. In short it is about meeting the specifications.
- **Validation** is the process that determines whether the software meets the expectations, which is a much more subjective than verification.
- **V&V - verification and validation**. The combination of verification and validation.
- **Workflow** is a set of processes, generally non-linear, which filter or transform data, possible triggering external events. The processes are not assumed to be running concurrently. The data flow diagram can branch or loop and there may not be any clearly defined "first" or "final" process. Workflows can be complex and long-lived and may involve human interactions. They can be expressed using MODA (or MODA-like) graphs representing the combination of methods for the prediction of properties of our objects/processes of interests (the totality or part of the user case).
- **WorkflowDesigner** is a tool that will use OpenModelDM to optimise ontological workflow descriptions by taking technical and business requirements into account.
- **WorkflowBuilder** is a component of OntoFlow that can compile the ontological workflow descriptions from the WorkflowDesigner to a format understood by ExecFlow.
- **Wrapper** is defined as an entity that encapsulates and hides the underlying complexity of another entity by means of well-defined interfaces.

7 ACKNOWLEDGMENT

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