2023

VALERIO LUNARDELLI (AMAT), OTELLO M ROSCIONI (GCL), ALESSANDRO CALVIO (UNI-BO), BIJAN YADOLLAHI (CMCL), LOUIS PONET (EPFL), FRANCESCA L. BLEKEN (SINTEF), JESPER FRIIS (SINTEF)

WP5-D5.5: FIRST DEMONSTRATION OF OPENMODEL OIP

D5.5 - FIRST DEMONSTRATION OF OPENMODEL OIP

DOCUMENT CONTROL

Document Type	Deliverable Report
Status	Final
Version	1.0
Responsible	Valerio Lunardelli (AMAT)
Author(s)	Valerio Lunardelli (AMAT), Otello M Roscioni (GCL), Alessandro Calvio (UNIBO), Bijan Yadollahi (CMCL), Louis Ponet (EPFL), Francesca L. Bleken (SINTEF), Jesper Friis (SINTEF).
Release Date	2023-09-22

ABSTRACT

This report aims to give an overall description of the OpenModel OIP infrastructure and its main components, and to provide a proof-of-concept demonstration based on a simple use case. The current deliverable (D5.5) is the first one developed for T5.5, which focuses on the application and testing of the OIP platform. The objective of T5.5 is to test OIP in a real environment based on six use cases. Using the KPI metrics defined in D5.5, the OpenModel OIP will be evaluated in terms of data analytics, workflow execution, scoring and visualisation to demonstrate the capabilities and its value proposition. This report has been delayed due to its dependency on software components, which have also been delayed.

CHANGE HISTORY

Version	Date	Comment
0.1	2022 to August 2023	Preparation and discussion, strong cooperation and add from partners, alignment of content; including agreement of WP partners, work and alignment with other WPs (WP1, WP2 and WP4) and approval from technical manager. updated with the documentation of the software components and execution of a demo use case.
0.2	2023-08-23	Version updated including revision from WP leader and partners.
0.3	2023-09-18	Final version, after M30 meeting discussions.
1.0	2023-09-22	Finalized for submission.

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

D5.5- FIRST DEMONSTRATION OF OPENMODEL OIP

1 INTRODUCTION

OVERVIEW

OpenModel provides an Open Innovation Platform (OIP) for Integrated Materials Modelling, which offers 5 main ingredients:

1) EMMO based ontology extensions as basis for all developments,
2) An Interoperability layer providing an implementation of EMMO-based,
3) A Simulation Platform based on standardised interfaces and semantic common application programming interfaces (API), to enable integration of third-party physics-based modelling codes,
4) Smart workflow builders that respond to semantic information and requirements and creates on the fly advanced workflows considering Key Business and Technical Performance Indicators (KPI) utilising the semantic power embedded in the platform, and
5) workflow executors and curators able to perform and manage the results making it readily and transparently available for further control and processing by other platforms.

Six success stories have been considered to demonstrate such features in OpenModel, covering a wide range of applications and reflecting the generic scope of the project, enabling it to address all materials modelling, processing, and characterization fields. These success stories include:

- Success Story [1] – Synaptic Electronics: From Materials Properties to Next-Generation Memory Devices (CNR, AMAT)
- Success Story [2] - Composite Manufacturing Simulation (SISW)
- Success Story [3] – Civil Engineering – Reinforced Concrete (HYDRO, SINTEF, HEREON)
- Success Story [4] – Metal Forming: Resource Efficient Processing and Manufacturing (HEREON)
- Success Story [5] – Digital Powder Testing (CMCL)
- Success Story [6] – Fuel Cell Technology (DCS, TOYOTA, HEREON)

In addition, an additional use case was developed in WP5 to allow the product manager to ensure that the OpenModel OIP meets the user's needs, providing feedback for the product owner to implement the required capabilities in WP2. Overall, the WP5 focuses on demonstration of success stories, in which a step-by-step process has been planned to ensure the successful execution of these success stories on the platform. The first step in this process, planned for in Task 5.1, was performed and reported in the submitted D5.1 where data and model information from these success stories has been gathered. Starting from D5.1 data inputs, we refine the process by identifying data collection and curation methods from each success story, as a second step in D5.2. Primary and secondary data have been identified, as well as their collection methods. Then, each Success Story defined its standard technique for data curation and for preserving and sharing the information generated by the success story. With that objective in mind, there will be many interdependencies between tasks within this WP or across WPs. While the dependency on other tasks is evident in the WP5, there are also tasks in other WPs that depend on this task. An example of this is Task 1.1, which deals with the platform's development. This task is complementary to Task 5.1 and Task 5.2 and focuses on the specific technical aspects of data and information exchange as the basis for ontology development.

The current deliverable (D5.5) is the first one developed for T5.5, which focuses on the application and testing of the OIP platform. The objective of T5.5 is to test the OIP in a real environment. Using the KPI metrics defined in D5.5, the OpenModel OIP will be evaluated in terms of data analytics, workflow execution, scoring and visualisation to demonstrate the capabilities and benefit achieved.

The D5.5 is focused on a proof-of-concept workflow demonstrating the overall OIP infrastructure, its main components and how they holistically work together. The proof-of-concept validation will be demonstrated with a simple use case before moving to a real environment validation phase with the OpenModel success stories.

OPENMODEL INNOVATION PLATFORM DESCRIPTION

The OpenModel main objective is to design, create, provide and maintain a sustainable integrated OIP that delivers predictable, validated, and traceable simulation workflows integrating third-party physics-based models, solvers, post-processors and databases seamlessly. The OpenModel project is part of a large EMMC (European Materials Modelling Council) initiative to contribute to and develop a European Ecosystem in materials modelling.

In the following section, we describe

- the targeted OIP user profile
- OIP system overview and architecture
- Design goals and quality attributes/tactics

Finally, we demonstrate the proof-of-concept capabilities of selected modules and provide some restrictions on the OpenModel software design. These suggestions will be outlined in the current document.

OIP USER PROFILE

The primary end user of the OpenModel OIP will typically be industrial translators and modellers. The OIP will help them to build and execute arbitrarily complex materials modelling and simulation workflows, considering both technical and business KPIs. It aims to be an attractive platform for industrial research, and to include verification and validation services.

According to the EMMC definition[1], we consider "Translation" as the process of transforming industrial problems/issues into questions to be solved by modelling and simulation tools, e.g., by supporting industrial innovation. Translators guide manufacturers to find the optimal solution to industrial challenges using modelling workflows and advise industrial end-users in modelling / simulation execution and interpretation of the results, as depicted in the Figure below:

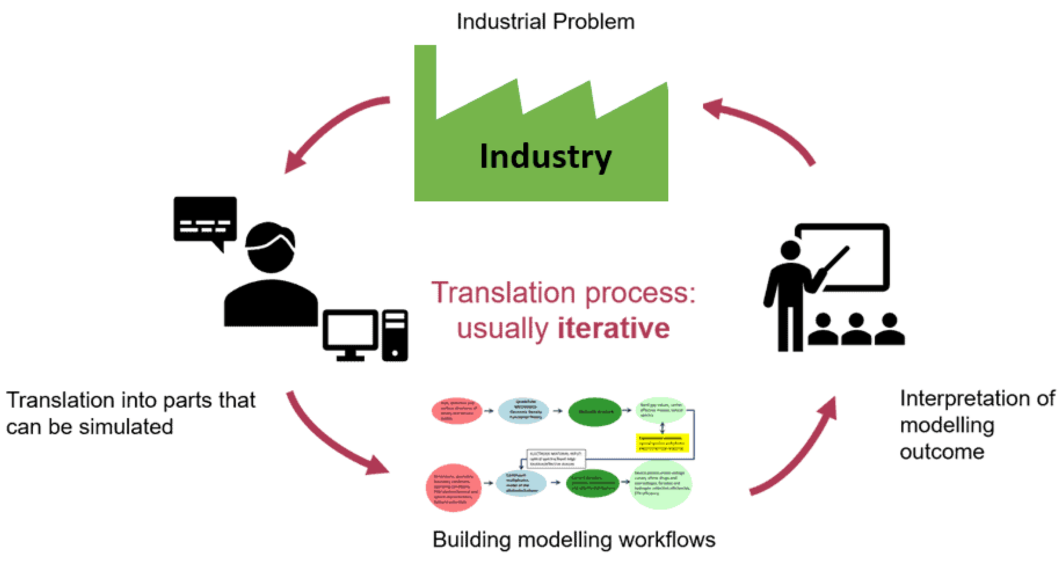

Industrial Problem
Industry
Translation process: usually iterative
Translation into parts that can be simulated
Interpretation of modelling outcome
Building modelling workflows

Figure 1: Translator process and role

The Translator comes in as a multi-professional specialist (or team of professionals) who fulfils a role to understand both the modelling and industrial worlds and speaks both languages.

Following the EMMC definition, the OIP is tailored for an individual or team of "Translator" users with these skill sets:

- Industrial background
- Deep and broad knowledge of modelling and software tools, including their limitations. Extensive network of modellers and an EU-based network of collaborators.

[1] *Hristova-Bogaerds, Denka, Asinari, Pietro, Konchakova, Nnatalia, Bergamasco, Luca, Marcos Ramos, Alicia, Goldbeck, Gerhard, Hoeche, Daniel, Sswang, Ole, Schmitz, Georg j., EMMC translators guide, Zenodo, 2019.* https://doi.org/10.5281/zenodo.3552259

- Broad understanding of different experimental techniques and data analysis tools (suitability, quality)
- Knowledge of economic impact: Balance between investments and expected return; use/define measurable benefits from the modelling.
- Soft and analytical skills: communication, explanation, listening, reporting, organisation, flexibility, multi-tasking, quick learning.
- Being neutral: find the best expert and the most suitable modelling tools, with objective argumentation on the selected models/software/executors.
- Expected to show a proven “track record” of expertise in translation, including success stories, and modelling, if being also the modelling executor
- Managing data confidentiality.

Using the OIP, the “Translator” user will boost European Industry uptake of materials modelling and will contribute to:

- support the business decision and reduce the time to market,
- deliver better/faster in-depth expert solutions, bridge the gap between simulation scales,
- allow to access and use multidisciplinary state-of-the-art modelling and simulation tools,
- limit the number of experiments and time to deliver,
- improve sensitivity analysis.

SYSTEM OVERVIEW

The OIP may be envisioned as a web service where the user, via the browser, can connect to the OIP server and operate the APPs. External providers are used for authentication. The back end provides the interoperability platform, wrappers (to external party data sources, models and simulation software), data storage (triplestore and dataspace) and system monitoring and logging (see Figure 2).

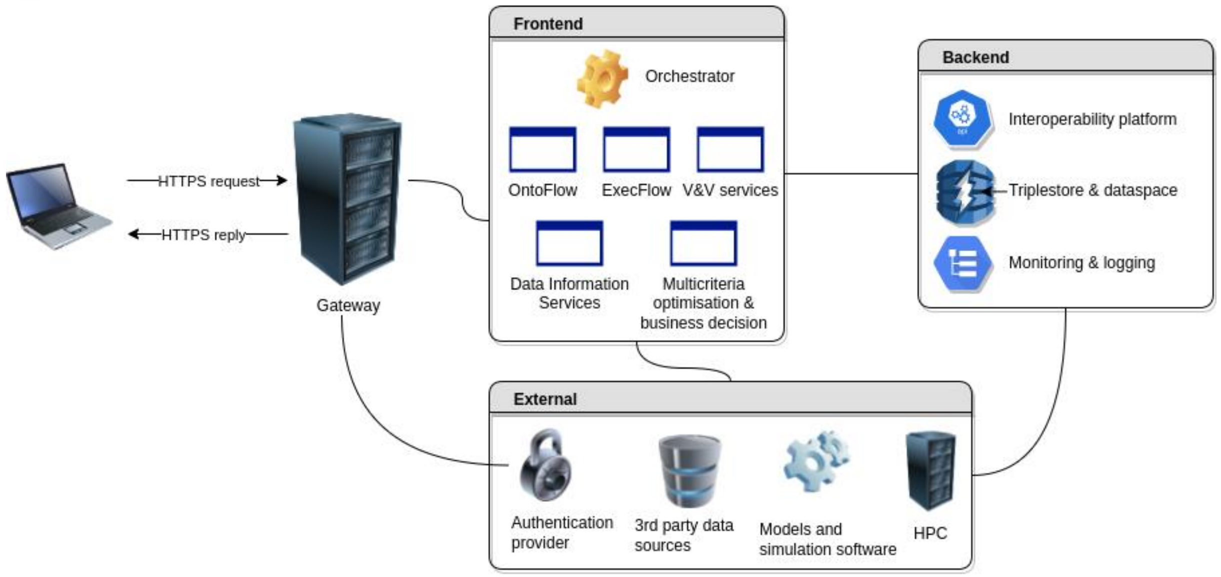


Figure 2: OIP envisioned as a web service. The OIP consist of frontend elements (user interfaces), backend components and external systems and information sources.

OIP has been conceived with a modular approach and the Figure 3 summarizes the holistic interaction between the several components:

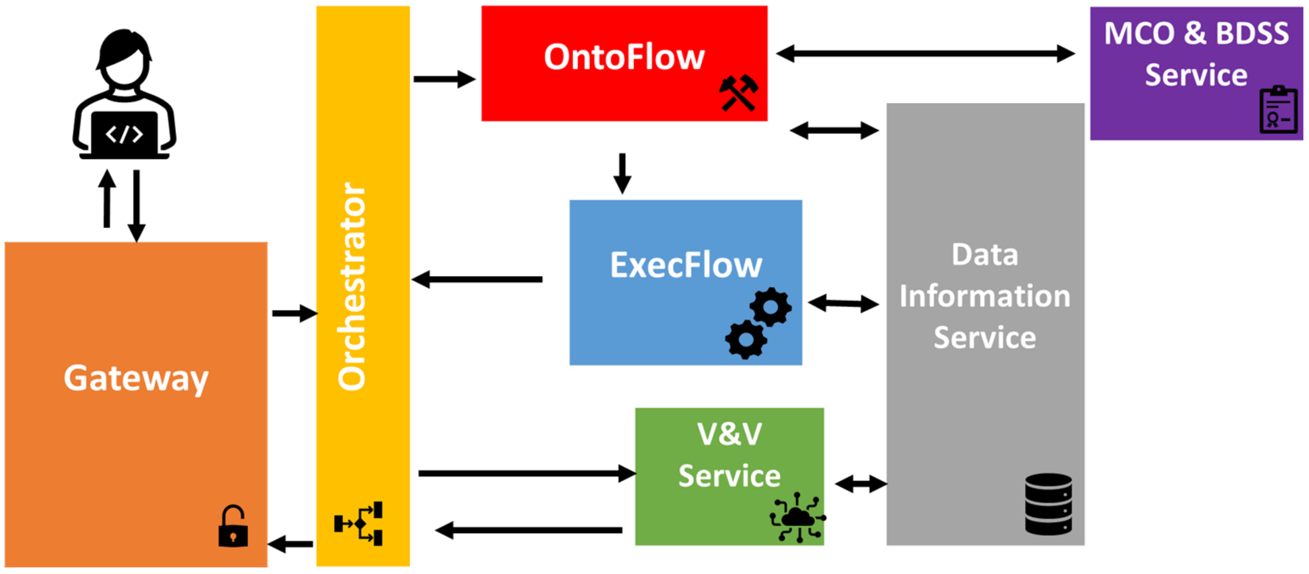


Figure 3: OIP framework and main components

The OIP concept is depicted in the Figure 3. The translator will access the OIP through the gateway. The gateway is responsible for managing the authentication on the OIP and the validation of software licenses, ensuring the security of the entire infrastructure. Below, an orchestrator redirects the requests to the relevant OIP component and keeps the communications isolated from the gateway. Other services at the higher level of the architecture are the OntoFlow, the ExecFlow, the Database

Information Service (DIS), the Verification and Validation (V&V) Services, the Multicriteria Optimisation (MCO), and Business Decision Support System (BDSS) Services.
In a holistic view, OntoFlow can be considered the heart of the platform. It is responsible for specifying and building the workflow based on input specifications, including KPIs from the user. ExecFlow is the workflow executor based on the AiiDA. AiiDA is an open-source Python infrastructure created to help researchers automate, manage, persist, share and reproduce the complex workflows associated with modern computational science and all associated data.
Verification and validation (V&V) services validate the executed workflows and perform the so-called gap analysis.

The complete set of business, technical and simulation KPIs are then passed to the OpenModel semantic workflow builder, OntoFlow, that utilises a shared knowledge base documenting available data sources, models and data sinks as well as an enhanced set of attributes of materials modelling workflow elements. OntoFlow suggests a set of possible workflows that can be evaluated and se-lected by the translator before executing them with the workflow runner, ExecFlow. ExecFlow fetches the needed input to the workflow either directly from the databases integrated into the OpenModel platform in a standardised manner or from external sources like OTE or a Materials Modelling Markeplace (MMMP). The results may be available in an internal database or uploaded to an external marketplace, OTE or BDSS, as chosen by the user. Hence, OpenModel selects, opti-mises, builds and executes complex workflows chosen by the industry, considering technical and business requirements directly. Including the KPIs and MCO enables a straightforward integration of the entire workflow building into existing BDSS, as it facilitates the choice of models and tools. In essence, OpenModel provides a novel decision or recommendation system to support users in choosing the proper workflows and components to execute.

DEMONSTRATION OVERVIEW

The proof-of-concept of OIP selected for D5.5 is based on the molecular dynamics simulation of water. The current demo is divided in three workflows with an increasing level of complexity. In D5.5 we will give an overview of the demonstration phases and results. The full demonstration is available on the H2020-OpenModel repository https://github.com/H2020-OpenModel/Public

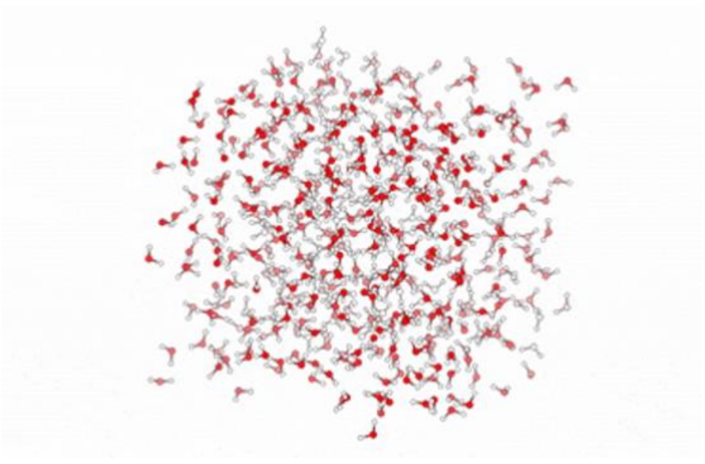


Figure 4: Molecular dynamics simulation of water.

Specifically, the demo is organised as follows:

1. A molecular dynamics simulation of 512 water molecules to demonstrate the use of ontologies for the description, execution, and data management of a simple workflow.
2. A workflow sharing the basic structure of demo [1.], but extended to cover 10 solvents, each described with 3 different material models. This calculation is used also to demonstrate the MCO and V&V services.
3. Full multiscale workflow of the wettability of a polymer membrane.

The first demonstration is based on a simple workflow, whose mereological representation is shown in Figure 5. The workflow uses four datasets: the first one (dataset 1) is the input for task A

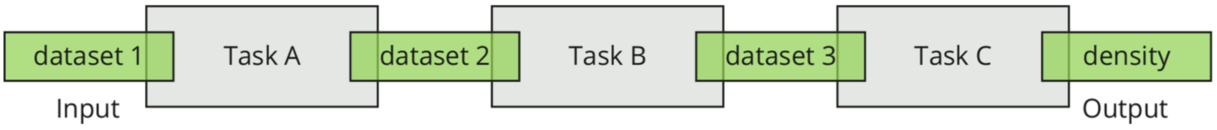


Figure 5: Mereological representation of the Demo number 1.

and includes the metadata controlling the physics-based simulation. The second dataset (dataset 2) is created by task A and used as input for task B. The dataset 3 is the output of task B and contains thermodynamic observables at a given pressure and temperature. Task C is a post-processing step taking the dataset 3 as input. It parses dataset 3 and averages the density at equilibrium and outputs a final data set (density). The task A is the pre-processing of the initial structure and force field, done with the program MOLTEMPLATE, while task B is a physics-based molecular dynamics simulation (RoMM 2.2.2.1) executed with the program LAMMPS. The BPNM representation of the workflow, shown in Figure 6 highlights the data flow between tasks A and B, and the causal sequence of execution of the two programs.

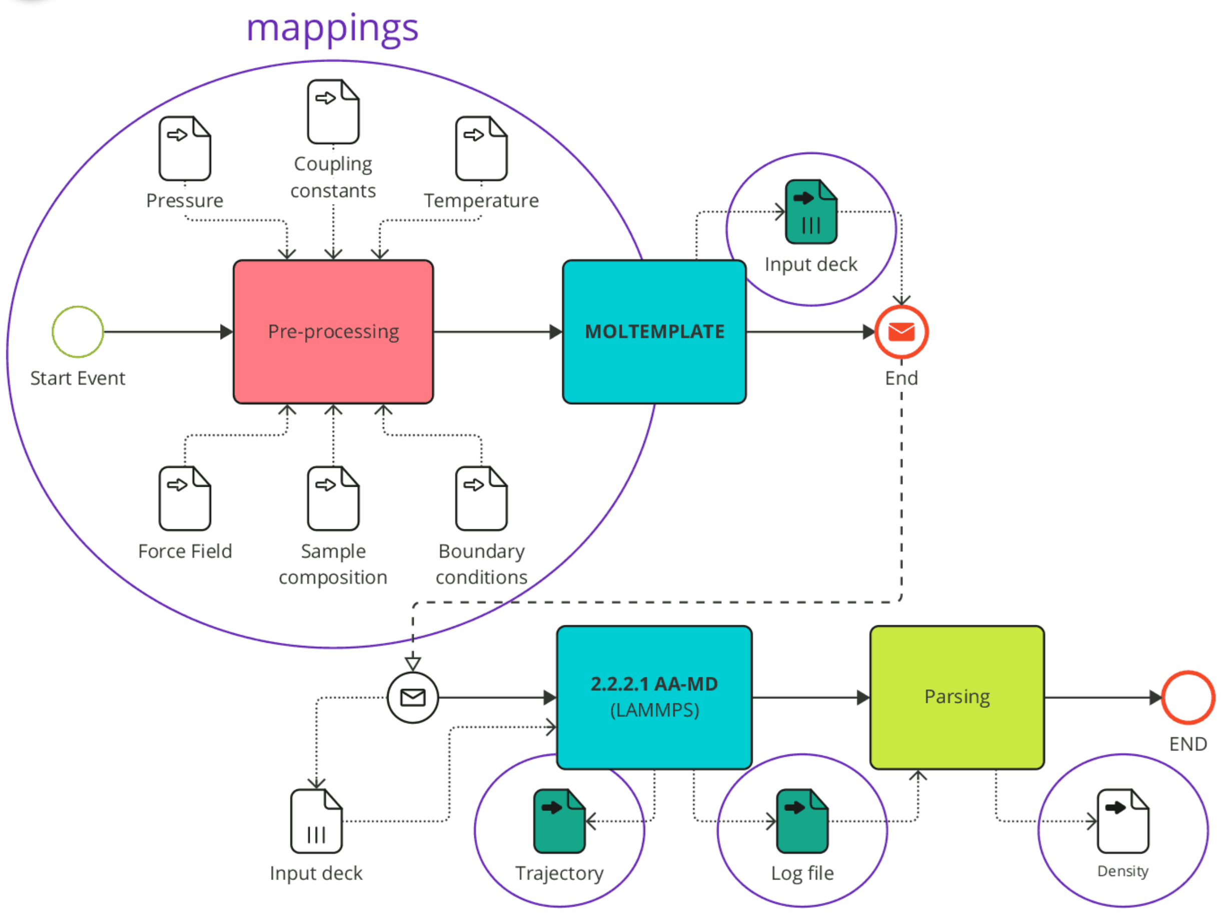


Figure 6: Workflow of the demo number 1 represented as a BPMN diagram.

An initial stub of the Knowledge Base (KB) has been instantiated using EMMO classes and perspectives to create a comprehensive taxonomy of models, materials, and processes. The schematic representation of the OpenModel workflow taxonomy is shown in Figure 7.

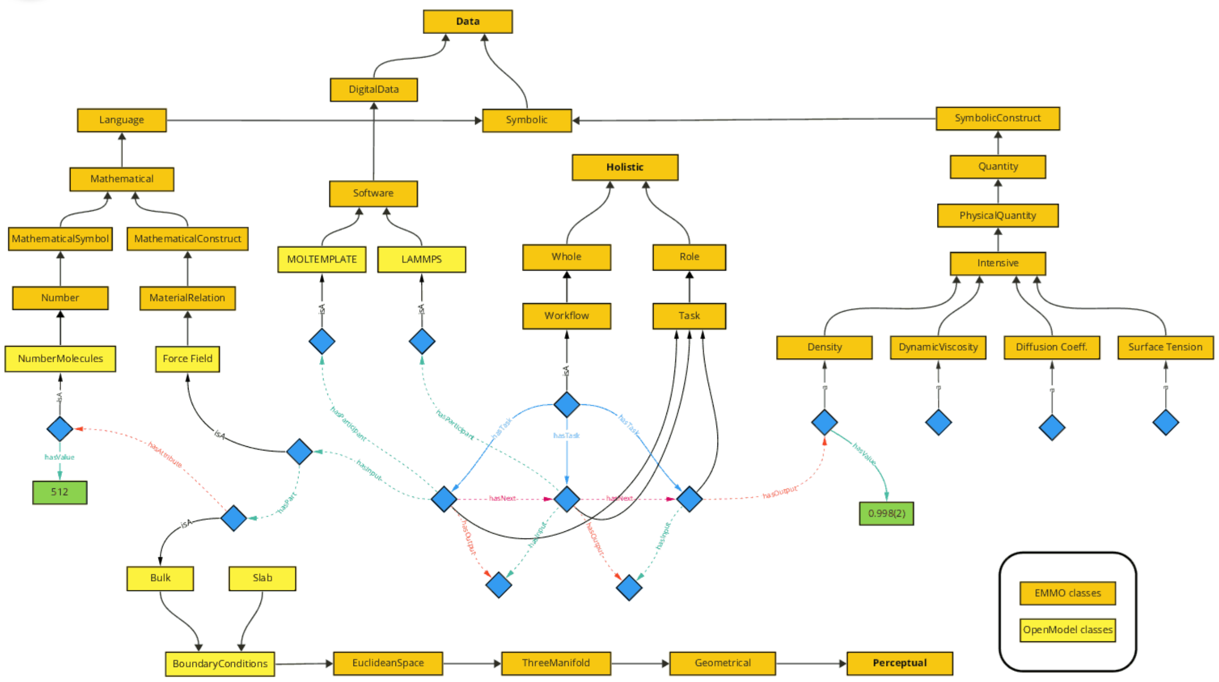


Figure 7: Schematic representation of the Knowledge Base describing the demo number 1.

The ontology provides rules for combining different simulations, if they share certain components, and exploits a modular approach based on units of information. For instance, a fine description of the demo workflow shows that several basic units are shared among different workflows (Figure 8).

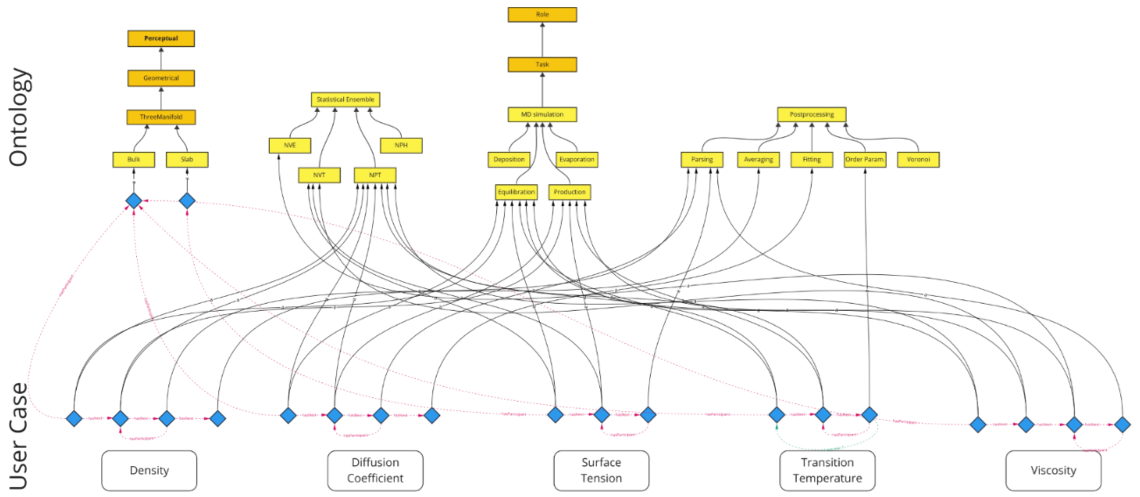


Figure 8: Ontological representation of different workflows sharing reusable components.

The gap analysis performed by the Verification and validation (V&V) services aims at identifying those conditions where the model fails to agree with the reference data (black spots) and regions where lack of reference data does not allow the validation of the model (blind spots). The validation and verification will be performed on 10 solvents using 3 models, namely atomistic, united atoms, and coarse-grained force fields. At the time of writing, the V&V services have been presented as a stand-alone component which is not yet integrated with the other software components of the OpenModel platform.

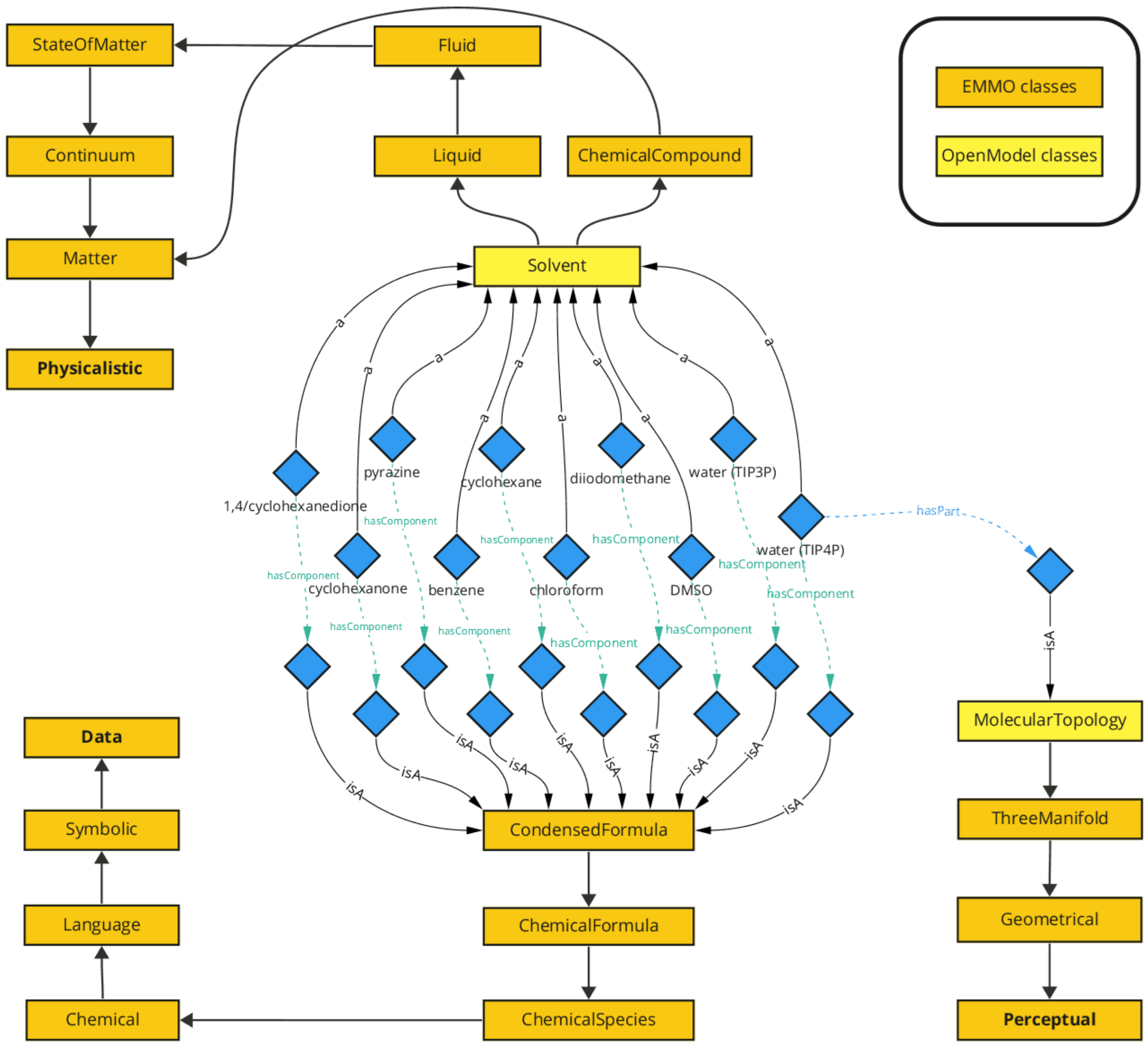


Figure 9: Extension of the demo number 1 for the Verification & Validation services.

The MCO will compare the results with reference data and assign a cost function based on the wall-time and resources used. The use of the MCO is demonstrated on the dataset created for the V&V services (i.e., the solvents listed in Figure 9), where the result of molecular simulations is linked to the model which has been used to describe the system. A taxonomy of computational models and materials relations, shown in Figure 10, is used to specify the atomistic and coarse-grained (mesoscopic) force fields used to compute the same physical observables, thus linking Key Performance Indicators to numerical accuracy. The design of a general knowledge base allows an easy on-boarding of the six industrial cases that are being developed to demonstrate the functionality of the OpenModel infrastructure.

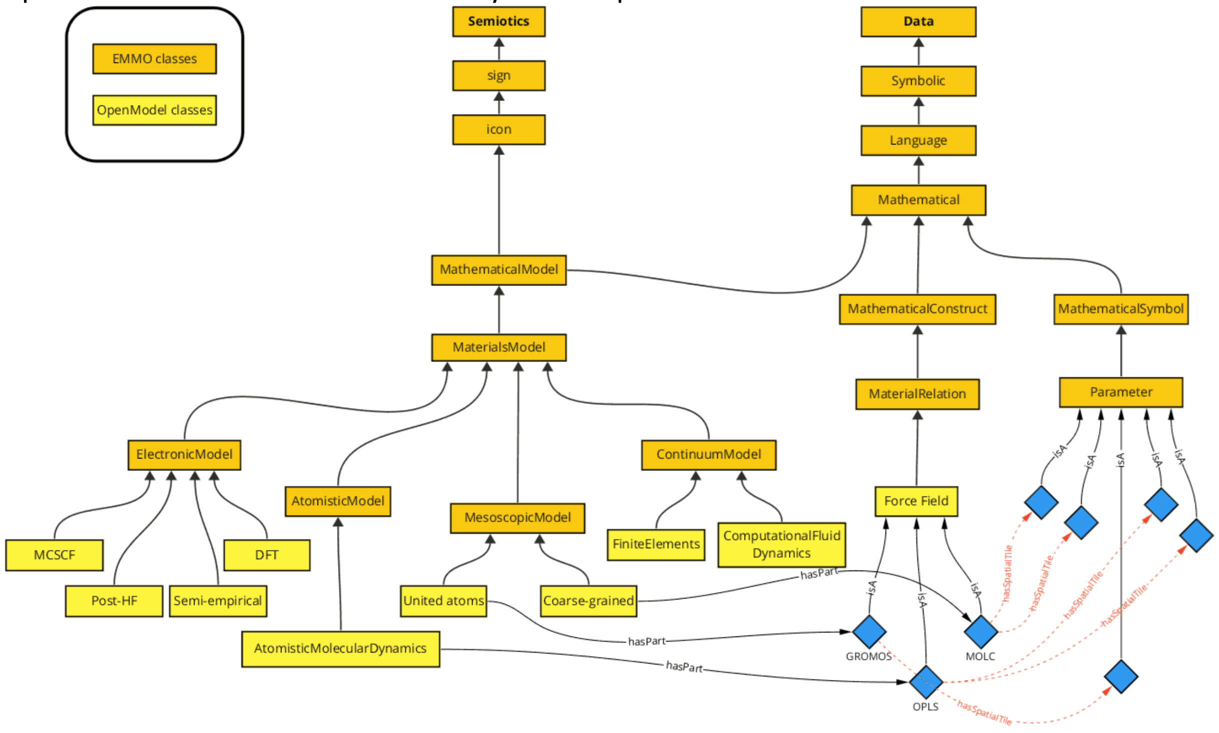


Figure 10: Taxonomy of materials models and relations used to specify the computational models used for the demo number 1.

2 EXECUTION AND DOCUMENTATION

The demo number 1 is an atomistic molecular dynamics (MD) simulation that uses the software Moltemplate to manage the creation of the simulation input, and LAMMPS as the MD engine. This section describes how to install the necessary software on a workstation. The workflow requires a recent version of python >3.9, suggested 3.10, to work. Starting from a knowledge base (i.e. OntoKB) describing the various tasks and their input/output datasets, OntoFlow builds a tree of all the possible workflows leading to the density of a fluid. The output of OntoFlow is a high-level description of the executable workflows, leading to one or more solutions for a user-specified query. The conversion between the ontological representation of a workflow and its serialisation in the YAML format (i.e., the declarative workflow syntax used in ExecFlow) is out of the scope of this demonstration and will be addressed in a future deliverable. Machine-executable scripts describing the three solutions identified by OntoFlow are provided for execution in AiiDA, thus demonstrating the use of ExecFlow and OTEAPI pipelines for the execution of a physics-based simulation.

2.1 INSTALLING THE OPENMODEL SOFTWARE STACK

The local execution of this demonstration requires the following software components, which are the core OpenModel software stack. Here are the steps necessary to install the software.

The original instructions to set up a working AiiDA environment can be found at the following link. Please refer to the official AiiDA documentation for troubleshooting. Here we report the steps to perform a system-wide installation on a Debian/Ubuntu OS. Open a terminal and execute:

```
sudo apt install git python3-dev python3-pip postgresql postgresql-server-dev-all postgresql-client rab-
bitmq-server
```

To avoid clash with locally installed libraries (e.g., VMD), modify the file `$HOME/envs/aiida/bin/activate` by adding the following line:

```
export LD_LIBRARY_PATH=""
```

Then (from AiiDA instructions):

```
$ python -m venv ~/envs/aiida
(aiida) $ verdi quicksetup
(aiida) $ verdi daemon start 2
```

Clone the OpenModel Public repository and install the python modules including *ontoflow*, *execflow*, and *oteapi-dlite*. Open a terminal and execute:

```
git clone https://github.com/H2020-OpenModel/Public.git
cd Public/Deliverable5.5
pip install -e .
```

To avoid changing the names of local files stored in the repository, absolute paths with root `/tmp/ExecFlowDemo` have been used. Independently from where your repository is stored, create a symbolic link as follows:

```
cd /tmp
ln -s /path/to/ExecFlowDemo
```

At the time of writing, these are the versions of the various python packages installed:

```
aiida-core          2.4.0
```

```
aiida-shell          0.3.0
execflow             0.1.0
execflowdemo              0.1.0 #This is installed by pip install in the Public/Deliverable5.5
oteapi-core            0.4.3
oteapi-dlite           0.1.4
otelib               0.3.2
tripper               0.2.6
```

In addition to installing the core components of OpenModel, the softwares used in the D5.5 demo must be installed:

Install Moltemplate: the original instructions can be found at the following link. Open a terminal and execute:

```
git clone https://github.com/jewettaij/moltemplate moltemplate
```

Add the following lines to `~/.bashrc`:

```
export PATH="$PATH:/path/to/moltemplate/moltemplate"
export PATH="$PATH:/path/to/moltemplate/moltemplate/scripts"
```

There are also alternative ways of installing Moltemplate, e.g., through `pip`. See the `INSTALL.md` file in the Moltemplate repository on GitHub.
Install LAMMPS: the easiest option is to download a static linux binary and copy the file to a folder on the `$PATH`, e.g `/usr/local/bin`. Alternatively, LAMMPS can be compiled from the source code using `make` or `cmake`. Follow the instructions here. Note that the LAMMPS binary in the YAML scripts is called `lmp_23Jun22`. You can either create a symbolic link with that name to any other valid LAMMPS binary file, or replace the string `command: "lmp_23Jun22"` in the files `workflow_nopipes.yaml`, `workflow_1oteapi.yaml`, and `workflow_2oteapi.yaml` with the name of your local LAMMPS binary.

Finally, in the current implementation of OntoFlow, StarDog is used as TripleStore. The already installed tripper (installed as a dependency of OntoFlow, see above) has a back-end for StarDock, but the actual StarDog installation needs a license. We refer to StarDog for instructions on this.

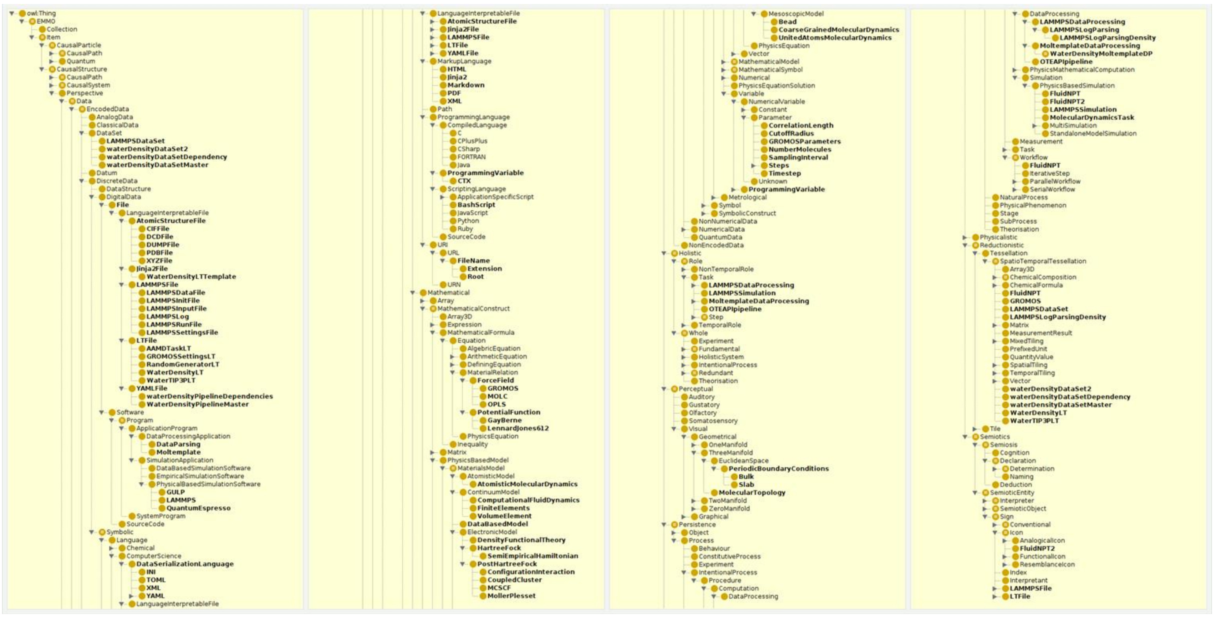
2.2 OPENMODEL ONTOLOGY

A general ontology for OpenModel has been developed, which uses EMMO v1.0.0-beta5 as top reference ontology. It contains the following EMMO modules:

- **mereocausality.**

- **disciplines:** chemistry, computerscience, isq, materials, math, metrology, models,units/si-dimensionalunits.
- **perspectives:** data, holistic, perceptual, perholistic, persistence, perspective, physicalistic, properties, reductionistic, semiotics, standardmodel, symbolic, workflow.

The OpenModel ontology extends EMMO classes with a detailed taxonomy of materials models, software packages, input files, input parameters, boundary conditions, file created and exchanged, programming and scripting languages, variables, and materials, as shown in Figure 11.



```
owl:Thing
  EMMO
    Collection
    Item
      CausalParticle
        CausalPath
        Quantum
      CausalStructure
        CausalPath
        CausalSystem
        Perspective
          Data
            EncodedData
              AnalogData
              ClassicalData
              DataSet
                LAMMPSDataSet
                waterDensityDataSet2
                waterDensityDataSetDependency
                waterDensityDataSetMaster
              Datum
              DiscreteData
                DataStructure
                DigitalData
                  File
                    LanguageInterpretableFile
                      AtomicStructureFile
                        CIFFile
                        DCDFile
                        DUMPFile
                        PDBFile
                        XYZFile
                      Jinja2File
                        WaterDensityLTTemplate
                      LAMMPSFile
                        LAMMPSDataFile
                        LAMMPSInitFile
                        LAMMPSInputFile
                        LAMMPSLog
                        LAMMPSRunFile
                        LAMMPSSettingsFile
                      LTFile
                        AAMDTaskLT
                        GROMOSSettingsLT
                        RandomGeneratorLT
                        WaterDensityLT
                        WaterTIP3PLT
                      YAMLFile
                        waterDensityPipelineDependencies
                        WaterDensityPipelineMaster
                  Software
                    Program
                      ApplicationProgram
                        DataProcessingApplication
                          DataParsing
                          Moltemplate
                        SimulationApplication
                          DataBasedSimulationSoftware
                          EmpiricalSimulationSoftware
                          PhysicalBasedSimulationSoftware
                            GULP
                            LAMMPS
                            QuantumEspresso
                      SystemProgram
                    SourceCode
                Symbolic
                  Language
                    Chemical
                    ComputerScience
                      DataSerializationLanguage
                        INI
                        TOML
                        XML
                        YAML
                      LanguageInterpretableFile
                        AtomicStructureFile
                        Jinja2File
                        LAMMPSFile
                        LTFile
                        YAMLFile
                      MarkupLanguage
                        HTML
                        Jinja2
                        Markdown
                        PDF
                        XML
                      Path
                      ProgrammingLanguage
                        CompiledLanguage
                          C
                          CPlusPlus
                          CSharp
                          FORTRAN
                          Java
                        ProgrammingVariable
                          CTX
                        ScriptingLanguage
                          ApplicationSpecificScript
                          BashScript
                          JavaScript
                          Python
                          Ruby
                        SourceCode
                      URI
                        URL
                          FileName
                            Extension
                            Root
                        URN
                    Mathematical
                      Array
                      MathematicalConstruct
                        Array3D
                        Expression
                        MathematicalFormula
                          Equation
                            AlgebricEquation
                            ArithmeticEquation
                            DefiningEquation
                            MaterialRelation
                              ForceField
                                GROMOS
                                MOLC
                                OPLS
                              PotentialFunction
                                GayBerne
                                LennardJones612
                            PhysicsEquation
                          Inequality
                        Matrix
                        PhysicsBasedModel
                          MaterialsModel
                            AtomisticModel
                              AtomisticMolecularDynamics
                            ContinuumModel
                              ComputationalFluidDynamics
                              FiniteElements
                              VolumeElement
                            DataBasedModel
                            ElectronicModel
                              DensityFunctionalTheory
                              HartreeFock
                                SemiEmpiricalHamiltonian
                              PostHartreeFock
                                ConfigurationInteraction
                                CoupledCluster
                                MCSCF
                                MollerPlesset
                            MesoscopicModel
                              Bead
                              CoarseGrainedMolecularDynamics
                              UnitedAtomsMolecularDynamics
                          PhysicsEquation
                        Vector
                      MathematicalModel
                      MathematicalSymbol
                      Numerical
                      PhysicsEquationSolution
                      Variable
                        NumericalVariable
                          Constant
                          Parameter
                            CorrelationLength
                            CutoffRadius
                            GROMOSParameters
                            NumberMolecules
                            SamplingInterval
                            Steps
                            Timestep
                          Unknown
                        ProgrammingVariable
                    Metrological
                  Symbol
                  SymbolicConstruct
              NonNumericalData
              NumericalData
              QuantumData
            NonEncodedData
          Holistic
            Role
              NonTemporalRole
              Task
                LAMMPSDataProcessing
                LAMMPSSimulation
                MoltemplateDataProcessing
                OTEAPIpipeline
                Step
              TemporalRole
            Whole
              Experiment
              Fundamental
              HolisticSystem
              IntentionalProcess
              Redundant
              Theorisation
          Perceptual
            Auditory
            Gustatory
            Olfactory
            Somatosensory
            Visual
              Geometrical
                OneManifold
                ThreeManifold
                  EuclideanSpace
                    PeriodicBoundaryConditions
                      Bulk
                      Slab
                  MolecularTopology
                TwoManifold
                ZeroManifold
              Graphical
          Persistence
            Object
            Process
              Behaviour
              ConstitutiveProcess
              Experiment
              IntentionalProcess
                Procedure
                  Computation
                    DataProcessing
                      LAMMPSDataProcessing
                        LAMMPSLogParsing
                          LAMMPSLogParsingDensity
                      MoltemplateDataProcessing
                        WaterDensityMoltemplateDP
                      OTEAPIpipeline
                    PhysicsMathematicalComputation
                    Simulation
                      PhysicsBasedSimulation
                        FluidNPT
                        FluidNPT2
                        LAMMPSSimulation
                        MolecularDynamicsTask
                        MultiSimulation
                        StandaloneModelSimulation
                  Measurement
                  Task
                  Workflow
                    FluidNPT
                    IterativeStep
                    ParallelWorkflow
                    SerialWorkflow
              NaturalProcess
              PhysicalPhenomenon
              Stage
              SubProcess
              Theorisation
          Physicalistic
          Reductionistic
            Tessellation
              SpatioTemporalTessellation
                Array3D
                ChemicalComposition
                ChemicalFormula
                FluidNPT
                GROMOS
                LAMMPSDataSet
                LAMMPSLogParsingDensity
                Matrix
                MeasurementResult
                MixedTiling
                PrefixedUnit
                QuantityValue
                SpatialTiling
                TemporalTiling
                Vector
                waterDensityDataSet2
                waterDensityDataSetDependency
                waterDensityDataSetMaster
                WaterDensityLT
                WaterTIP3PLT
            Tile
          Semiotics
            Semiosis
              Cognition
              Declaration
                Determination
                Naming
              Deduction
            SemioticEntity
              Interpreter
              SemioticObject
              Sign
                Conventional
                Icon
                  AnalogicalIcon
                  FluidNPT2
                  FunctionalIcon
                  ResemblanceIcon
                Index
                Interpretant
                LAMMPSFile
                LTFile
```

Figure 11: Taxonomy tree of the OpenModel ontology, nested into the EMMO top-level ontology.

The new classes encode the meaning of methods, parameters, and their connections. For example, the abstract concept of a force field such as GROMOS refers to a collection of potential functions and parameters specific for this materials model. A possible serialisation of this particular materials model can be expressed as a file which has spatial overlap with specific parameters and keywords. The keywords also bind the file's syntax to an interpreter (e.g., a particular molecular dynamics code).

The ontology's level of details is needed to ensure that there is enough information to reconstruct a machine-executable workflow, and to provide a documentation of the tasks and methodologies used in different use cases. The OpenModel ontology has been designed to describe not only the workflow used to demonstrate the platform, but also the six success stories developed on WP5. By having different workflows and tasks described by the same ontology and stored in the same knowledge base, it will be possible to infer new connections between materials models and to create new workflows that can solve increasingly more complex problems.

2.3 ONTOFLOW

OntoFlow serves as a tool for designing and constructing workflows using ontological classes and relations. It has the capability to automatically identify and recommend the most appropriate combination of models, tools, and actions needed to achieve a specific desired outcome based on given inputs. To achieve this, OntoFlow utilises a knowledge base to navigate and match various workflows based on their underlying meanings, thereby comprehending their connections. Among the retrieved ones, the best workflow can be ascertained using uncomplicated cost-based guidelines or more complex approaches like Multi-Criteria Optimizers (MCO).

The OntoFlow framework consists of two key components (Figure 12): the knowledge base (OntoFlowKB) and the decision-making component (OntoFlowDM). OntoFlowKB is essentially a triplestore database that is purpose-built to store and manage semantic information, including the associations between ontological concepts. It draws upon the advancements made in the related OntoTrans project. OntoFlowDM, on the other hand, functions as the core element responsible for making decisions with the workflows. This involves integrating various Multi-Criteria Optimizers (MCOs) to facilitate the decision-making process.

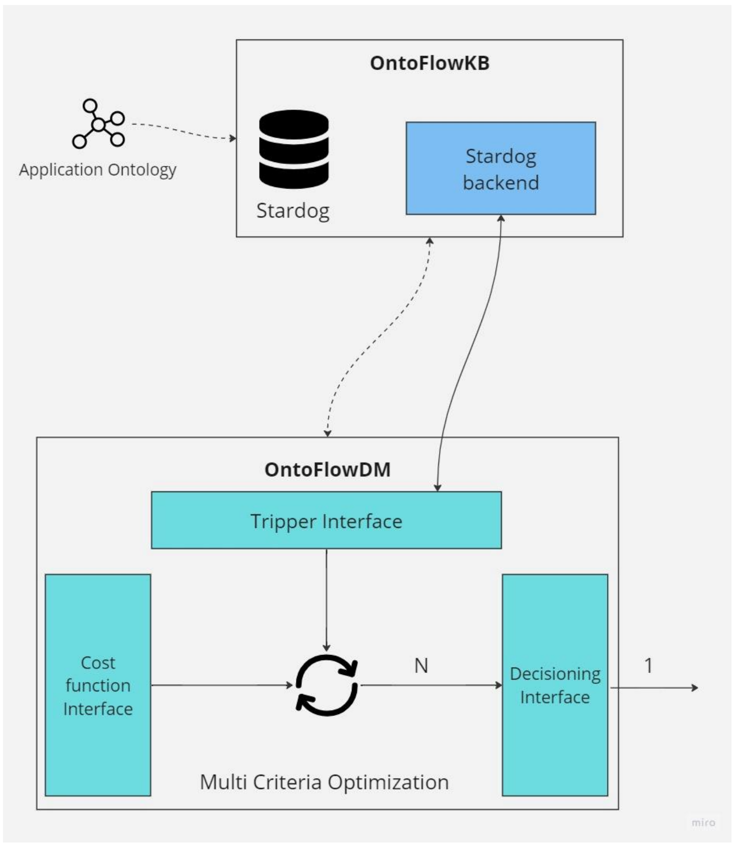
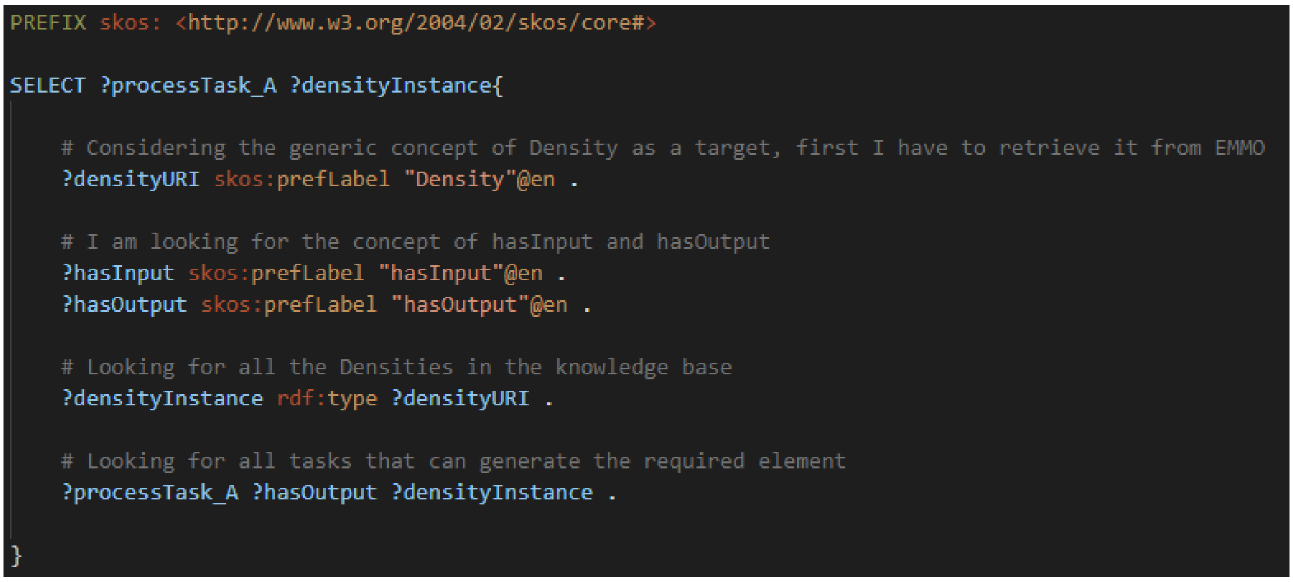


Figure 12: Illustration of OntoFlow framework consisting of two key components: the knowledge base (OntoFlowKB) and the decision-making component (OntoFlowDM)

Going into a bit more detail about the components involved in this demonstration, OntoFlowKB is a triplestore-based component that offers the ability not only to store information related to ontological concepts of workflows (general and otherwise) but also to query them through the use of SPARQL queries and infer new ones thanks to the appropriate engine. In addition, the implementation of OntoFlowKB attempts to free itself from the technology concretely used for the triplestore by exploiting abstraction interfaces that allow it to interact in a completely agnostic manner. This is done by means of the Tripper, a tool developed by SINTEF.

The triplestore on which this demonstration is based is Stardog, a commercial solution that has already been adopted and validated in the OntoTrans project.

OntoFlowKB represents the main tool on which the route-finding algorithm implemented within OntoFlowDM is based. This algorithm, comprised into the tripper package, allows the retrieval of all possible routes that manage to generate a precise output, starting from what are the inputs available or to be requested externally. In its current version, the algorithm works backward from output to final inputs. Among the advantages of this approach is the ability to be able to explore only those ontology branches that are actually connected to the desired output; this is advantageous in large search spaces such as in the case of a knowledge base storing diverse and apparently unrelated materials models. The ontology navigation, which allows for route reconstruction, is done based on how entities, defined by ontology objects, can be generated from model computations, simulations, or tasks. The algorithm assumes that such information is modelled through the EMMO ontology and, therefore, the queries that are performed are based on the use of **hasInput** and **hasOutput** predicates to reconstruct the chain of relationships. Optionally, other predicates are also used to navigate the ontology, this is the case with **subclassOf** or **instanceOf**, which are useful in all those cases where there are additional levels of structure and abstraction between input and output.



```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

SELECT ?processTask_A ?densityInstance{

    # Considering the generic concept of Density as a target, first I have to retrieve it from EMMO
    ?densityURI skos:prefLabel "Density"@en .

    # I am looking for the concept of hasInput and hasOutput
    ?hasInput skos:prefLabel "hasInput"@en .
    ?hasOutput skos:prefLabel "hasOutput"@en .

    # Looking for all the Densities in the knowledge base
    ?densityInstance rdf:type ?densityURI .

    # Looking for all tasks that can generate the required element
    ?processTask_A ?hasOutput ?densityInstance .

}
```

The algorithm terminates when there are no more elements that can be derived from Task, thus defining the input dataset needed for the entire process.

2.4 ONTOFLOW AND ONTOKB EXECUTION

This section illustrates how to use OntoFlow in practice. In this example of usage, OntoFlow is used to retrieve all the possible pathways leading to the computation of the density of a fluid (water, in this case). The semantic representation of materials modelling workflows, data, and computational methods is based on a domain ontology developed using EMMO v1.0.0-beta5 as top reference.

A simple query to the OntoKB is presented in the file `application.py`. Basically, the user asks for any route leading to the computation of the density of a fluid, expressed by the ontological class with IRI `http://emmo.info/emmo#FluidDensity`. From a terminal, execute:

```
cd Public/Deliverable5.5/ontoKB
python application.py
```

The result of this query is a branched tree of the possible pathways leading to the computation of the density of a fluid. A graphical representation of the various paths is shown in Figure 13.

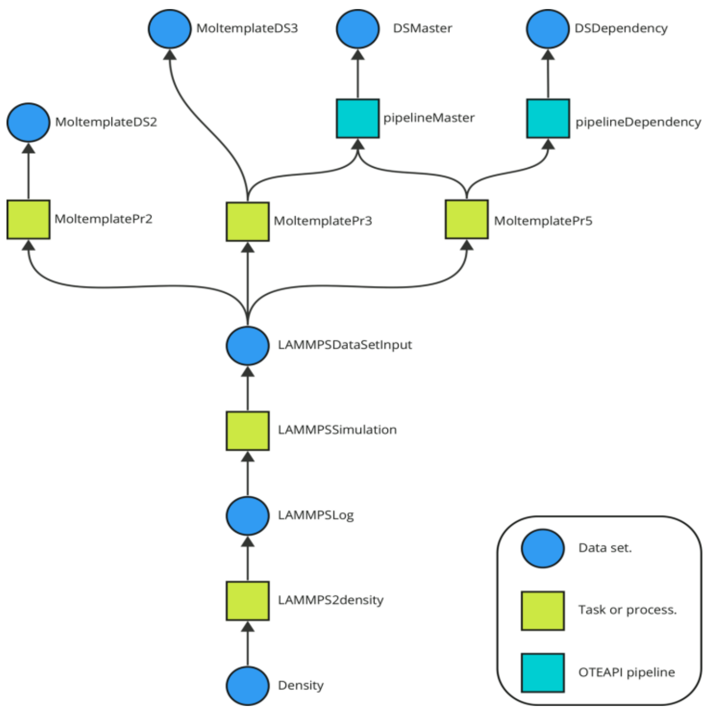


Figure 13: Graphical representation of all the possible workflows leading to the computation of the density of a fluid.

The demonstration of OntoFlow is based on a python script which is structured in the following way. The first step, not strictly related to OntoFlow, is to store the ontology information and related concepts within OntoKB. This can be done programmatically by exploiting the abstraction interface described earlier.

```
## Init triplestore
Triplestore.remove_database(backend="stardog", database="d55_usecase", triplestore_url="http://localhost:5820")
Triplestore.create_database(backend="stardog", database="d55_usecase", triplestore_url="http://localhost:5820")
ts = Triplestore(backend="stardog", triplestore_url="http://localhost:5820", database="d55_usecase")
```

A new triplestore database is initialised with the Triplestore class, which represents the main abstraction element between tripper and the OntoFlow architecture, and allows to perform all the classical operations of a triplestore without being bound to a specific implementation. In the case of the code snippet in the figure, the Triplestore class is first used to create a database called *d55_usecase* and then to obtain an instance (e.g. ts) that will later be used to interact with the newly created database. Note how the Triplestore class needs some configuration information related to the location of the triplestore as well as the technology used underneath, in this case Stardog.

```
# Add ontology and individuals
ts.parse(os.path.join(ONTOLOGY_FOLDER, "..", ONTOLOGY_FILENAME), format="turtle")
ts.parse(os.path.join(ONTOLOGY_FOLDER, "..", INDIVIDUALS_FILENAME), format="turtle")
```

Next, two files in the turtle syntax are loaded, the first one being an ontology containing the schema of the classes supporting the use case, and the second containing the instances (individuals) of the relevant classes, thus forming a knowledge base. The saving is done through a parse method executed on the previously created instance of Triplestore that takes, among possible arguments, the name of a file.

```
engine = OntoFlowDMEngine(triplestore = ts, cost_file = COST_FILE, mco_interface = mco_strategy)
```

The core of the OntoFlow library is the OntoFlowDMEngine class, with which the search algorithm can be used. This class needs first to be instantiated with some configuration parameters, i.e. the triple-store instance that the engine will use to perform the necessary queries, a file containing the cost definition, and the MCO interface that will be used in the process. Since the emphasis of this demonstration is on the identification of possible workflows from a query, no additional details will be provided on the cost functions and MCO optimisation. This will be the subject of future demonstations.

```
target_IRI = "http://emmo.info/emmo#FluidDensity"
sources_IRIs = {}
route = engine.getmappingroute(target_IRI, sources_IRIs)
```

The actual search is done via the engine's **getmappingroute** method, to which the URI of the ontology object for which the routes are to be obtained must be passed. Optionally, a list of strings can also be passed and in this case the algorithm will be executed iteratively on each of them.
At the end of the search, a dictionary is provided as output containing for each URI passed as input, the set of routes obtained in tree form (this means that any nodes in common to the routes are repeated).

2.5 WORKING WITH OTEAPI PIPELINES

In the OpenModel platform, input and output datasets are managed through pipelines that provide mappings to semantic concepts and interoperability between different data serialisations. In the demo number 1, the input dataset is a simple YAML file containing the following key-value pair:

```
moltemplate_input:
 meta: http://ontotrans.eu/meta/0.1/FluidNPTInput
 dimensions: {}
 properties:
  run: "water_aa"
  ts: 2
  temp: 298.15
  p: 1.0
  cutoff: 13.
  cl: 40
  s: 5
  prod: 400
  force_field1: "water.lt"
  aa_atb: "aa_atb.lt"
  aa_tasks: "aa_tasks.lt"
  TYPE1: TIP3
  nmols1: 64
```

The input dataset is described by a **data model (in this case it is actually a serialized instance of the data model)**, which is a JSON file with the following structure:

```
{

 "uri": "http://ontotrans.eu/meta/0.1/FluidNPTInput",
 "description": "Data model describing input variables for the simulation",
 "dimensions": [],
 "properties": {
   "run": {
      "type": "str",
      "description": "Name of the root files created by MOLTEMPLATE and LAMMPS."
   },
   "ts": {
      "type": "int",
      "description": "Time step for numerical integration.",
      "unit": "fs"
   },
   "temp": {
      "type": "float",
      "description": "Temperature",
      "unit": "K"
   },
   "p": {
      "type": "float",
      "description": "Pressure.",
      "unit": "atmospheres"
    },
    "cl": {
      "type": "int",
      "description": "Correlation length, used to sample the thermodynamic output."
    },
    "s": {
      "type": "int",
      "description": "Sample interval, used to sample the thermodynamic output."
    },
    "prod": {
      "type": "int",
      "description": "Number of steps to be computed in the current MD simulation."
    },
    "force_field1": {
      "type": "str",
      "description": "Filename of the force field for the 1st molecule type in the simulation."
    },
    "aa_atb": {
      "type": "str",
      "description": "LT file containing the macros for the GROMOS-ATB force field."
    },
    "aa_tasks": {
      "type": "str",
      "description": "LT file containing the macros defining various simulation styles and thermodynamic
outputs."
    },
    "TYPE1": {
      "type": "str",
      "description": "Name of the 1st molecule type in the simulation, as defined in the corresponding LT
file."
    },
    "nmols1": {
      "type": "int",
      "description": "Number of molecules of type 1."
    }
  }
}
```

The data model is constructed with:

- A URI uniquely identifying the data model. The data model filename does not have to match the URI label (domain/version/label) as DLite parse all the data models in the source directories. However, it makes life easier to have consistent filenames and labels.
- A human readable description of the data model.

- A set of dimensions used by the properties. In our case all properties are scalar, so no dimensions has to be specified.
- An entry for each property in the data set, containing:
    - property name
    - property type, ex: "str", "int", "float", "boolean", ...
    - property shape (not needed for scalars)
    - property description", which is a human-readable explanation of the datum

The pipeline used to load the input dataset in the demo number 1 follows:

```
version: 1
strategies:
 - dataresource: load_data
   downloadUrl: file:///tmp/ExecFlowDemo/meso_multi_sim_demo/input/moltemplate_input3.yml
   mediaType: application/vnd.dlite-parse
   configuration:
     driver: yaml
     label: parameters_input

 - mapping: mappings
   mappingType: triples
   prefixes:
     emmo: http://emmo.info/emmo#
     om:  http://emmo.info/emmo/domain/openmodel#
     map:  http://emmo.info/domain-mappings#
     fluid: http://ontotrans.eu/meta/0.1/FluidNPTInput#
   triples:
     - ["fluid:run", "map:mapsTo", "om:Root"]
     - ["fluid:ts", "map:mapsTo", "om:Timestep"]
     - ["fluid:temp", "map:mapsTo", "om:ThermodynamicTemperature"]
     - ["fluid:p", "map:mapsTo", "om:Pressure"]
     - ["fluid:cutoff", "map:mapsTo", "om:CutoffRadius"]
     - ["fluid:cl", "map:mapsTo", "om:CorrelationLength"]
     - ["fluid:s", "map:mapsTo", "om:SamplingInterval"]
     - ["fluid:prod", "map:mapsTo", "om:ProductionSteps"]
     - ["fluid:force_field1", "map:mapsTo", "om:WaterTIP3PLT"]
     - ["fluid:aa_atb", "map:mapsTo", "om:GROMOSSettingsLT"]
     - ["fluid:aa_tasks", "map:mapsTo", "om:AAMDTaskLT  "]
     - ["fluid:TYPE1", "map:mapsTo", "om:MolecularType"]
     - ["fluid:nmols1", "map:mapsTo", "om:NumberMolecules"]

 - function: cuds2datanode
   functionType: aiidacuds/cuds2datanode
   configuration:
     names: from_cuds
 - function: write_masterfile_lt
   functionType: application/vnd.dlite-generate
   configuration:
     driver: template
     location: /tmp/water_aa.lt
     options: "template=/tmp/ExecFlowDemo/meso_multi_sim_demo/case_aiida_wrapper/lt_wa-
ter_aa.template;engine=jinja"
     label: parameters_input

 - function: file2collection
   functionType: aiidacuds/file2collection
   configuration:
     path: /tmp/water_aa.lt
     label: lt_input

pipelines:
 pipe: load_data | mappings | write_masterfile_lt | file2collection |
 cuds2datanode
```

This pipeline creates two AiiDA DataNodes with labels `parameters_input` and `lt_input`, which are then imported into the declarative workchain script. The ontological classes to which the input parameters are mapped to are defined in the OpenModel domain ontology and are explained in the next section. Data pipeline specifications:

- Each "-" specifies the type of strategy used (in this case a data_resource, a mapping and three function strategies), acting as filters in the data pipe.
- "load_data" is an ID attached to a strategy to read the input dataset (this example is the file `moltemplate_input3.yml`) and is used to put the filters together in the specificaltion of the partial pipeline (at the bottom of the file).
- The "downloadUrl" specifies where to fetch the data
- "mediaType" specifies which strategy to use to parse the input dataset.
- For this strategy (mediaType) there are additional parameters specified: the `driver:` dlite-parse, and a `label:` that points to the parsed dataset. Later in the pipe, the dataset can be assessed with the label `input_dataset1`.
- In the "mapping" strategy a local name is given (mappings)
    - The mappings use a set of prefixes. These are not required, but make the readability easier for humans. The prefixes are:the URI of the EMMO ontology (not used in this example).
    - the URI of the OpenModel ontology.
    - the URI of the mapping ontology, developed in the OntoTrans project.
    - the URI of the JSON data model, described above.

- The mappings the are provided as a list of "subject · predicate · object" triplets specifying the mapping relations between the data model (in this example, fluid:) and ontological concepts. The list of subjects to map **must** be consistent with the input dataset, e.g. it must have the same properties.
    - Furthermore, there is a strategy for generating the file needed for the software. Ituses the functionType: application/vnd.dlite-generate to write the dataset into a TEMPLATE FILE. The configuration: specifies additional parameters:“driver”: which dlite storage plugin to use. template.py replaces variables in a template file, written in the jinja or python-format template syntax.“location”: the destination of the output file.
    - “options” for the the path of the template file to be used, and the engine used to interpret the template file (possible values: jinja or format).“label” is a reference to the label created by the dataresource strategy. The result is that the input dataset is substituted into the template file, resulting in a serialised output file.
    - The function “file2collection” takes the file from the path: and puts it into a collection, i.e. it creates an AiiDA data node from the input file written by a previous filter. The configuration: specifies additional parameters:
    - “path” to the file written by the previous filter, specified with the location key.“label” is the ID attached to the datanode, that is retrieved in the execflow.oteapipipeline step using the from_cuds keyword.

This pipeline uses the Jinja2 syntax to turn an input file into a generic template, whose values are provided separately from the input dataset. In this example, we use the Jinja delimiters for expressions to print to the template output: {{ ... }}. For example:

Template

```
# This is a LAMMPS TEMPLATE file,
# with hard-coded values replaced
# with Jinja web template syntax.

write_once("In Init"){
 # Input variables.

 variable run  string {{ run }}
 variable ts   equal  {{ ts }}
 variable tf   equal  {{ temp }}
 variable p    equal  {{ p }}
 variable cl   equal  {{ cl }}
 variable s    equal  {{ s }}
 variable prod equal  {{ prod }}
```

Output

```
# This is a LAMMPS TEMPLATE file,
# with hard-coded values replaced
# with Jinja web template syntax.

write_once("In Init"){
 # Input variables.

 variable run  string water_aa
 variable ts   equal  2
 variable tf   equal  298.15
 variable p    equal  1.0
 variable cl   equal  40
 variable s    equal  5
 variable prod equal  400
 # PBC                                          # PBC
 boundary p p p                                 boundary p p p
}                                              }

# Import the force field.                      # Import the force field.
import {{force_field1}}                        import water.lt
import {{aa_atb}}                              import aa_atb.lt
ff = new atb_long                              ff = new atb_long

# Create the molecules.                        # Create the molecules.
sol = new {{TYPE1}}[{{nmols1}}]                sol = new TIP3[64]

# Create the initial velocity.                 # Create the initial velocity.
write_once("In Run"){                          write_once("In Run"){
 variable r format r1 %.0f                      variable r format r1 %.0f
 velocity all create \$\{tf\} \$r dist gaussian velocity all create \$\{tf\} \$r dist gaussian

 # Apply the SHAKE algorithm                    # Apply the SHAKE algorithm
 # to hydrogen atoms.                           # to hydrogen atoms.
 fix SHK all shake .0001 10 0 m 1.0079 a 1      fix SHK all shake .0001 10 0 m 1.0079 a 1
}                                              }

# Define the task to execute.                  # Define the task to execute.
import {{aa_tasks}}                            import aa_tasks.lt
task = default                                 task = default
run = new aa_npt                               run = new aa_npt
```

The master file is written from the template and stored to an AiiDA DataNode using the file2collection function. In the main workflow, the parameters_input and lt_input CUDSs are retrieved and passed to the context with a different name. The individual keys of the input dataset moltemplate_input can be accessed later in the workflow, e.g., via the ctx.parameters.run variable. Note that this dataset is called moltemplate_input in the source, parameters_input in the pipeline and the corresponding CUDS, and finally parameters when it is passed to the context. Its internal structure, however, remains unchanged. Another data pipeline (pipeline_dependencies.yaml) is used to read another input dataset containing the URI of various files that are needed by a following task. The mapping strategy links each of those URIs to an ontological class describing the meaning of corresponding file, while the values of each URI are stored in an AiiDA DataNode named moltemplate_includes which is then passed to the context simply as includes and finally accessed, e.g. as "{{ ctx.includes.md_tasks }}".

2.6 EXECFLOW

The routes identified by OntoFlow are a high-level description of workflows that can be executed by a workflow manager such as AiiDA. As there is not yet a software component converting this representation to a syntax that can be readily executed, individual workflows implementing each of the three routes have been written by hand in the declarative syntax developed for ExecFlow. To run these examples, start the AiiDA environment and the Verdi daemon. Open a terminal and execute:

```
source ~/envs/aiida/bin/activate
verdi daemon start
```

The three workflows are executed from the AiiDA prompt with `verdi` running in the background:

```
cd /path/to/Public/Deliverable5.5/demo1
python run_workflow.py       workflow_nopipes.yaml
python run_workflow_pipes.py workflow_1oteapi.yaml
python run_workflow_pipes.py workflow_2oteapi.yaml
```

The AiiDA nodes are inspected with the commands:

```
verdi process list -a
verdi node show 5601 #Use the numbers you see from the output above to investigate
verdi node attributes 5601
```

2.7 DECLARATIVE WORKFLOW SYNTAX

ExecFlow uses AiiDA to execute workflows and a declarative syntax in YAML format to specify them. A schematic example of a declarative workchain follows, where OTEAPI pipelines are used to load the input dataset and store the output dataset using the `execflow.oteapipipeline` function, and an external software is executed with the wrapper `execflow.exec_wrapper`.

```
---
steps:
 # Create a data pipeline reading the input dataset.
 - workflow: execflow.oteapipipeline
   inputs:
     pipeline:
       $ref: file:pipeline_input.yml
     from_cuds:
        - ...
   postprocess:
```

```
  - ...

 # Execute a task.
 - workflow: execflow.exec_wrapper
   inputs:
     command: "mysoftware.exe"
     arguments:
       - ...
     files:
       ...
     outputs:
       - ...
   postprocess:
     - ...

 # Output parsing.
 - workflow: execflow.oteapipipeline
   inputs:
     pipeline:
       $ref: file:pipeline_output.yml
     to_cuds:
       - cuds1
     cuds1: "{{ ctx.density }}"
...
```

More specifically, the workflow describing the demo number 1 with two data pipelines follows :

```
---
steps:

 # Create a data pipeline reading the input dataset.
 - workflow: execflow.oteapipipeline
   inputs:
     pipeline:
       $ref: file:pipeline_waterdensity.yml
     from_cuds:
       - parameters_input
       - lt_input
   # The first oteapi pipeline creates the collection and parameter set,
   # which are then passed to the ctx.
   postprocess:
     - "{{ ctx.current.outputs['collection_id'] | to_ctx('collection_uuid') }}"
     - "{{ ctx.current.outputs.results['parameters_input']|to_ctx('parameters') }}"
     - "{{ ctx.current.outputs.results['lt_input']|to_ctx('master_lt') }}"

 # Another pipeline retrieving the file includes, resolved from the URIs
 # pointing to various input files.
 - workflow: execflow.oteapipipeline
   inputs:
     pipeline:
       $ref: file:pipeline_dependencies.yml
     from_cuds:
       - moltemplate_includes
   postprocess:
     - "{{ ctx.current.outputs['collection_id']|to_ctx('collection_uuid') }}"
     - "{{ ctx.current.outputs.results['moltemplate_includes']|to_ctx('includes') }}"

 - workflow: execflow.exec_wrapper
   inputs:
     command: "moltemplate.sh"
     arguments:
       - "-atomstyle"
       - "full"
       - "-overlay-all"
       - "-pdb"
       - "{in_pdb}"
       - "{in_lt}"
     files:
       aa_atb:
         filename: "aa_atb.lt"
         template: "{{ ctx.includes.gromos_settings }}"
       aa_tasks:
         filename: "aa_tasks.lt"
         template: "{{ ctx.includes.md_tasks }}"
       random:
         filename: "random_init.lt"
         template: "{{ ctx.includes.random_init }}"
       force_field:
         filename: "water.lt"
         template: "{{ ctx.includes.force_field }}"
       in_pdb:
         filename: "input.pdb"
         template: "{{ ctx.includes.input_structure_pdb }}"
       in_lt:
         filename: "{{ ctx.parameters.run }}.lt" # i.e. "water_aa.lt"
         node: "{{ ctx.master_lt }}"
     outputs:
       - water_aa.in
       - water_aa.data
       - water_aa.in.settings
       - water_aa.in.run
       - water_aa.in.init
     postprocess:
       - "{{ ctx.current.outputs['remote_folder']|to_ctx('lammps_dir') }}"
       - "{{ ctx.current.outputs['water_aa_in']|to_ctx('lammps_in') }}"
       - "{{ ctx.current.outputs['water_aa_data']|to_ctx('lammps_data')}}"
       - "{{ ctx.current.outputs['water_aa_in_settings']|to_ctx('lammps_settings') }}"
       - "{{ ctx.current.outputs['water_aa_in_run']|to_ctx('lammps_run') }}"
       - "{{ ctx.current.outputs['water_aa_in_init']|to_ctx('lammps_init') }}"

 - workflow: execflow.exec_wrapper
   inputs:
     command: "lmp_23Jun22"
     arguments:
       - "-in"
       - "{in}"
       - "-l"
       - "water_aa.log"
     files:
       in:
         filename: "water_aa.in"
         node: "{{ ctx.lammps_in }}"
       data:
         filename: "water_aa.data"
         node: "{{ ctx.lammps_data }}"
       settings:
         node: "{{ ctx.lammps_settings }}"
         filename: "water_aa.in.settings"
       run:
         node: "{{ ctx.lammps_run }}"
         filename: "water_aa.in.run"
       init:
         node: "{{ ctx.lammps_init }}"
         filename: "water_aa.in.init"
     outputs:
       - water_aa.log
       - water_aa.dcd
   postprocess:
     - "{{ ctx.current.outputs['water_aa_log']|to_ctx('lammps_log') }}"

 - calcjob: execflowdemo.lammps.density
   inputs:
     log: "{{ ctx.lammps_log }}"
   postprocess:
     - "{{ ctx.current.outputs['density'] | to_results('density') }}"
     - "{{ ctx.current.outputs['density'] | to_ctx('density') }}"
...
```

This workflow contains three tasks (i.e., computations) and two data pipelines. The first two tasks are based on the execflow.exec_wrapper, which is a generic wrapper based on aiida-shell allowing the execution of any software that is accessible through the command-line interface (CLI). The wrapper needs the following inputs:

- "command" specifies the name of the software. The command should be accessible via the $PATH variable. Alternatively, an absolute path can be used to point to a specific binary file.
- "arguments" specifies a list of arguments that are passed to the binary. This list is written to an aiida-shell script using double quotes, therefore every line will be parsed as a single string.
- "files" contains a nested collection of keys, each specifying an input file that will be written in a remote directory for execution. The filename: key specifies the name of the file to be copied in the remote folder. The template: key specifies the source file to be read. The node: key specifies the content of the file through an AiiDA data node, previously created.
- "outputs" specifies a list of the output files created. Each of the file names in the list is transformed into an AiiDA SingleFile data node, whose name is created by replacing dots (.) with a underscore (_) sign.

The last task is a calcjob: used to parse the output file, specified as an AiiDA data node, and producing a scalar value (the averaged density) as output. In this workflow, the first pipeline writes an input dataset (parameters_input) into a template file, creating an input file (lt_input) with a specific syntax. The input dataset and the input file are stored as AiiDA DataNodes, passed to the context, and accessed later into the declarative workchain. The second pipeline creates an AiiDA DataNode containing a list of URIs pointing to the files that are needed later in the workflow. Note that the input files in the first task (e.g. the moltemplate.sh execution) are written into a remote folder with a specific filename:. If the source of the file is specified with the template: key, then the argument is the URI of the source file[2]. If the source of the file is specified with the node: key , then the content of the remote file is copied from an AiiDA data node.

3 CONCLUSION

This deliverable describes the overall OIP infrastructure and provides a proof-of-concept demonstration of the OIP main components and how they holistically work together. The proof-of-concept validation has been demonstrated with a simple use case based on molecular dynamics simulation of liquid water at room temperature and pressure.

[2] The "template:" key is normally used to read a template file with JINJA2 elements which are substituted with keys from the "parameters:" key, as in the example workflow_nopipes.yaml. However, it can also be used to simply copy and paste the content of a source file to the destination.

The value proposition of OpenModel OIP is the use of semantic technologies to search for and identify patterns in data that might otherwise be difficult to discern. By formally defining workflows and material science concepts through ontologies, the platform can help users identify the most efficient solution to a given problem and discover new sources of information and insights. Therefore, the first demonstration of the OIP focuses on showing the semantic layer's functionality and how it will drive commercial exploitation. The Product Manager created a sandbox comprising a series of simulation workflows with increasing complexity (Agile style) based on open-source software and described in a Knowledge Base (KB) with the following characteristics:

1. Capable of describing materials and processes.
2. Based on reusable components.
3. General and diverse, covering a wide range of use cases.

This KB uses the EMMO top-level ontology to provide compliance with other European projects and interoperability with third-party software and tools. It includes a general taxonomy to accommodate the materials, workflows, and computational models for the present demonstration, the six success stories, and future use cases.

4 ACKNOWLEDGMENT

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