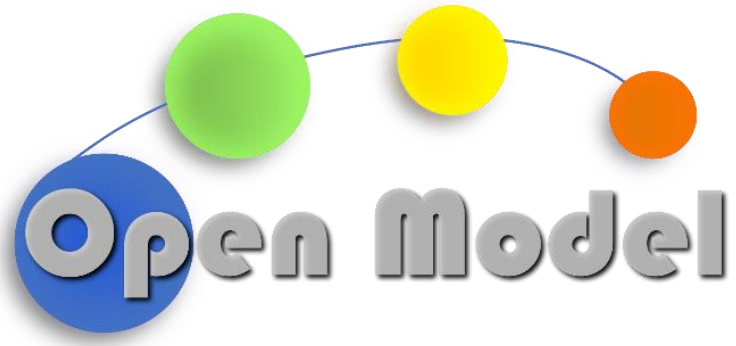
Otello M Roscioni (GCL)



Describing, retrieving, and executing materials modelling workflows on the OpenModel platform

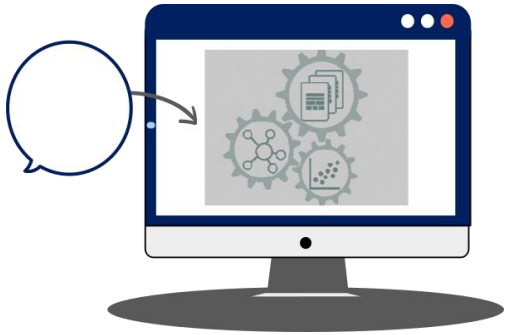
18 September 2024

The OpenModel project

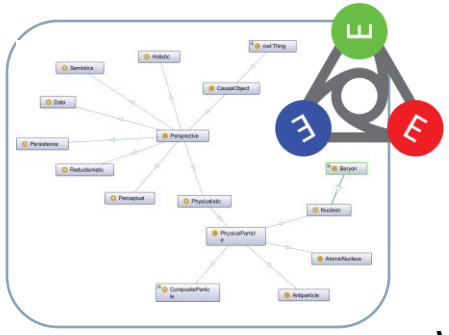
"Where have I seen this before?"

OpenModel is a **management system** for materials modelling based on **semantic** technologies:

- Interoperability between physics-based models, solvers, third-party software, post-processors, and databases via a semantic layer.
- Automatic generation and execution of materials modelling workflows.
- FAIR data creation.



User front-end



Ontologies

AiiDA

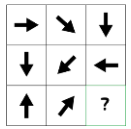
SimPhoNy

DLite

Workflows and data management

DB

Software

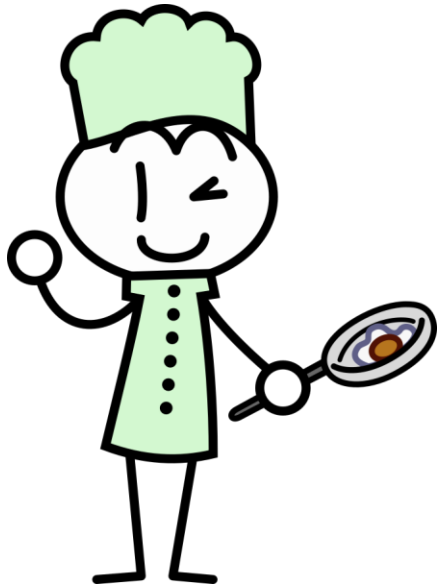


Reasoning

V&V Services

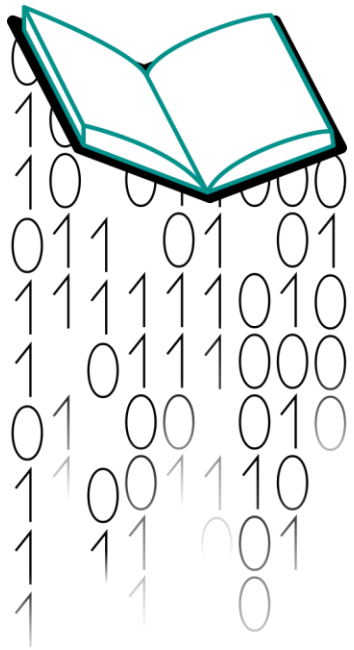
Third-party tools

Semantics in Materials Modelling



OpenModel is a **virtual kitchen** for materials modelling workflows.

1. We are compiling a digital **cookbook** for materials science.
2. The **recipes** capture the knowledge in different domains and use a standard vocabulary.
3. The recipes can be **combined** in novel and complex ways using syntactic rules.
4. The recipes are cooked in an **open-innovation platform**.

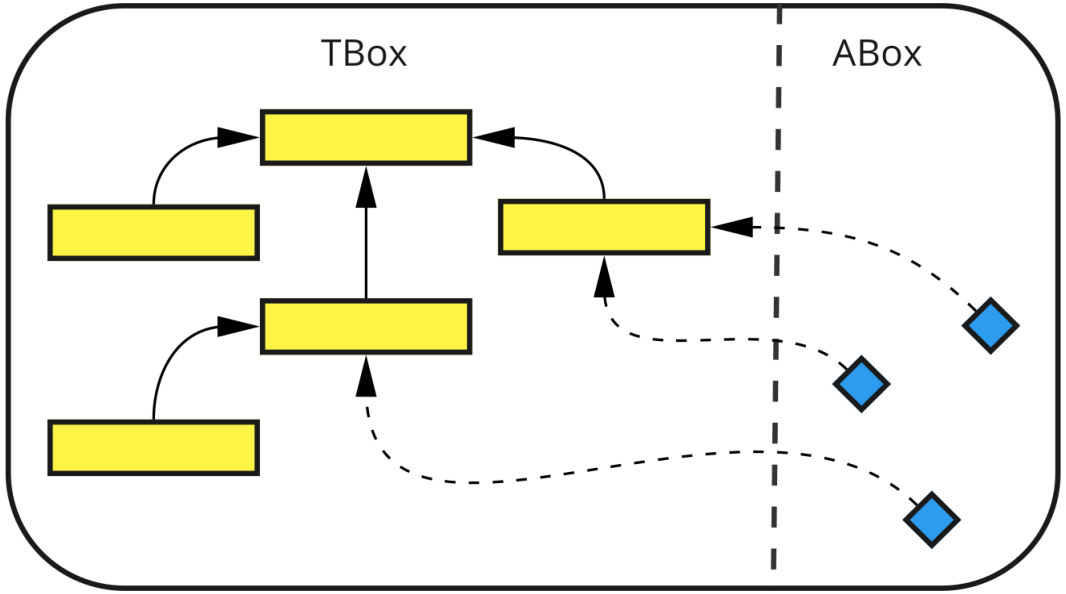


How is this done in practice?

Behind the scenes: the Knowledge Base

The Knowledge Base is composed of an ontology describing concepts and their relationships (aka, the TBox) and instances of these classes storing the information (aka, the ABox).

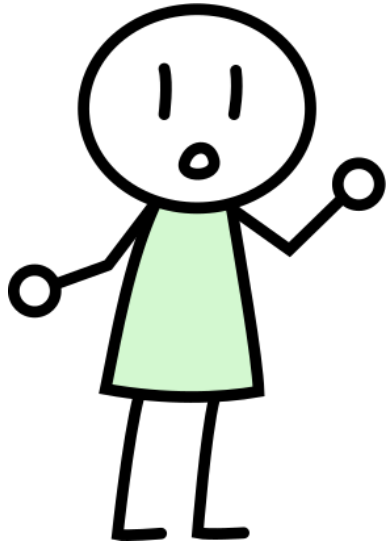
Triplestore Technology

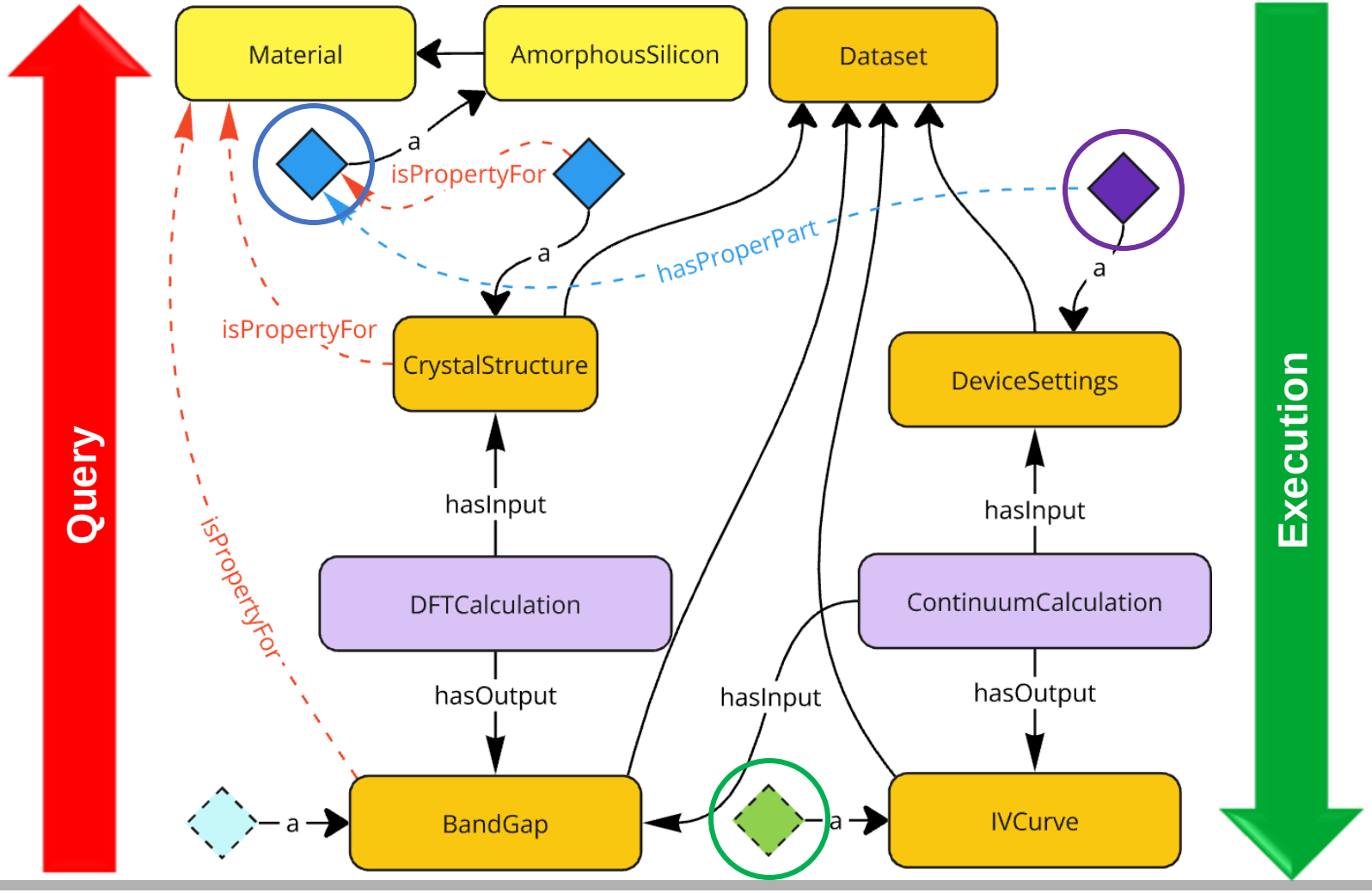


- How to write an ontology from scratch?
- What are the rules to follow in order to decribe a workflow?
- Which granularity is required? (spoiler: it depends on the functionality you want to achieve)

Semantic representation of a simple workflow

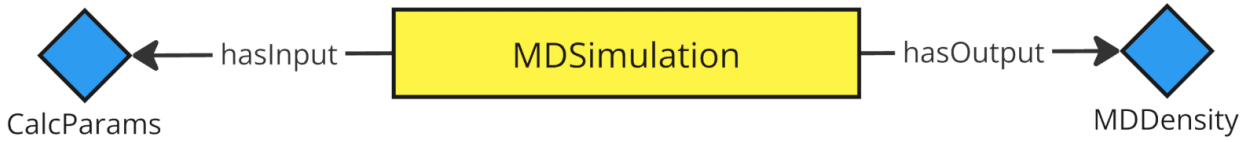
I want to simulate the [*physical property*] of a [*device*] made of [*material*]





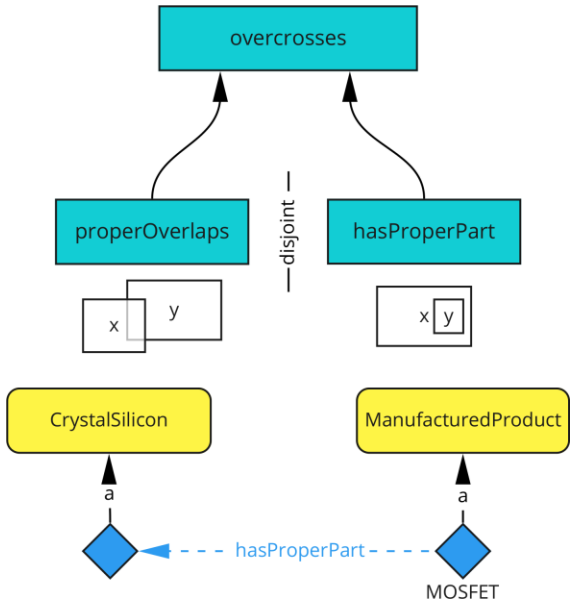
Semantic description of workflows

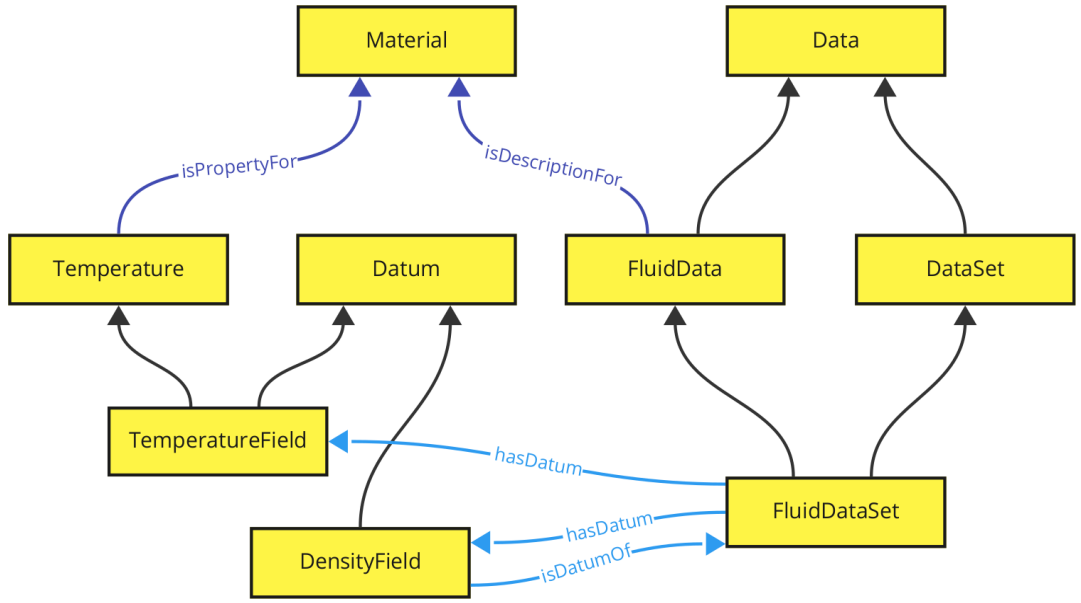
- The topology of a workflow is build using **hasInput** and **hasOutput** relations connecting processes to datasets.



- OntoFlow uses semantic relations to filter possible workflow routes.
- What is the meaning of these relations?

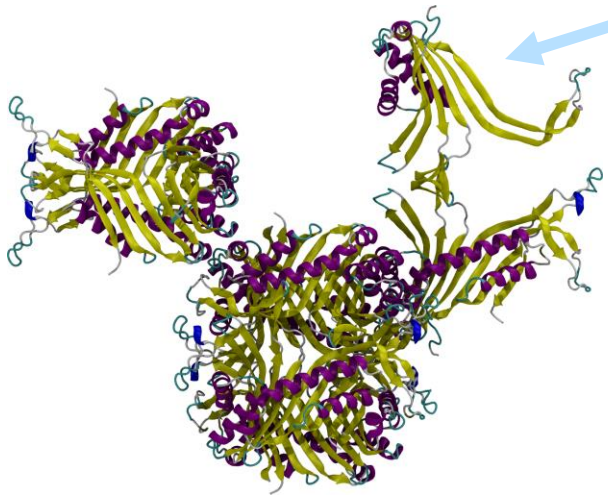
```
RELATIONS:
HasProperPart
IsDatumOf
HasDatum
IsPropertyFor
isDescriptionFor
```



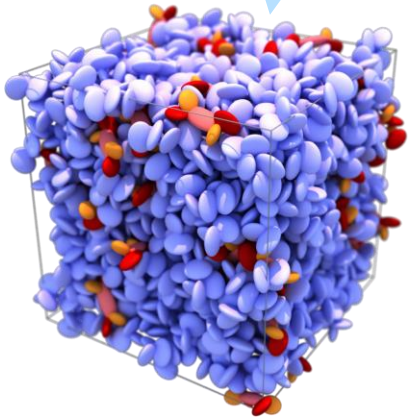


Breaking workflows into single tasks

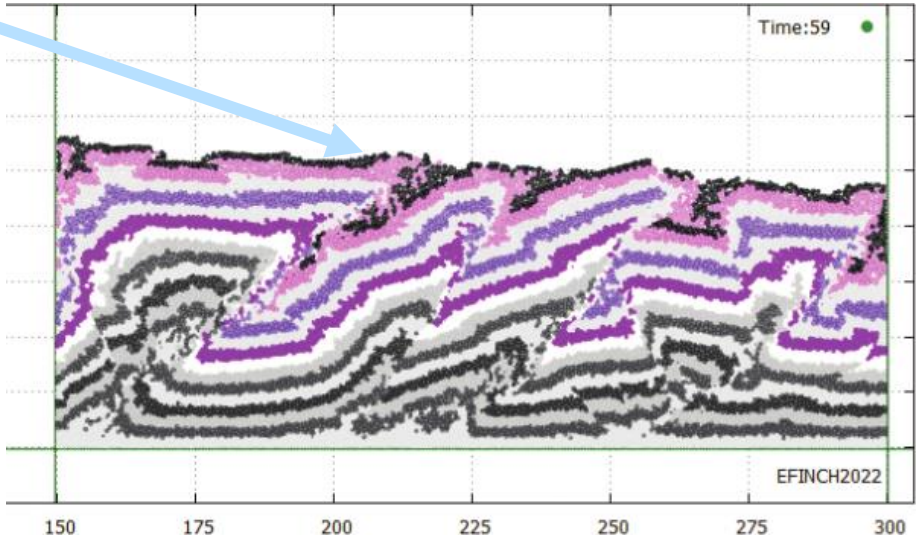
- Each workflow is broken down into a series of individual tasks.
- Each task is a computational **process** transforming input data into output data.
- Each process has a **limited scope** and is described with a **wrapper**.
- **Same** software, **different uses**: `lmp -i sample01_t02_02.in -l sample01_t02_02.log`



7PWN protein, 22K atoms

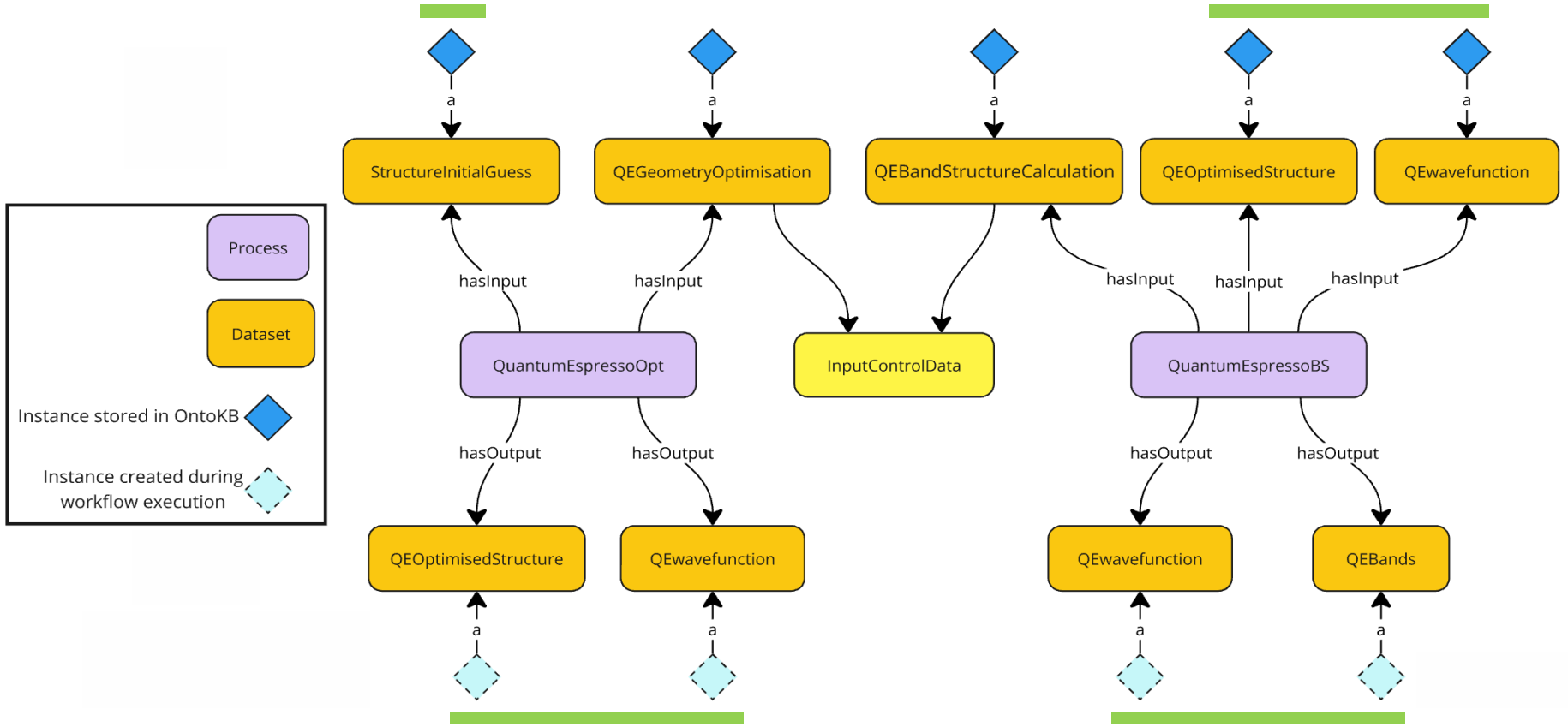


Coarse-grained model of organic semiconductors (MOLC model)



Structural geology simulations
Credit: Emma Finch, U. Manchester

Describing Quantum Espresso tasks



- Tasks are divided into different classes, that uses **hasInput** and **hasOutput** relations to generate different topologies.
- How to split the input data into different datasets?

Using *exec_wrapper* for CLI software

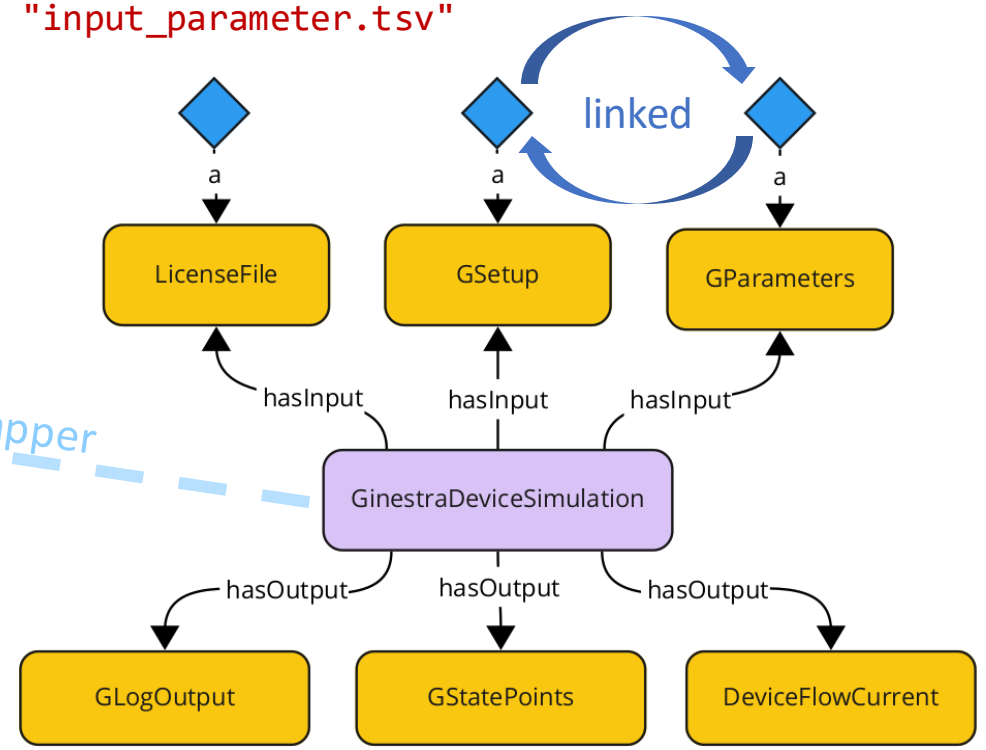
The I-V curve of a transistor where the insulator is set, and the semiconductor can be changed.

Suppose we want to describe a simulation that is executed with the command:

```
$ ginestra-core-sim.exe -l "ginestra.lic" -i "input.ini" -p "input_parameter.tsv"
```

```
- workflow: execflow.exec_wrapper
    inputs:
      shelljob:
       metadata:
         options:
           resources:
             num_machines: "{{ ctx.ginestra.m_n_machines }}"
             num_mpiprocs_per_machine: "{{ ctx.ginestra.m_n_mpiprocs_pm }}"
      command: "{{ ctx.ginestra.executable }}"
      arguments:
        - "-l"
        - "{{ ctx.ginestra.licence }}"
        - "-i"
        - "{{ ctx.ginestra.g_setup }}"
        - "-p"
        - "{parameters}"
      files:
        parameters:
          filename: "input_parameters.in"
          node: "{{ ctx.g_parameters }}"
      outputs:
        - test.log
    postprocess:
      - "{{ ctx.current.outputs['test_log']|to_ctx('g_output') }}"
```

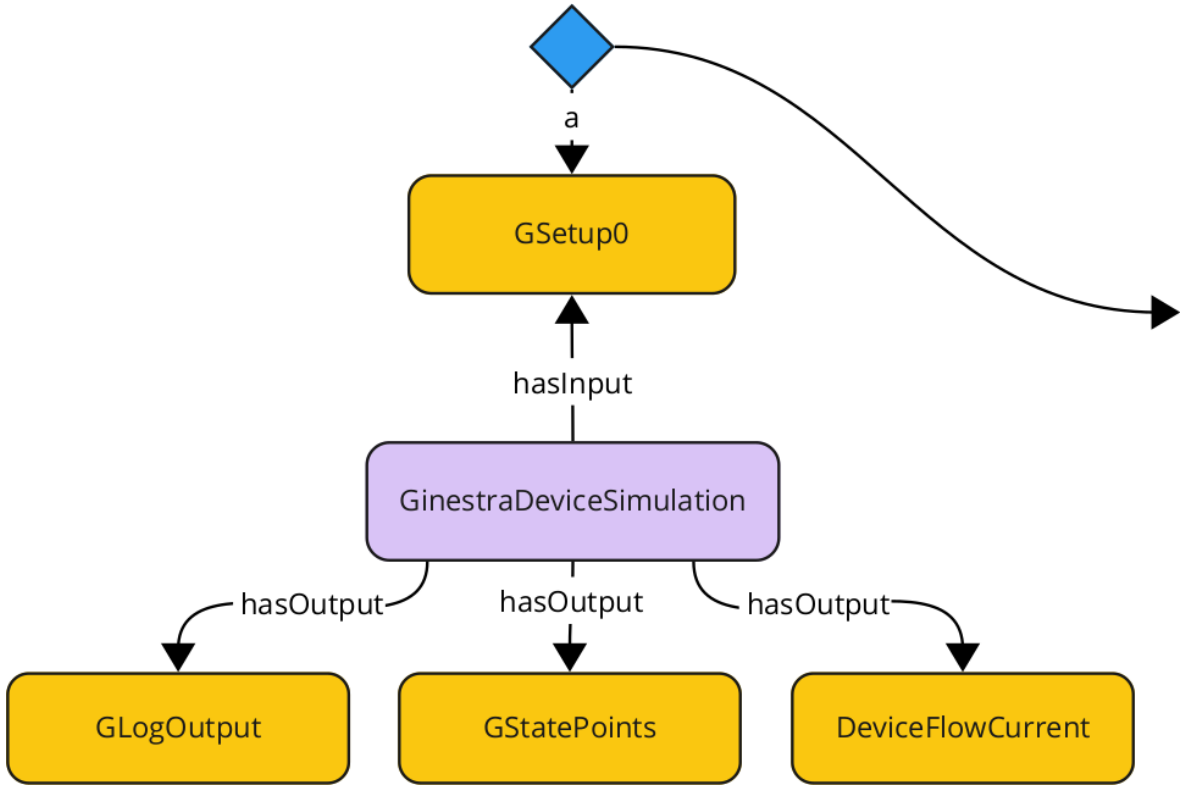
Two ways of passing files to the wrapper.



- Where are the semantics?

Conceptualization 1: dataset content hidden

If we are not interested in knowing the content of the input data, the files' location can be stored in a **single dataset**.

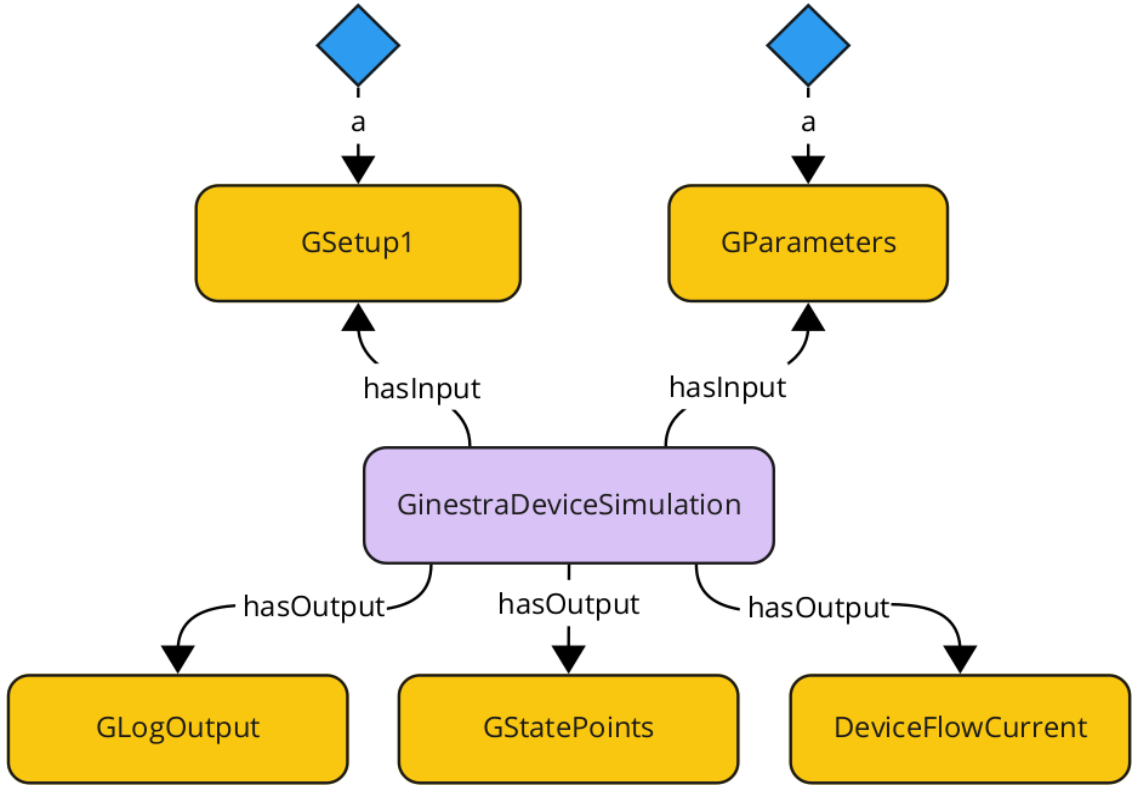


Input dataset

```
---
uFDE13MVbM:
  meta: http://ontotrans.eu/meta/0.1/InputGinestraSimulation
  dimensions: {}
  properties:
    executable: /tmp/ss1/dummy/ginestra-core-sim.exe
    licence: /tmp/ss1/Inputs/ginestra.lic
    g_input: /tmp/ss1/Inputs/input.ini
    g_parameters: /tmp/ss1/Inputs/input_parameter.tsv
    m_n_machines: 1
    m_n_mpiprocs_pm: 1
...
```

Conceptualization 2: expose user-controlled parameters

Here, the input data is split in **two datasets**. This allows to access the content of the input file `input_parameter.tsv`



Input datasets

```
---
egbRESXwrn:
  meta: http://ontotrans.eu/meta/0.1/InputGinestraSimulation
  dimensions: {}
  properties:
     executable: /tmp/ss1/dummy/ginestra-core-sim.exe
     licence: /tmp/ss1/Inputs/ginestra.lic
     g_input: /tmp/ss1/Inputs/input1.ini
     m_n_machines: 1
     m_n_mpiprocs_pm: 1
...

---
VHNl24cU4M:
  meta: http://ontotrans.eu/meta/0.1/GinestraParameters
  dimensions: {}
  properties:
     bandgap: 5.0
     effective_mass: 0.5
     temperature: 298.5
     grid_size: 100
...
```

Conceptualization 2: add the semantic layer with data mapping

Semantic relations are added through the data mapping:

Data mappings

```
mappings = [
  (GSET,                EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GSET.executable,     MAP.mapsTo,            SIMSOFT:Ginestra),
  (GSET.licence,        MAP.mapsTo,            SIMSOFT:LicenceFile),
  (GSET.g_input,        MAP.mapsTo,            SIMSOFT:TOMLFile),
  (GSET.g_input,        EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GSET.m_n_machines,   MAP.mapsTo,            SIMSOFT:MPIComputerNode),
  (GSET.m_n_mpiprocs_pm, MAP.mapsTo,           SIMSOFT:MPIProcsPerNode)
]
```

```
mappings = [
  (GPAR,                EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GPAR.bandgap,        MAP.mapsTo,            SS1:BandGap),
  (GPAR.bandgap,        EMMO.isPropertyFor,    SS1:VirtualMaterial),
  (GPAR.effective_mass, MAP.mapsTo,            EMMO:EffectiveMass),
  (GPAR.effective_mass, EMMO.isPropertyFor,    SS1:VirtualMaterial),
  (GPAR.temperature,    MAP.mapsTo,            EMMO.ThermodynamicTemperature),
  (GPAR.temperature,    EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GPAR.grid_size,      MAP.mapsTo,            MICROSTR.Grid)
]
```

Input datasets

```
---
egbRESXwrn:
  meta: http://ontotrans.eu/meta/0.1/InputGinestraSimulation
  dimensions: {}
  properties:
     executable: /tmp/ss1/dummy/ginestra-core-sim.exe
     licence: /tmp/ss1/Inputs/ginestra.lic
     g_input: /tmp/ss1/Inputs/input1.ini
     m_n_machines: 1
     m_n_mpiprocs_pm: 1
...
```

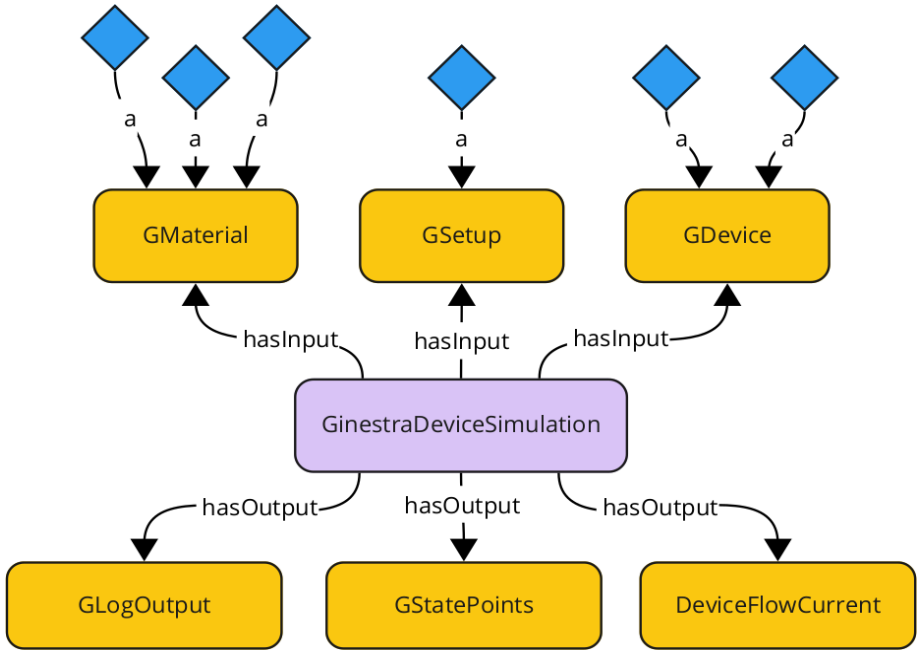
```
---
VHNl24cU4M:
  meta: http://ontotrans.eu/meta/0.1/GinestraParameters
  dimensions: {}
  properties:
     bandgap: 5.0
     effective_mass: 0.5
     temperature: 298.5
     grid_size: 100
...
```

- The user-defined parameters refer both to the device and the material it is composed of.
- To filter many datasets, we need to access the content of file `input1.ini`

Conceptualization 3

Here, the input data is **sorted** into a dataset referring to the active material, and one referring to the device.

```yaml
---
nThlFwkFpI:
  meta: http://ontotrans.eu/meta/0.1/GinestraMaterial
  dimensions: {}
  properties:
     bandgap: 5.0
     effective_mass: 0.5
...
```



Input datasets

```yaml
---
Tf0GCu7eH3:
  meta: http://ontotrans.eu/meta/0.1/InputGinestraSimulation
  dimensions: {}
  properties:
     executable: /tmp/ss1/dummy/ginestra-core-sim.exe
     licence: /tmp/ss1/Inputs/ginestra.lic
     m_n_machines: 1
     m_n_mpiprocs_pm: 1
     input_template: /tmp/ss1/Inputs/ginestra_input.template # generic
     state: /tmp/ss1/Inputs/state.hdf5
     out_folder: ./
     output: sim_device.hdf5
     grid_size: 100
...
---
4I4DsZ4Kjj:
  meta: http://ontotrans.eu/meta/0.1/GinestraDevice
  dimensions: {}
  properties:
     g_device: /tmp/ss1/Inputs/device1.xml
     g_test: /tmp/ss1/Inputs/test1.xml
     param_template: /tmp/ss1/Inputs/ginestra_input_parameters1.template # test1
     temperature: 298.5
...
```

From file `input1.ini`

Conceptualization 3

The semantic mapping now allows to create dataset classes for **materials** and **devices**, enabling classification and filtering of the information stored in the Knowledge Base.

Data mappings

```
mappings = [
  (GDEV,                EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GDEV,                EMMO.hasProperPart,    SS1.VirtualMaterial),
  (GDEV.g_device,       MAP.mapsTo,            SIMSOFT:XMLFile),
  (GDEV.g_device,       EMMO.isDescriptionFor, EMMO.ManufacturedProduct),
  (GDEV.g_test,         MAP.mapsTo,            SIMSOFT:XMLFile),
  (GDEV.param_template, MAP.mapsTo,            SIMSOFT:Jinja2File),
  (GDEV.temperature,    MAP.mapsTo,            EMMO.ThermodynamicTemperature),
  (GDEV.temperature,    EMMO.isDescriptionFor, EMMO.ManufacturedProduct)
]
```

Input datasets

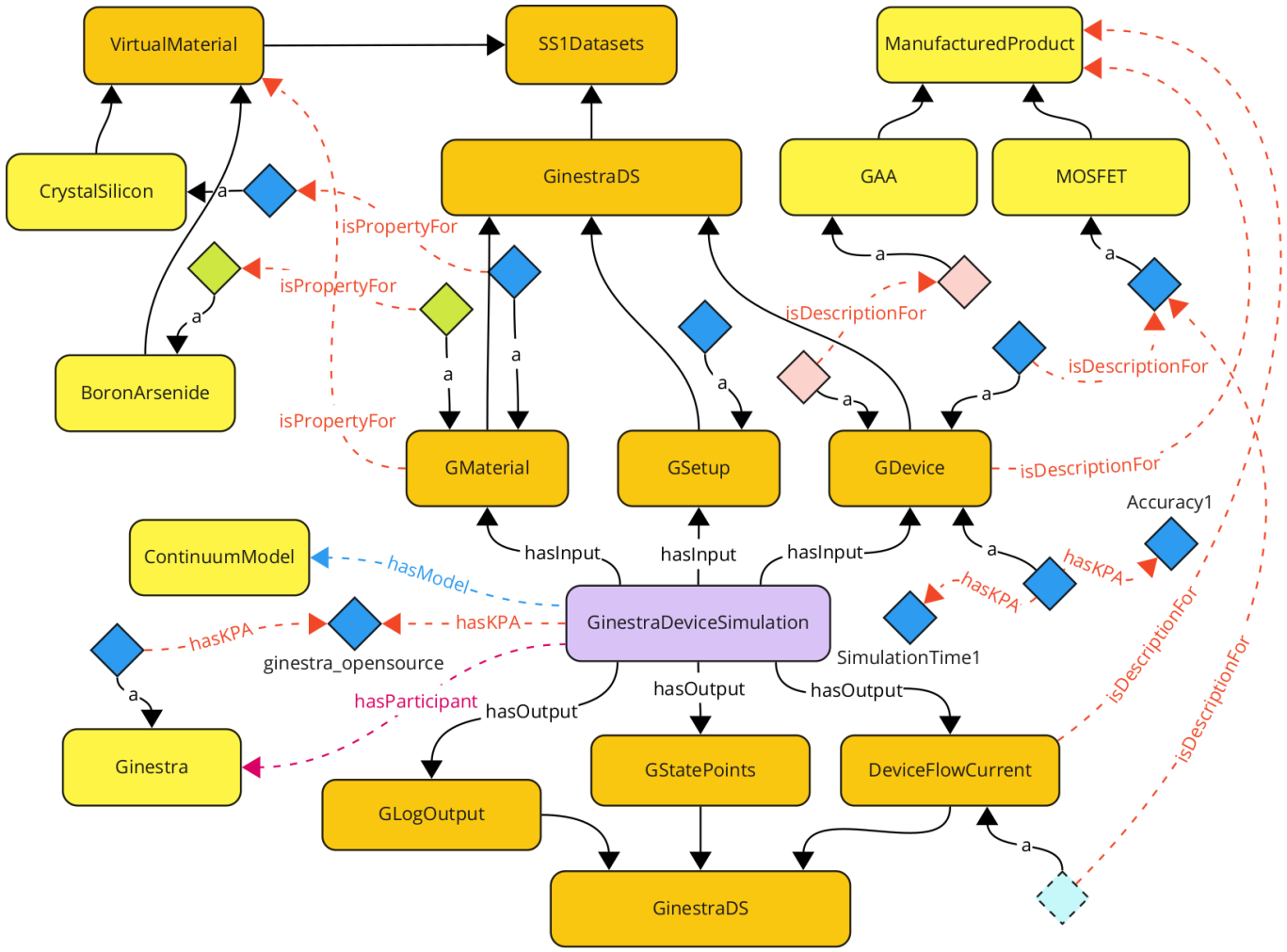
```
---
4I4DsZ4Kjj:
  meta: http://ontotrans.eu/meta/0.1/GinestraDevice
  dimensions: {}
  properties:
     g_device: /tmp/ss1/Inputs/device1.xml
     g_test: /tmp/ss1/Inputs/test1.xml
     param_template: /tmp/ss1/Inputs/ginestra_input_parameters1.template # test1
     temperature: 298.5
...
```

```
mappings = [
  (GMAT,               EMMO.isDescriptionFor, SS1.VirtualMaterial),
  (GMAT.bandgap,       MAP.mapsTo,            SS1:BandGap),
  (GMAT.bandgap,       EMMO.isPropertyFor,    SS1:VirtualMaterial),
  (GMAT.effective_mass, MAP.mapsTo,           EMMO:EffectiveMass),
  (GMAT.effective_mass, EMMO.isPropertyFor,   SS1:VirtualMaterial)
]
```

```
---
nThlFwkFpI:
  meta: http://ontotrans.eu/meta/0.1/GinestraMaterial
  dimensions: {}
  properties:
     bandgap: 5.0
     effective_mass: 0.5
...
```

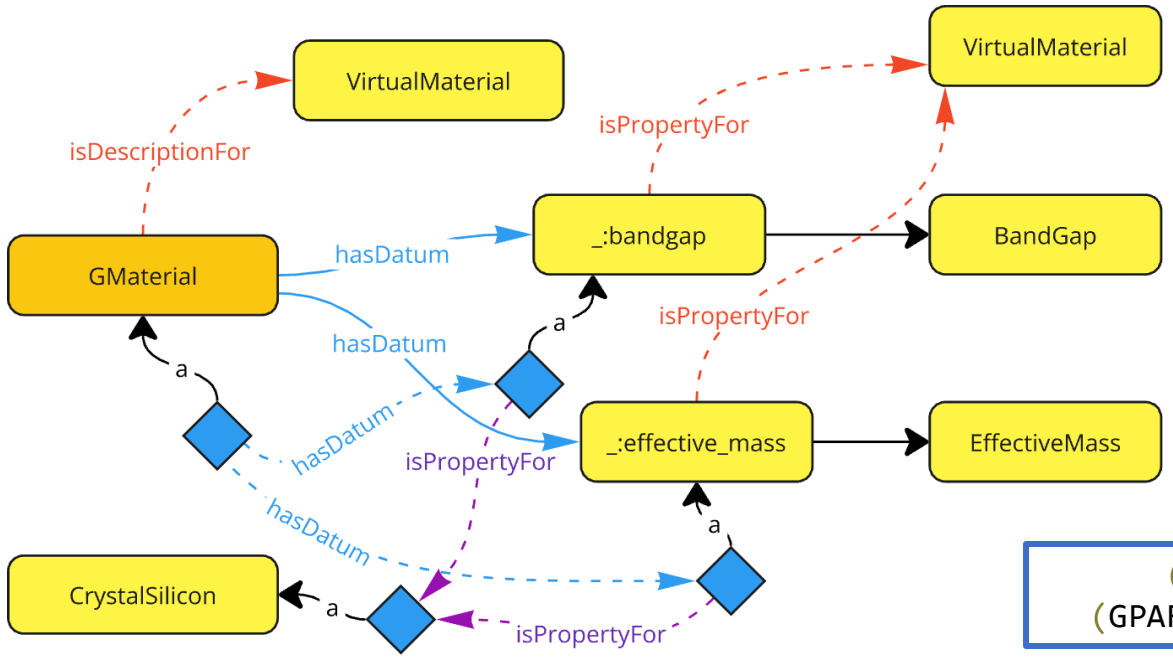
- OTEAPI pipelines to serialise the input datasets into the correct files.

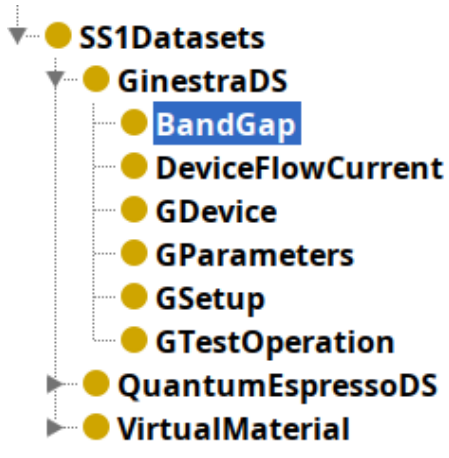
Conceptualization 3: the Knowledge Base

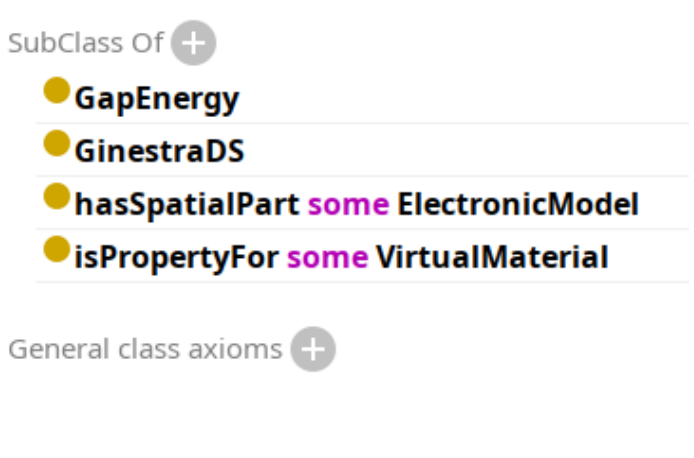


- It can handle arbitrary combinations of devices and semiconductors.
- It can be enriched with additional relations.
- Rules to propagate object property assertions:
  - if an axiom is shared between the input and output datasets, then the new instance of the output dataset inherits the assertion declared for the input dataset.
- It allows creating FAIR data.
- A new KPA for material datasets: **size**.

Serialising DLite datamodels as EMMO datasets





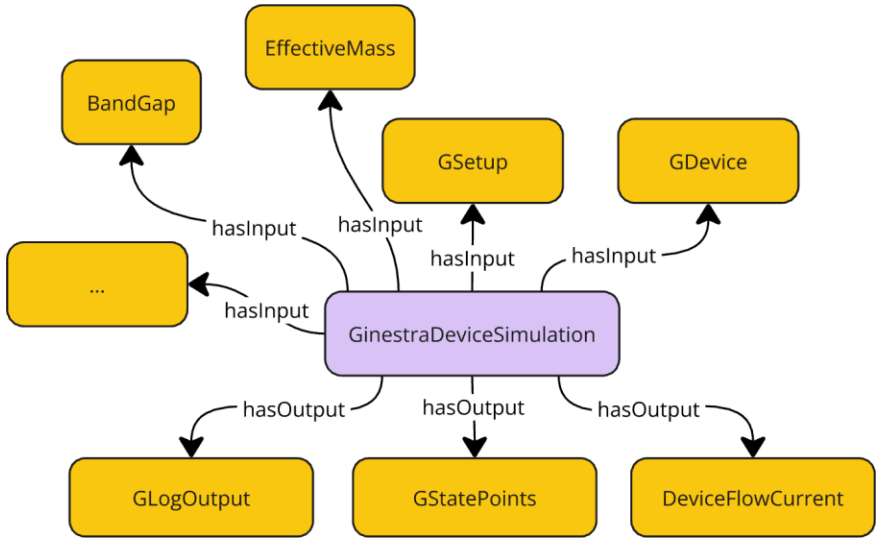



```
(GPAR.bandgap,      MAP.mapsTo,          SS1:BandGap),
(GPAR.bandgap,      EMMO.isPropertyFor,  SS1:VirtualMaterial),
```

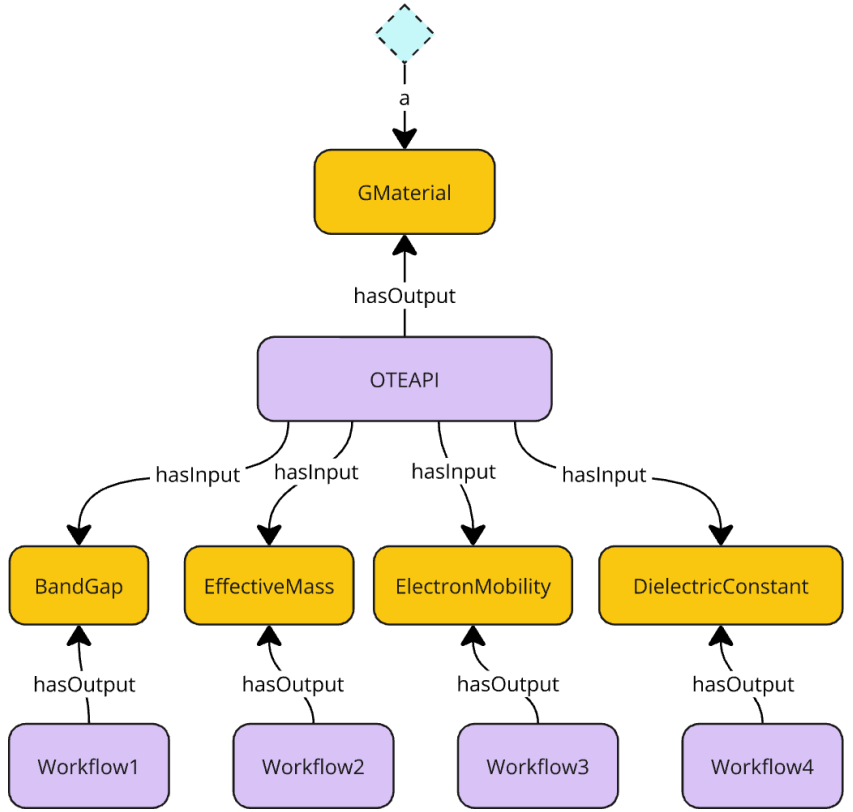
- Shall the Datum classes be defined entirely with mappings?
  - Then we don't need the mapsTo and the corresponding class in the TBox.
  - Blank nodes to be replaced with permanent classes.

Using OTEAPI pipelines to create plain datasets

Keep increasing the level of detail of the input dataset.



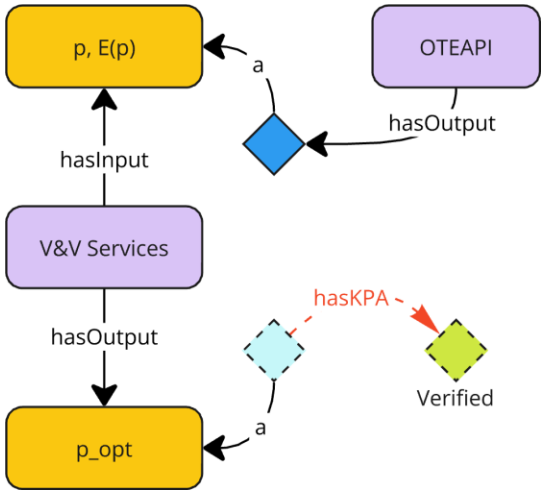
vs



- Gathers data from different workflows – through OntoFlow.
- Assemble a specific dataset with an OTEAPI pipeline.
- Should the pipeline be explicitly defined in the KB?

Discussion points

- To document workflow, begin by setting the modelling goals:
  - Set the level of detail for the input and output datasets.
  - Separate syntax from semantics.
  - Create the classes for processes and datasets,
  - … and then the wrappers and pipelines (OntoConv rules?).
- Dataset serialisation with pipelines: mapping, blank nodes, mapsTo.
- Use V&V services to populate the KB with KPA on accuracy and numerical stability.



Beware of semantics

actually, a composite

Thank you for your attention! ^__^

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