D2.5

Semantic interoperability of the automated workflows through SimPhoNy

Matthias Büschelberger, Kathrin Frei, Christian Eichheimer,
Joana Francisco Morgado, Francesco Benedetto, Claudio Rosati,
Matteo Bertocchi, and Arrigo Calzolari

Document information

Project acronym:	INTERSECT
Project full title:	Interoperable Material-to-Device simulation box for disruptive electronics
Research Action Project type:	Accelerating the uptake of materials modelling software (IA)
EC Grant agreement no.:	814487
Project starting / end date:	1st January 2019 (M1) / 31st January 2022 (M37)
Website:	www.intersect-project.eu
Final version:	27/07/2021
Deliverable No.:	D2.5
Responsible participant:	EPFL (participant number 2)
Contributing Consortium members:	FRA, AMAT, CNR
Due date of deliverable:	31/07/2021
Actual submission date:	29/07/2021
Dissemination level:	PU - Public

Authors: Matthias Büschelberger, Kathrin Frei, Christian Eichheimer, Joana Francisco Morgado Francesco Benedetto, Claudio Rosati, Matteo Bertocchi, and Arrigo Calzolari

To be cited as: M. Büschelberger, K. Frei, C. Eichheimer, J. Francisco Morgado, F. Benedetto, C. Rosati, M. Bertocchi, and A. Calzolari (2021): Semantic interoperability of the automated workflows through SimPhoNy. Deliverable D2.5 of the H2020 project INTERSECT (final version as of 27/07/2021). EC grant agreement no: 814487, EPFL, Lausanne, Switzerland.

Disclaimer:

Versioning and Contribution History

Version	Date	Modified by	Modification reason
v.01	01/07/2021	Matthias Büschelberger	First Version
v.02	27/07/2021	Arrigo Calzolari	Final Revision

Content

[1] Acronyms are marked in purple in the text and defined at the end of the document.

1. Executive Summary

This deliverable is a result of the activities carried out within Task 2.4 - *Interoperability hub: Upscaling of semantic interconnections* - and is dedicated to the implementation and integration of the European Materials Modeling Ontology (EMMO)-based semantic model (reported in D1.3) in the developed INTERSECT Interoperable Material-to-Device (IM2D) automated workflows and Graphical User Interface (GUI).

We developed a specific extension of the OSP-core (called OSP-wrapper – in this case AiiDA-wrapper) for SimPhoNy, which provides a flexible interface for the implemented ontologies. This interface controls the workflow process, facilitates the precision of the simulation results, and adjusts the inputs according to the different *persona,* as defined in D1.1. Furthermore, it enriches the computed data with materials scientific knowledge (i.e., metadata) stored into Resource Description Framework (RDF) triples, and it directs them towards the specific workflow routes. For the generation of RDF-triples, the third-party AllegroGraph triple store (https://allegrograph.com) has been added to IM2D. The combination of all these features enables the semantic interoperability upscale of IM2D.

In order to meet the Application Programming Interface (API) requirements of the new GUI architecture (D2.6), an additional REpresentational State Transfer (REST) service has been developed to instantiate the Common Universal Data Structure (CUDS) objects, and to control and execute the AiiDA-wrapper itself. On the technical ground, the need for an extra REST API is motivated by the fact that the GUI is written in Java and it requires an additional interface to generate the Python-native CUDS objects. In this way, the REST API provides a seamless integration of the IM2D-frontend into GINESTRA™ and a clear separation towards the remote computational services through AiiDA and SimPhoNy.

The work has been carried out in close collaboration with the project partners, through the joint participation in regular biweekly meetings and the set up of a shared development server.

2. Description of the work done

A lot of effort has been profused on further developing the EMMO INTERSECT extensions (D1.3), which drive the semantic interoperability across IM2D via CUDS. This deliverable reports on the architecture design and development of the AiiDAWrapper and SimPhoNy-REST API that exploit the EMMO compliant CUDS for the execution of the calculations in AiiDA and supply the crucial input information to/from the GUI (D2.6).

The need for the REST API interface on top of the semantic layer of SimPhoNy, due to the different native language of the GUI (i.e., Java), represented a critical challenge, which required an alternative interoperable solution. This technical issue has been solved by adopting a http-service that is targeted to query the ontology, to instantiate the individuals of the ontology classes as CUDS-objects, and finally to run the AiiDA-workflows according to the input specified by the CUDS.

Given the high dependency of the developed interfaces (wrapper and REST API) on the semantic model, the implementation has been carried out in parallel and adjusted to the new requirements defined at the ontology (formalized in the Web Ontology Language - OWL) and/or interface levels (Python). The parallel development has been made possible through the set up of a shared server hosted by EPFL. The source code of the developed wrapper is hosted on the Fraunhofer GitLab (https://gitlab.cc-asp.fraunhofer.de/simphony/wrappers/aiida-wrapper).

3. Deviation from planned work in the DoA

Originally, this deliverable was due in January 2021. However, due to delays in the ontology development (see D1.3), the submission date was postponed to 31/07/2021. The need to develop a new REST API from scratch (not foreseen in the original plan) also contributed to the delay of this Task.

Moreover, the ongoing developments of the INTERSECT AiiDA workflows specifications (Tasks 2.3 and 2.4) necessary for the IM2D requirement identification (e.g., in terms of the available "materials properties on demand") also contributed to the delay of the ontology and AiiDA-SimPhoNy wrapper development, which propagated to the release of D2.5.

At the time of preparation of this document (M31), the delay has been recovered and the activity of Task 2.4 is back in line with the Description of the Action (DoA). This has been possible also thanks to the reorganization of the development team at Fraunhofer IWM that allowed for a more efficient re-distributions of the tasks and the control of the implementation progress with respect to the project deadline.

4. Results

The architecture design of the IM2D interoperability layer is based on (i) the GUI developed by AMAT (see D2.6-T2.5), (ii) the open semantic simulation framework provided by SimPhoNy, and (iii) the AiiDA-based automated workflows (Task 2.3), as shown in Figure 1.

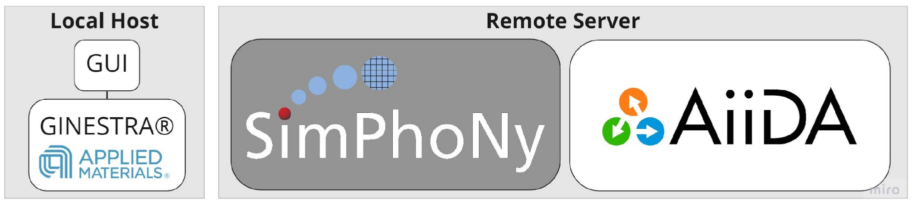


Figure 1: Architecture of the IM2D toolbox.

The user can contact the remote server with SimPhoNy and AiiDA through the GUI provided by GINESTRA™, as briefly demonstrated in Section 5. The GUI is designed to automatically send http-requests to the SimPhoNy-REST API for the required content and to dynamically adjust the displayed components on the basis of the data returned by the http-response. The http-queries are sent while the user is exploring the materials browser of IM2D. According to DoA, SimPhoNy performs the semantic upscaling of the AiiDA-workflows, so the user does not need to take care of the software-related specifications and syntax. This goes in the direction of simplifying the complexity of the computational problem for industrial stakeholders. To fulfil this task, SimPhoNy configures and forwards several http-queries while communicating with AiiDA. A detailed explanation of the dataflow and related components is provided in the following subsections. It should be highlighted that the REST APIs of AiiDA and SimPhoNy need to run on the same host machine, same programming environment, but on two different ports.

The requirements from various types of users, with different expertise levels, are realized through the control of the technical input parameters that affect the accuracy of the results (e.g., cutoff of the kinetic energy, number of K-points, convergence thresholds, etc.). For example, an advanced user might be more aware and interested in certain Key Performance Indicators (KPIs, such as the occupation method and spin polarization type) than a user with intermediate expertise. This simple example indicates that requirements provided at the GUI by the user can be related to both the material and the simulation specifications.

Since these KPIs are normally connected to the syntactic input parameters of the simulation backend, an ontological representation of these user expertise parameters has been specifically developed and described in D1.3. Due to this connection, the AiiDA-wrapper reacts to code-representative CUDS as well as to generic material-representative CUDS.

4.1 GET-method for *persona* level input

The procedure for the GET-method described in the following is schematically presented in Figure 2. The first task that SimPhoNy has to handle is the supply of the user levels. The user connects to the GUI and enters the name of a material property of interest and their knowledge level.

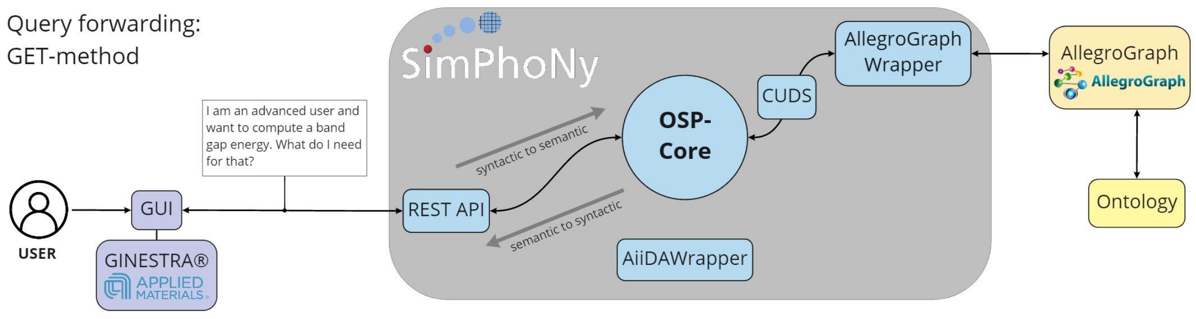
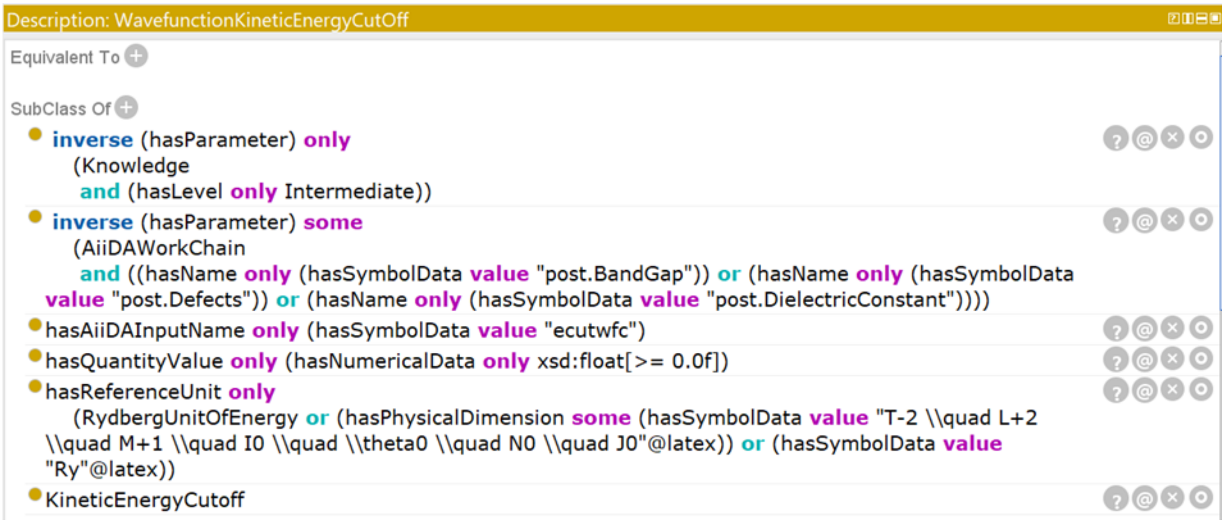


Figure 2: Query forwarding by GET-method: In the GUI, the user selects the expertise level and the material property of interest, e.g., the band gap energy. The GUI sends a query to SimPhoNy to get a list of properties that are required as input parameters for the calculation of the demanded material property. The REST-API triggers the OSP-core to filter the required parameters from the ontology (saved in AllegroGraph as triples). CUDS objects are generated from the ontology, carrying the semantic information of the query parameters. On the way back to the GUI, the semantic data is converted to syntactic data and the user receives a set of the required parameters, for which the user can enter any values of interest.

Both conditions are embedded into an http-route in the following format:

```
localhost:8000/api/v4/intersect/properties/<my-property-of-interest>/inputs/<my-user-level>.
```

When this route is taken via a GET-method in the SimPhoNy-REST service, the OSP-core sends a SPARQL-query to the AllegroGraph triple store, where the ontology-based triples are located. The SPARQL-query is dynamically adjusted with respect to the specified *persona* profile and AiiDA-workflow, and it gets as a response to the query a set of recommended parameters along with information about the data range-restrictions, data type-restrictions, unit-descriptions, unit-expressions, and additional user advices, such as further reading references. These are then transferred back as http-response from the SimPhoNy-REST API to the GUI (see Figure 4). Some of these strings relative to the units and physical dimensions are given in latex-syntax so to be displayed by the GUI. Figure 3 provides an example and shows how the ontology class is restricted to certain user expertises, workflow availability, AiiDA-input name, and the corresponding reference unit.

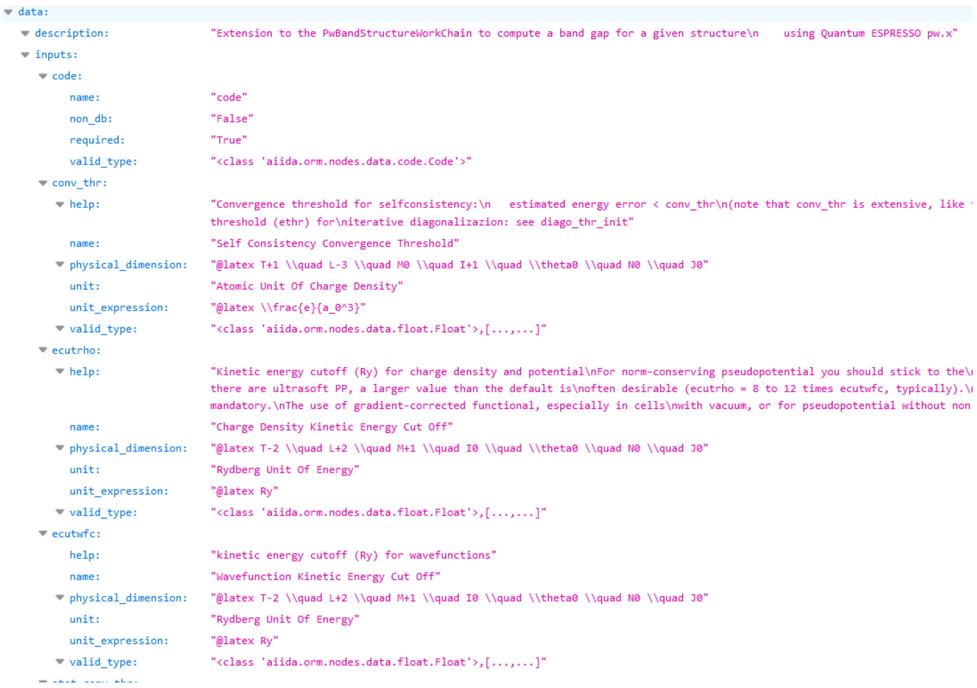


```
Description: WavefunctionKineticEnergyCutOff

Equivalent To

SubClass Of
  inverse (hasParameter) only
     (Knowledge
      and (hasLevel only Intermediate))
  inverse (hasParameter) some
     (AiiDAWorkChain
      and ((hasName only (hasSymbolData value "post.BandGap")) or (hasName only (hasSymbolData
 value "post.Defects")) or (hasName only (hasSymbolData value "post.DielectricConstant"))))
 hasAiiDAInputName only (hasSymbolData value "ecutwfc")
 hasQuantityValue only (hasNumericalData only xsd:float[>= 0.0f])
 hasReferenceUnit only
     (RydbergUnitOfEnergy or (hasPhysicalDimension some (hasSymbolData value "T-2 \\quad L+2
 \\quad M+1 \\quad I0 \\quad \\theta0 \\quad N0 \\quad J0"@latex)) or (hasSymbolData value
 "Ry"@latex))
 KineticEnergyCutoff
```

Figure 3: Screenshot from Protégé of the kinetic energy cutoff for the wavefunction and the according class restriction expressed through OWL DL. Note the latex notation of the physical dimension.



```
data:
  description:               "Extension to the PwBandStructureWorkChain to compute a band gap for a given structure\n    using Quantum ESPRESSO pw.x"
  inputs:
    code:
      name:                  "code"
      non_db:                "False"
      required:              "True"
      valid_type:            "<class 'aiida.orm.nodes.data.code.Code'>"
    conv_thr:
      help:                  "Convergence threshold for selfconsistency:\n   estimated energy error < conv_thr\n(note that conv_thr is extensive, like
                             threshold (ethr) for\niterative diagonalizazion: see diago_thr_init"
      name:                  "Self Consistency Convergence Threshold"
      physical_dimension:    "@latex T+1 \\quad L-3 \\quad M0 \\quad I+1 \\quad \\theta0 \\quad N0 \\quad J0"
      unit:                  "Atomic Unit Of Charge Density"
      unit_expression:       "@latex \\frac{e}{a_0^3}"
      valid_type:            "<class 'aiida.orm.nodes.data.float.Float'>,[...,...]"
    ecutrho:
      help:                  "Kinetic energy cutoff (Ry) for charge density and potential\nFor norm-conserving pseudopotential you should stick to the\
                             there are ultrasoft PP, a larger value than the default is\noften desirable (ecutrho = 8 to 12 times ecutwfc, typically).\
                             mandatory.\nThe use of gradient-corrected functional, especially in cells\nwith vacuum, or for pseudopotential without non
      name:                  "Charge Density Kinetic Energy Cut Off"
      physical_dimension:    "@latex T-2 \\quad L+2 \\quad M+1 \\quad I0 \\quad \\theta0 \\quad N0 \\quad J0"
      unit:                  "Rydberg Unit Of Energy"
      unit_expression:       "@latex Ry"
      valid_type:            "<class 'aiida.orm.nodes.data.float.Float'>,[...,...]"
    ecutwfc:
      help:                  "kinetic energy cutoff (Ry) for wavefunctions"
      name:                  "Wavefunction Kinetic Energy Cut Off"
      physical_dimension:    "@latex T-2 \\quad L+2 \\quad M+1 \\quad I0 \\quad \\theta0 \\quad N0 \\quad J0"
      unit:                  "Rydberg Unit Of Energy"
      unit_expression:       "@latex Ry"
      valid_type:            "<class 'aiida.orm.nodes.data.float.Float'>,[...,...]"
```

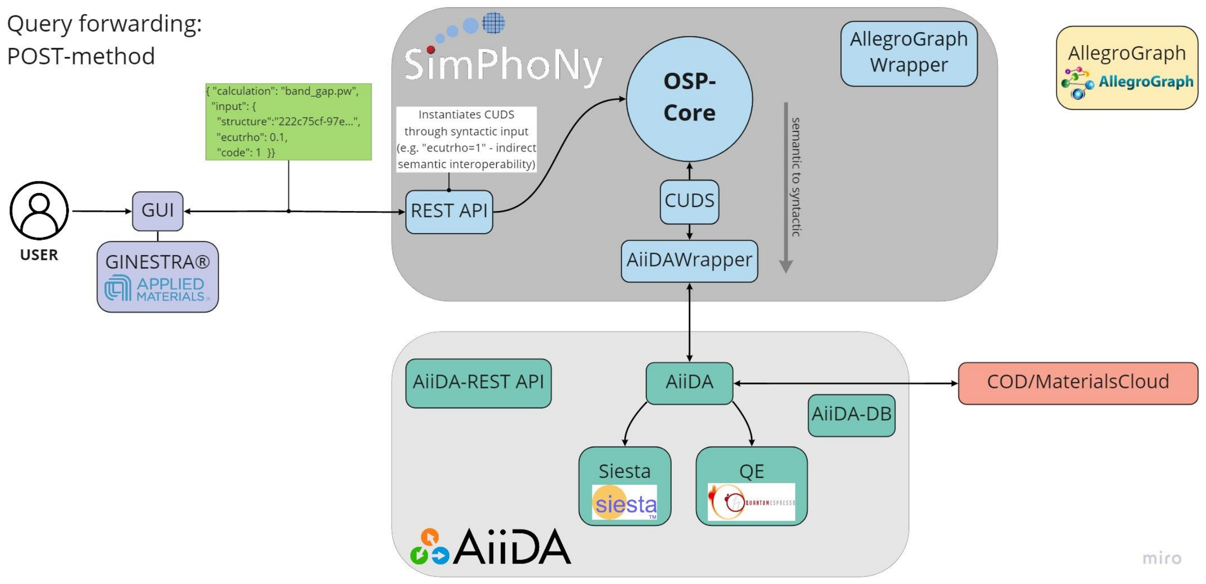
Figure 4: Example response to a query for http://localhost:8000/api/v4/intersect/properties/band_gap.pw/inputs/intermediate with respective parameters like the self-consistency threshold for the simulation convergence, cut off energy for the charge density, etc.

4.2 POST-method for submission of workflows through the SimPhoNy-wrapper

Once the user has chosen a set of parameters that they can set through the GUI, the workflow settings are posted via http to the SimPhoNy-REST server (POST-method, see Figure 5) by the following route:

```
localhost:8000/api/v4/intersect/submit
```

This json-formatted setup contains the name of the physical property to be computed, the Universally Unique IDentifier (UUID) of the crystal structure (standing for a structure from a Crystallographic Information File CIF-file, from Crystallography Open Database - COD - or self-uploaded) and, depending on the *persona* profile, additional input values, as shown in Figure 3.



```
{ "calculation": "band_gap.pw",
 "input": {
  "structure":"222c75cf-97e...",
  "ecutrho": 0.1,
  "code": 1  }}
```

Figure 5: Query forwarding by POST-method: As the user has entered the inputs for the required parameters and the UUID of the crystal structure of interest, the GUI sends a json file (green) with the submitted data to the SimPhoNy REST API, which triggers the OSP-core to give semantic data (CUDS objects) to the AiiDAWrapper (where the semantic data is transformed into syntactic). The AiiDAWrapper forwards the data to AiiDA, where the computation is conducted with Siesta or Quantum ESPRESSO.

The SimPhoNy-REST service instantiates the CUDS-objects, which represent the material-scientific characterization through ontological entities and their relationships. These might be, e.g., the formula and CIF-UUID of the host structure, the crystal system, the material property (e.g., band gap, dielectric constant, etc.), the associated units, defect types, defect positions, etc. A detailed description of entities represented by the CUDS objects can be found in D1.3.

The communication between the wrapper and AiiDA is built up by a three layer design, which consists of:

- **Semantic layer**: it represents the imported ontology by means of a syntactic language (CUDS classes and objects);

- **Syntactic layer**: it is in direct communication with the third-party tools. It provides a simple interface and interacts with the wrapped application, which in turn communicates with the interoperability layer;
- **Interoperability layer**: it provides the connection and translation between the semantic and syntactic parts. It also contains the storage of all the objects belonging to a session.

The interlinked CUDS are passed to a `Wrapper Session Class` of SimPhoNy within the Python-Integrated Development Environment (IDE), which reacts and executes the AiiDA-workflows according to the ontology-class to which the CUDS-object belongs. For example, if an ontology-class of type `emmo.BandGap` is received by the wrapper, the corresponding `PWBandGapWorkChain` of AiiDA is executed with an interlinked CIF-Structure UUID. The main implementation is done; nonetheless in the remaining months of the project we will work to its refinement. For example, a feature will be added, so that only a CUDS-object with the chemical formula needs to be passed, while the corresponding CIF-file will be automatically downloaded from the COD through the `CODImportWorkChain` of AiiDA.

Additionally, the low-level and code-specific CUDS from a certain user profile are known to the wrapper, so that software-specific parameters for the model accuracy can be passed and processed too. However, these individuals are not mandatory and are closer to the more generic material-scientific entities, thus the recommended values from the AiiDA-backend are used.

Once the wrapper has launched the AiiDA-workchain, the process is constantly monitored until the computation is finished. The monitoring can also be directly done through the GUI by taking the route:

```
localhost:8000/api/v4/intersect/status.
```

This route directly proxies the AiiDA-REST API on the server machine, which exposes the process state.

The wrapper monitors the process as well. Once a material computation has finished, the numerical values are extracted from the output and added to the already instantiated CUDS-objects (crystal, crystal formula, crystal system, etc.), in order to complete the material characterization.

The overall cluster of all CUDS-objects from a simulation session is committed to a separate database within the AllegroGraph triple store for the long-time-storage in the form of their original RDF-triples, while the buffer of the `WrapperSessionClass` is resetted. This process is schematically shown in Figure 6.

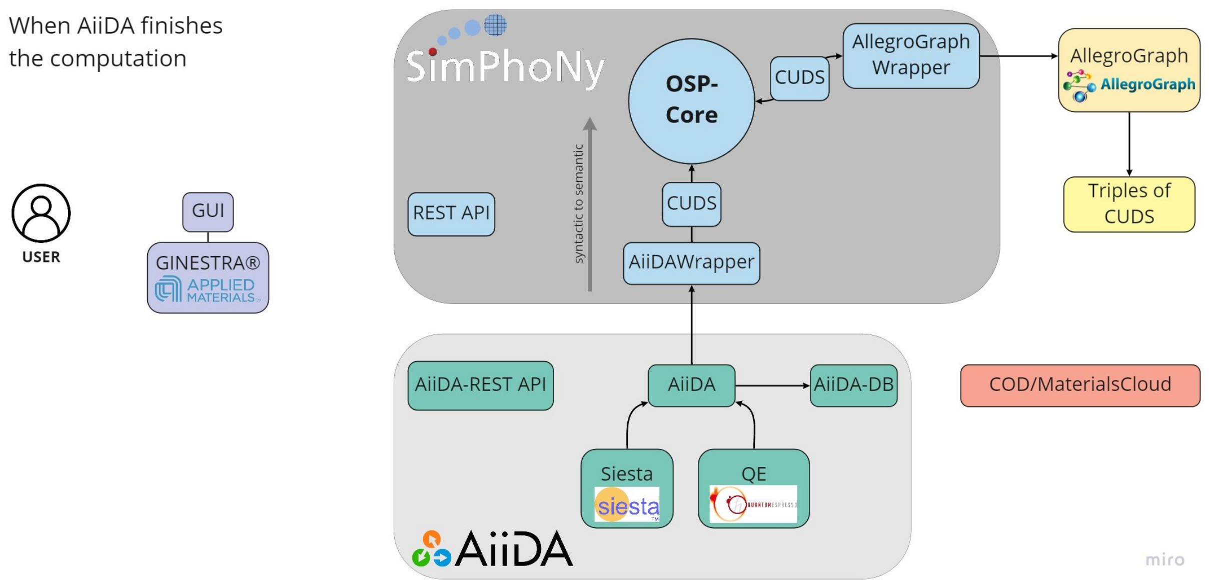


Figure 6: As soon as the computations in AiiDA are finished, the results are parsed into additional CUDS-objects through SimPhoNy and the underlying triples are pushed to the AllegroGraph triple store. Furthermore, the syntactic results and simulation protocols are stored in the AiiDA-DB (database).

Since the wrapper is an independently running Python-class, the execution of workflows through CUDS can also be handled without the SimPhoNy-REST API. In this case, it requires the instantiation and the interlinking of the CUDS-objects through an additional API like, e.g., the MarketPlace platform.

4.3 GET-methods for fetching the computation results

For the time being, the SimPhoNy-REST API forwards the requests for calculated properties directly to the AiiDA-REST API. However, the GUI needs to provide a set of http-routes in order to query, find and link all the syntactic information from the AiiDA-backend according to their individual nodes (holding a UUID).

The next weeks and months, we plan to extend the functionality of the SimPhoNy REST API by providing a route that queries all the CUDS-objects of the already derived properties from AllegroGraph in one single request. This shall be realized via passing a SPARQL-query or simply providing the human-readable name of the ontology class (e.g., “band gap”, “formation energy”), as shown in Figure 7. The output also covers all the linked CUDS objects like, e.g., for the crystal system, formula, etc.

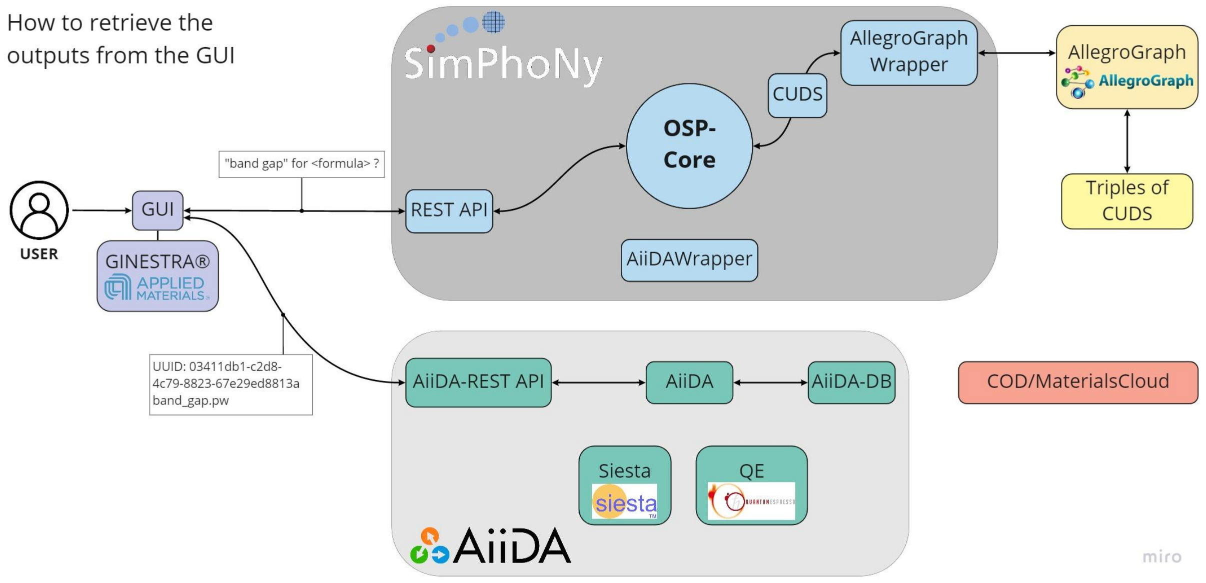


Figure 7: Schema of the two possible options on how the GUI retrieves the outputs: a) Direct querying of AiiDA-nodes from the AiiDA-DB through the AiiDA REST-API. b) Querying of human-readable keywords or SPARQL-queries to the Allgrograph through SimPhoNy and its REST API (work in progress).

More extensive information, such as large band diagrams or the exact protocols from the simulation, are not planned to be committed to AllegroGraph. Moreover, the respective CUDS-objects shall reference the exact origin, by connecting the computation node to the http-route of the AiiDA REST API, where the information can be found.

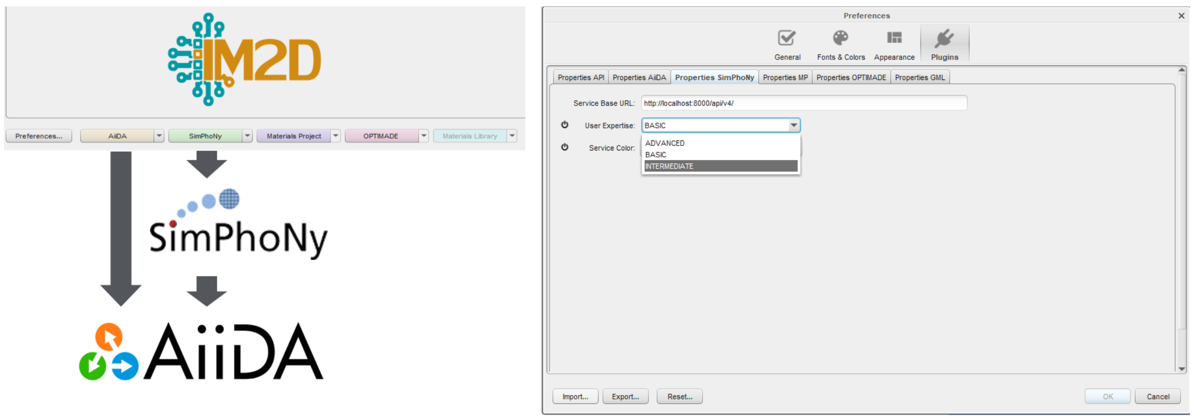
The AllegroGraph triple store backend realizes a semantic upscaling of the syntactic information model of AiiDA and also provides a user-friendly backend interconnection to the material scientific knowledge through independent interoperable platforms, like the MarketPlace project.

Triple stores play a significant role as a knowledge source in the machine learning models, e.g., when prediction models are fitted to experimental data.

The current implementations realize additional GET-routes from the AiiDA-REST API through SimPhoNy. This facilitates the full backend-connection to AiiDA, since SimPhoNy forwards all routes through an http-proxy realized by the Flask-package, if they are not explicitly defined in the SimPhoNy API. This has the advantage that the host and port of the user interface do not need to be switched within the workflow.

5. Interim GUI

As described in detail in D2.6, the GINESTRA-AiiDA plugin has been conceived to connect SimPhoNy as well. The goal is to finalize the interface and use it as the final IM2D GUI.

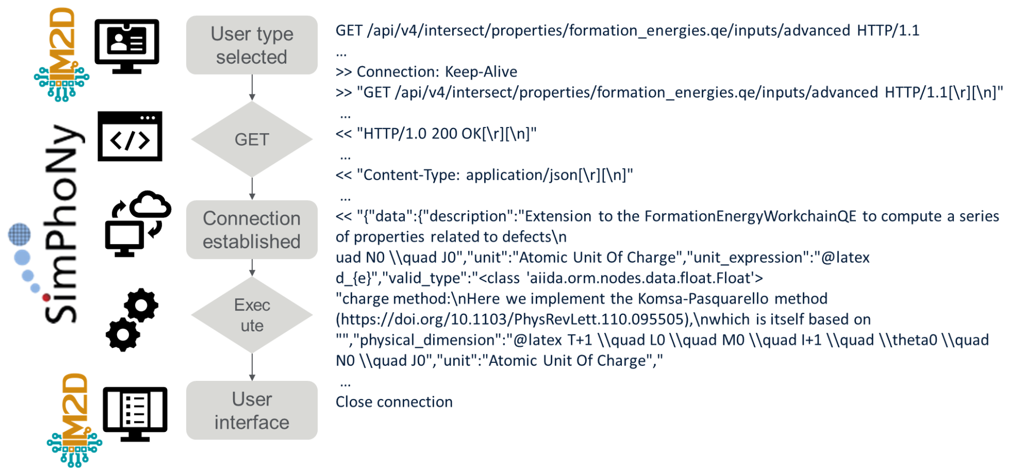


Service Base URL: http://localhost:8000/api/v4/
User Expertise: BASIC
Service Color:

Figure 8: Left: Interface of the connection framework. Right: SimPhoNy preference panel interface.

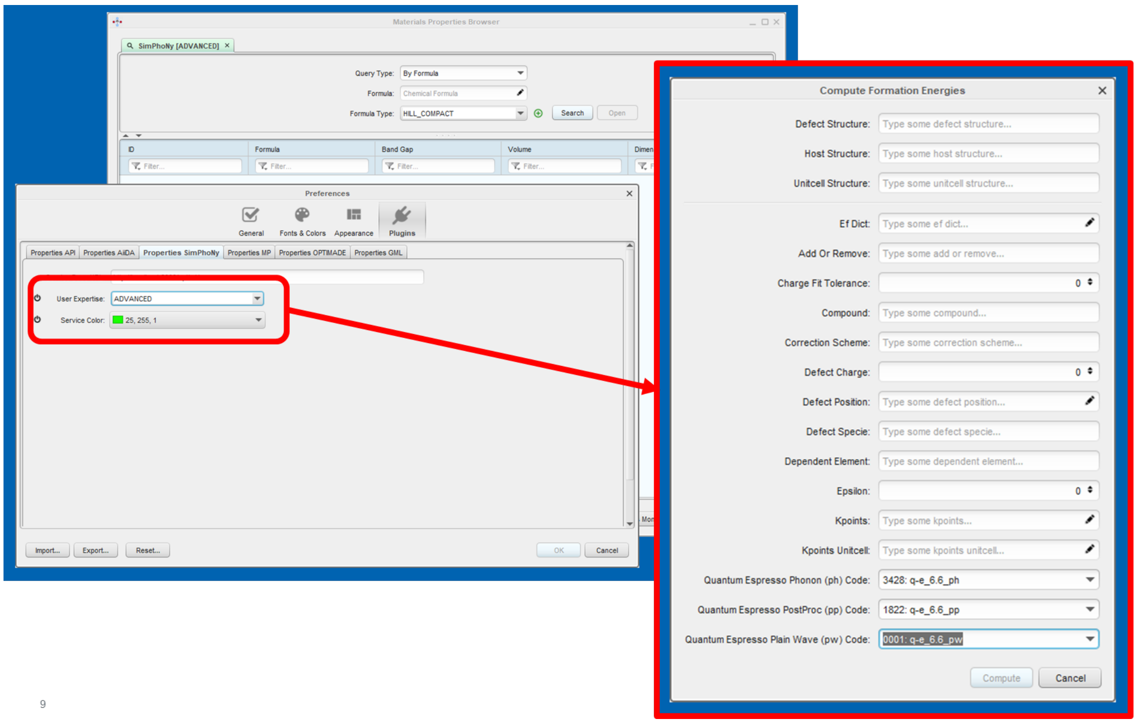
As shown in Figure 8, the interface is now optimized for selecting the SimPhoNy user profile in the preference panel. In the plugin section of the preference panel (Figure 8, right) it is possible to set the user's level of experience (*User Expertise*: Basic, Intermediate or Advanced) by selecting the SimPhoNy tab.

Once the user expertise level has been selected, the interface connects to SimPhoNy and retrieves the configured user type interface. By using this interface, the user can retrieve data from AiiDA according to the selected knowledge level, as shown in Figure 9 and Figure 10.



```
GET /api/v4/intersect/properties/formation_energies.qe/inputs/advanced  HTTP/1.1
…
>> Connection: Keep-Alive
>> "GET /api/v4/intersect/properties/formation_energies.qe/inputs/advanced  HTTP/1.1[\r][\n]"
 …
<< "HTTP/1.0 200 OK[\r][\n]"
 …
<< "Content-Type: application/json[\r][\n]"
 …
<< "{"data":{"description":"Extension  to the FormationEnergyWorkchainQE  to compute a series
of properties  related to defects\n
uad N0 \\quad J0","unit":"Atomic Unit Of Charge","unit_expression":"@latex
d_{e}","valid_type":"<class  'aiida.orm.nodes.data.float.Float'>
"charge method:\nHere  we implement  the Komsa-Pasquarello  method
(https://doi.org/10.1103/PhysRevLett.110.095505),\nwhich  is itself based on
"","physical_dimension":"@latex T+1 \\quad L0 \\quad M0 \\quad I+1 \\quad \\theta0 \\quad
N0 \\quad J0","unit":"Atomic  Unit Of Charge","
 …
Close connection
```

Figure 9: SimPhoNy connection framework for an advanced user example.


Materials Properties Browser

SimPhoNy [ADVANCED]

Query Type: By Formula
Formula: Chemical Formula
Formula Type: HILL_COMPACT

ID	Formula	Band Gap	Volume	Dimen
Filter...	Filter...	Filter...	Filter...	F

Preferences

User Expertise: ADVANCED
Service Color: 🟩 25, 255, 1

Compute Formation Energies

Defect Structure: Type some defect structure...
Host Structure: Type some host structure...
Unitcell Structure: Type some unitcell structure...
Ef Dict: Type some ef dict...
Add Or Remove: Type some add or remove...
Charge Fit Tolerance: 0
Compound: Type some compound...
Correction Scheme: Type some correction scheme...
Defect Charge: 0
Defect Position: Type some defect position...
Defect Specie: Type some defect specie...
Dependent Element: Type some dependent element...
Epsilon: 0
Kpoints: Type some kpoints...
Kpoints Unitcell: Type some kpoints unitcell...
Quantum Espresso Phonon (ph) Code: 3428: q-e_6.6_ph
Quantum Espresso PostProc (pp) Code: 1822: q-e_6.6_pp
Quantum Espresso Plain Wave (pw) Code: 0001: q-e_6.6_pw

9

Figure 10: Advanced user interface example for the computation of defect formation energies.

6. Conclusion and outlook

The semantic interconnection between the GUI and the AiiDA-workflow manager has been established through the implementation of an interface to the previously developed ontologies from D1.3. The workflow can be controlled by this AiiDA-wrapper, which is able to perform the computation for material properties through AiiDA by a semantic, ontology-based format (CUDS). In order to provide a seamless control to instantiate the CUDS, an additional REST API for SimPhoNy has been implemented. This also enables the ontology querying for input parameters through simple http-GET-requests.

The next steps will include the support of further SIESTA workflows through the wrapper, as well as the direct querying of the AllegroGraph triple store by SPARQL-queries or human readable keywords (and therefore without the need to build a sequence of interconnected http-requests to AiiDA) through the SimPhoNy REST-API.

ACRONYMS

API	- Application Programming Interface
CIF	– Crystallographic Information File
COD	- Crystallography Open Database
CUDS	- Common Universal Unified Data Structures
DoA	– Description of the Action
EMMO	- European Materials Modelling Ontology
GUI	- Graphical Users Interface
IDE	– Integrated Developing Environment
IM2D	– Interoperable Materials-To-Device
JSON	- JavaScript Object Notation
KPI	– Key Performance Indicator
OWL	- Web Ontology Language
RDF	- Resource Description Framework
REST	- Representational State Transfer
UUID	- Universally Unique IDentifier