Integrating Materials and Manufacturing Innovation (2025) 14:136–152
https://doi.org/10.1007/s40192-025-00395-5

TECHNICAL ARTICLE

Streamlining Multi-scale Materials Modeling: The MUSICODE Low-Code Approach for Simulation Workflows and Executable MODAs

Dario Campagna[1] · Alan Del Piccolo[1] · Konstantinos Kaklamanis[2] · Stanislav Šulc[3] · Mattia De Bernardi[1] · Flavio Ellero[1] · Saeideh F. Kalourazi[6] · Klaus Reimann[5] · Maria Andrea[2] · Konstantinos Kordos[2] · Michael Selzer[6] · Britta Nestler[6] · Dimitrios G. Papageorgiou[2] · Aron Kneer[5] · Davide Di Stefano[4] · Bořek Patzák[3] · Elefterios Lidorikis[2]

Received: 10 December 2024 / Accepted: 6 March 2025 / Published online: 22 April 2025

Abstract
MOdeling DAta (MODA) is a template for the standardized description of materials models. It is meant to guide users toward a complete high-level documentation of modeling workflows, starting from the end-user case down to the computational details. While a MODA workflow outlines the sequence of models and data interactions, it is not executable by itself and requires implementation. Traditionally, this involves custom coding to link together solvers and data sources or using an integration framework like the Multi-Physics Integration Framework (MuPIF). Both options demand software development expertise. To simplify this process, the MUSICODE H2020 project introduced in its Open Innovation Platform a low-code approach that enables easier transformation of MODA workflows into executable multi-scale workflows. This bridges the space between MODA workflow definition and its executable encoding, merging them into a single entity while ensuring FAIR compliance. In this paper, we illustrate the implementation of this approach based on the Business Process Model and Notation (BPMN) standard, demonstrating how MODA workflows can be converted into executable workflows through examples and a real-world use case.

Keywords Materials modeling · MODA · BPMN · MuPIF

Abbreviations

AiiDA	Automated Interactive Infrastructure and DAtabase for Computational Science
API	Application programming interface
BPMN	Business process model and notation
CFD	Computational fluid dynamics
DFT	Density functional theory
DMS	Data management system
EMMC	European Materials Modeling Council
FAIR	Findable, accessible, interoperable and reusable
HOMO/LUMO	Highest-occupied/lowest-unoccupied molecular orbital energy
HPC	High-performance computing
LDT	Logical data transfer
MD	Molecular dynamics
MODA	MOdeling DAta
MR	Materials relations
MuPIF	Multi-physics integration framework
OIPMM	Open innovation platform for material modeling
OLAE	Organic and large area electronics
PE	Physics/chemistry equation
PE+MR	Physics/chemistry equation + materials relations

Extended author information available on the last page of the article

Introduction

Materials modeling and simulation are widely used in diverse fields such as energy, environment, transportation, healthcare, ICT, and manufacturing, supporting research and development on new or improved applications. It provides crucial insights for discovering new materials, tailoring them, and designing materials for various structures and systems [1], while its penetration in the innovation process is only expected to increase given the nearly exponential growth of available computational resources. However, due to the vast diversity and complexity of materials, as well as the wide range of targeted applications, the field of materials modeling is composed of several distinct communities,

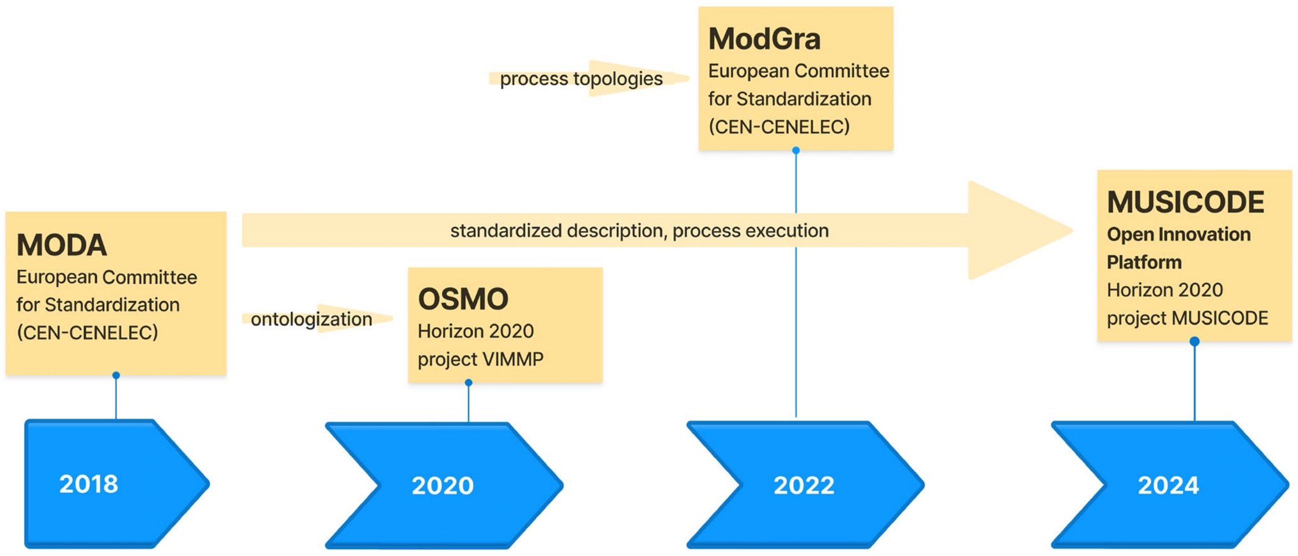

process topologies
ModGra
European Committee for Standardization (CEN-CENELEC)
MODA
European Committee for Standardization (CEN-CENELEC)
standardized description, process execution
ontologization
OSMO
Horizon 2020 project VIMMP
MUSICODE
Open Innovation Platform
Horizon 2020 project MUSICODE
2018
2020
2022
2024

Fig. 1 Timeline of MODA development and subsequent initiatives

each having developed its own terminology, often centered around specific application areas and particular types of models [2]. At the same time, it is clear that a strong interaction between these communities and industry is necessary to successfully apply materials modeling to industrial problems [1].

MOdeling DAta (MODA) [2] was developed to facilitate this interaction by establishing a common terminology in materials modeling. It defined basic entities and developed a shared conceptual framework for materials modeling, which has been adopted by the European Materials Modeling Council [1] (EMMC) community and brought together research groups that previously lacked a common terminology and viewpoint [3]. The adoption of MODA has been required in several Horizon 2020 calls and remains a specific requirement in the Horizon Europe work program. As a result, a significant collection of MODA workflow descriptions has been gathered from various Horizon projects.

MODA offers a template for the standardized description of material models and workflows. It is designed to guide users through comprehensive high-level documentation covering the end-user case down to the computational details. While it effectively addressed the need for a common language in materials modeling, its level of formalization remains intermediate, and several limitations have been identified: The descriptors for use cases, models, solvers, and processors in MODA are limited to plain-text entries [4]; the semantics of the characteristic blue-arrow edges that connect the sections (i.e., nodes) of a MODA workflow graph are undefined [4]; the notation for the workflow graph in MODA is not sufficiently univocal [4]; and the documentation generated with MODA is not machine-actionable [3], thus not providing executable representations of simulation workflows. To address these shortcomings, several initiatives have been developed, including OSMO [4] and ModGra [5]. Figure 1 shows the timeline of these developments and highlights their main purpose.

OSMO provides a complete ontologization of MODA, making MODA-based annotations machine-processable, and through the logical data transfer (LDT) notation, it clarifies the relationships between use case, model, solver, and processor entities within a MODA workflow [4]. ModGra, on the other hand, is an alternative to MODA, offering a more flexible and meaningful way of representing the relationships between physical quantities at different levels and how one simulation step can feed into another [3]. With ModGra, modeling and simulation workflows are documented using process topologies that are potentially machine-actionable [3]. However, while OSMO and ModGra address some limitations of MODA, the process for transforming a workflow representation into an executable form remains unclear. In addition, the adoption of LDT and OSMO has been limited [3].

To achieve an executable representation of a MODA workflow, two main options exist: custom coding to integrate solvers and data sources, or use of an integration framework like the Multi-Physics Integration Framework (MuPIF) [6] or AiiDA [7]. However, both approaches require significant software development expertise, along with the modeling and domain-specific skills needed to translate an industrial challenge into a solution using materials modeling and

[1] European Materials Modeling Council (EMMC ASBL) https://emmc.eu

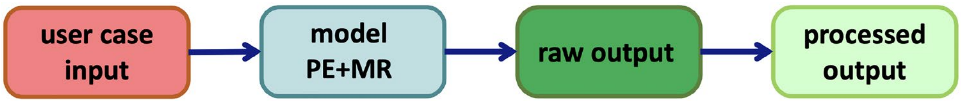


Fig. 2 MODA template for stand-alone model (source [10])

simulation tools. Ideally, a low-code or no-code approach to workflow definition and implementation is needed, which would simplify the process and reduce the associated coding costs. Such a concept was explored and introduced by the MUSICODE H2020 project.[2]

MUSICODE addresses the H2020 Call DT-NMBP-11-2020 "Open Innovation Platform for Materials Modeling." The project developed a novel Open Innovation Platform for Material Modeling (OIPMM) to empower the Organic and Large Area Electronics (OLAE) industry to make informed, physics-based, and data-driven decisions on materials design and processing, ultimately optimizing the efficiency and quality of OLAE device manufacturing. Within this platform, expert users can design complex modeling workflows that span multiple length scales from electronic to continuum. They can address different engineering issues such as new organic material formulations and properties, device architectures and performance, and optimization of fabrication processes. The resulting workflows can then be published on the platform as executable templates for non-expert users.

To employ this low-code approach to workflow design, the MUSICODE OIPMM leverages the Business Process Modeling and Notation (BPMN) standard [8]. Through BPMN, machine-actionable MODA workflows can be defined, which are then automatically translated into MuPIF workflows for execution. This approach leads to significant development time savings for expert users, but most importantly, it makes the process more accessible for people without coding abilities. In addition, the BPMN models become in their own merit a standardized documentation of what the workflow is doing: It significantly contributes to the need for well-documented modeling, complementary to the executable representation, that ensures repeatability of modeling tasks, as well as findability of the workflows.

This paper illustrates the implementation of the MUSICODE low-code approach, and it is structured as follows. Section From MODA to Executable Workflows describes the elements of MODA workflows and outlines two basic approaches for creating an executable representation of a MODA. The MUSICODE low-code approach is presented in Section MUSICODE BPMN-Based Low-Code Approach and demonstrated on a real use case in Section Results. The paper concludes with Section Conclusions, outlining the advantages of the presented low-code approach over traditional ones and highlighting future research directions.

From MODA to Executable Workflows

MODA [2] has been designed to standardize the collection of information on simulation models across various fields. It provides a step-by-step framework for the systematic description and documentation of models, encompassing all essential aspects: use case, model, solver, and post-processor. MODA supports the development of workflows in which multiple models can work together to answer specific user questions: It is therefore an aid to translators [9], i.e., experts in the process of translating an industrial need/challenge into a solution by means of materials modeling and simulation tools.

Materials inherently present a multi-scale problem, requiring a well-structured hierarchy of simulations at various levels of representation to develop accurate and predictive models. A MODA model is composed by an approximated physics/chemistry equation (PE) describing generic physics and its materials relations (MR) describing a specific material and its behavior [1]. A MODA workflow outlines this hierarchy by specifying how models are integrated into a chain and simulated, e.g., sequentially or concurrently. It also details the timing and methods for incorporating databases containing experimental or simulated data. More specifically, a workflow describes the input data for each model, the model itself, the raw output generated, and any post-processing steps applied to this output. MODA offers a comprehensive framework for representing simulations with material models and includes four workflow templates, each illustrating a different approach to running models [2]:

- Stand-alone model (see Fig. 2).
- Chain of linked models: models are solved sequentially (see Fig. 3).
- Chain of iterative coupled models: iterative solution of segregated models (see Fig. 4).
- Tightly coupled models: models solved together (see Fig. 5).

The MODA representation for a multi-scale workflow employs the most suitable combination of these templates to represent the hierarchy of simulations (see [1], for example). In addition to the graphical representation, the MODA

[2] MUSICODE H2020 project https://cordis.europa.eu/project/id/953187/reporting

includes a detailed textual description of each component of the simulation related to a model used in the chain. This description covers various aspects, including the specific element of the use case or system being simulated, the governing equations, the solver, the computational implementation, and any post-processing steps. For more detailed information on these sections of the MODA representation, we refer the reader to [2].

MODA serves as a crucial tool for translators, enabling them to document the modeling workflow throughout the translation process [9]. While MODA provides all the necessary information to document a multi-scale simulation workflow, it is not an executable format. To run the workflow, it must be implemented through appropriate means, usually requiring collaboration among experts with diverse skills, such as materials modeling and code development. A translator is expected to have a broad network of individuals and institutions capable of performing the actual modeling, along with a deep understanding of relevant software tools [9].

The code-heavy way to create an executable representation of a MODA is to develop code from scratch that implements the workflow, making it runnable in a local, distributed, or high-performance computing (HPC) environment. The code development process includes creating code for the simulation models, linking models together, transferring data between models within the workflow, and executing the workflow in the chosen environment. In addition to the effort required, this approach lacks generality and reusability.

An alternative approach involves using an integration platform, such as the Multi-Physics Integration Framework (MuPIF) [6], to generate a MuPIF executable workflow. Implementing a MuPIF workflow is facilitated by the tools provided within the framework and requires the following steps: supplying the simulation models, creating a model API for each model to integrate it into MuPIF, and defining a MuPIF workflow to link the models together. This last step can be accomplished either by writing Python code or using an editor that supports the graphical definition of basic

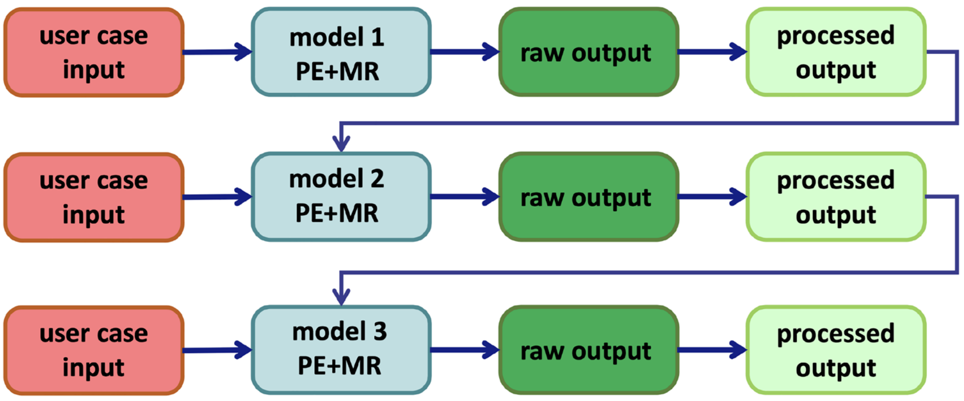


Fig. 3 MODA template for chain of linked models (source [10])

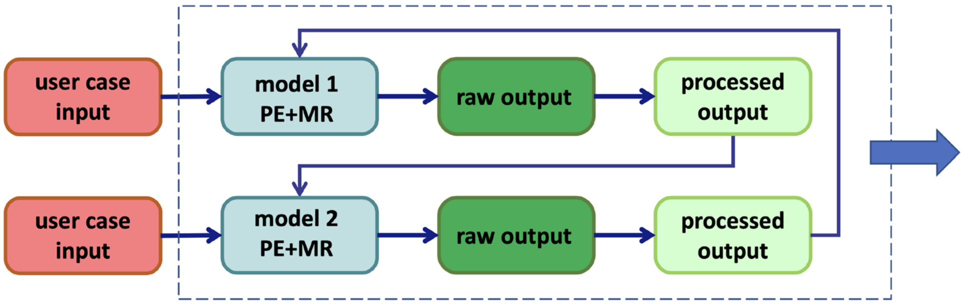


Fig. 4 MODA template for chain of iterative coupled models (source [10])

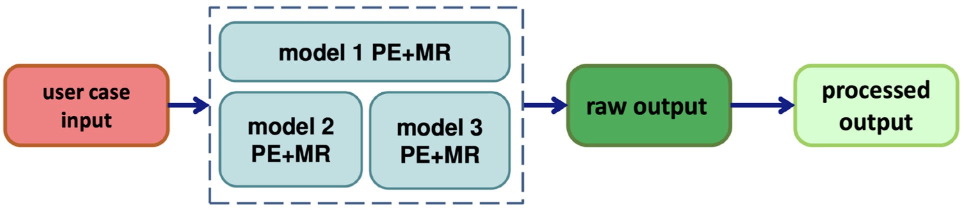


Fig. 5 MODA template for tightly coupled models (source [10])

workflows. Compared to the first approach, this method significantly reduces the amount of custom coding and the level of software development expertise required. For instance, there is no need to write scripts to run the workflow, as MuPIF handles this aspect. On top of this, the MuPIF APIs are generic, allowing to reuse model interfaces in different workflows, while additional benefits can come from the supported distributed deployment of models. Nonetheless, this approach still requires some expertise and familiarity with MuPIF to implement a workflow.

Both approaches described above require software development expertise and resources, such as time and manpower for custom coding, in addition to modeling expertise. Reducing the amount of custom coding needed for workflow implementation, while enhancing the compliance of models to FAIR [11] principles,[3] could lower the implementation costs in the translation process. One way to achieve this is through the low-code approach developed by the MUSICODE H2020 project, which is detailed in the next section.

MUSICODE BPMN-Based Low-Code Approach

A low-code development platform provides a development environment to create application software through a graphical user interface, producing entirely operational applications without the need for additional coding. BPMN [8] is an Object Management Group[4] and ISO standard[5] for the representation of Business Processes. The standard defines a graphical modeling language that is expressive enough to model scientific workflows and algorithms [12, 13], formalizes the execution semantics for all its elements, defines an extensibility mechanism, and specifies an XML serialization of BPMN models. BPMN models can be created by using one of many graphical editors available in the market, e.g., Cardanit,[6] therefore writing little to no programming code in the process. The MUSICODE project employs BPMN as a digital MODA representation of a materials modeling workflow that produces a concretely executable simulation workflow that runs on the MuPIF platform. Thus, BPMN bridges the space between MODA workflow definition and its executable encoding merging them into a single entity, while ensuring FAIR compliance and opening the possibility for reuse and generation of additional workflow implementations. In fact, the MUSICODE BPMN-based low-code approach enhances the following FAIR principles for MODA.

- *F1: (Meta)data are assigned a globally unique and persistent identifier* - The BPMN XML representation makes it possible to include globally unique identifiers for MODA workflows.
- *F2: Data are described with rich metadata* - With the BPMN XML representation, the MODA workflow and its metadata are encoded within the same file.
- *I1: (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation* - BPMN is a knowledge representation language that has all the listed characteristics.
- *I2: (Meta)data use vocabularies that follow the FAIR principles* - BPMN elements and attributes are defined in the BPMN specification [8], and the XML representation is formalized via an XML schema definition, both are easily findable and accessible.
- *R1: (Meta)data are richly described with a plurality of accurate and relevant attributes* - With BPMN, it is possible to encode any relevant attribute in the XML representation.

Figure 6 presents a graphical representation of the process to go from a MODA workflow to its execution and highlights the main characteristics of and differences between the code-heavy approach, the approach based on MuPIF, and the MUSICODE low-code approach. The following sections dive into the details of the translation of MODA workflows into BPMN models and the conversion of the latter into MuPIF workflows for execution.

From MODA to BPMN

The building blocks of a BPMN workflow are Activities, Events, and Gateways. The control flow is designed by drawing arcs known as Sequence Flows from one block to the next, while the data flow is designed by drawing Data Associations and by using data elements to represent the actual data. The types of BPMN data elements are Data Input, Data Output, Data Object, and Data Store. They are designed to contain the following categories of data:

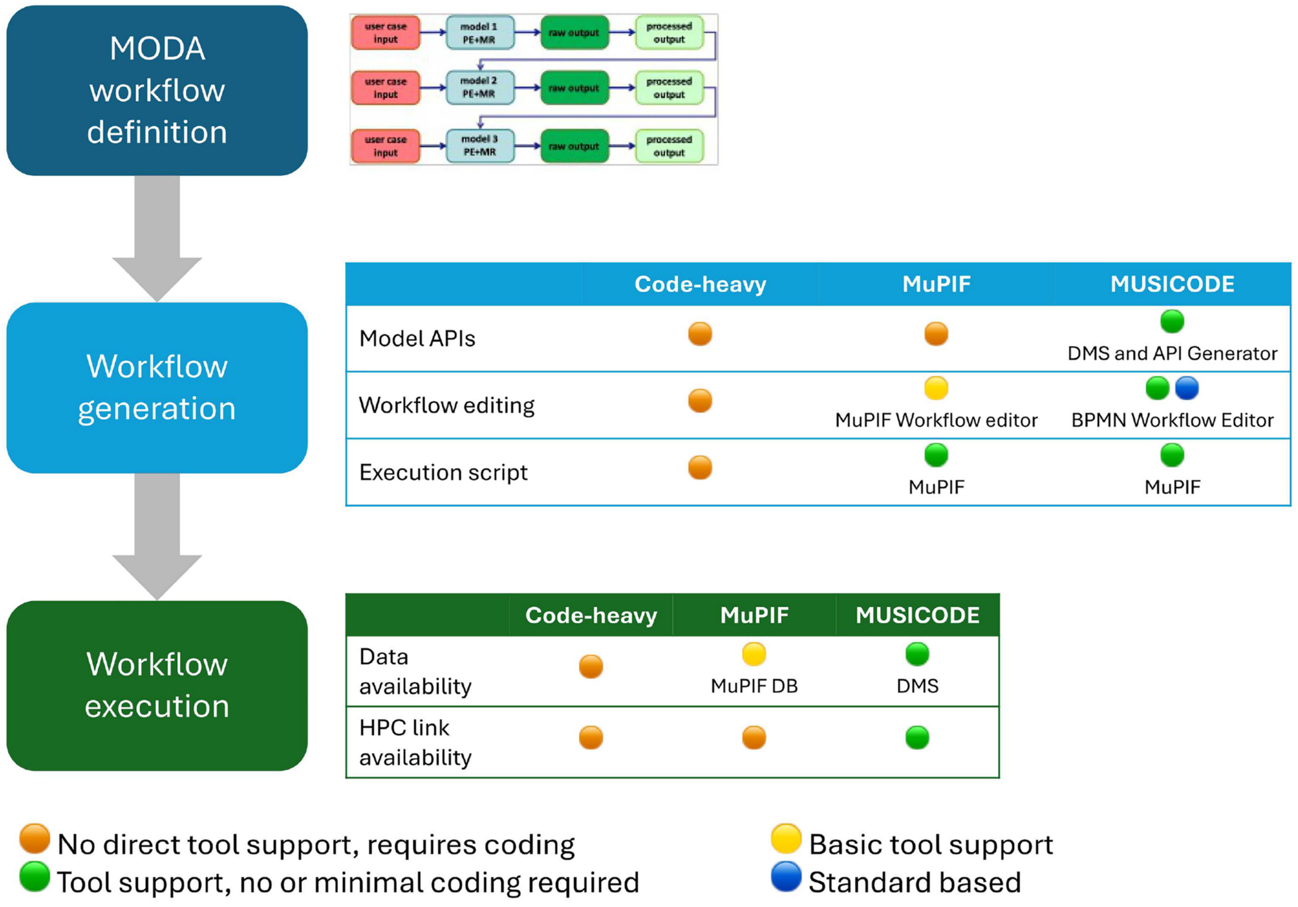
- Data Input: an input provided to the BPMN workflow from the outside (e.g., from user input, or from an external information system);
- Data Output: an output produced by the workflow and available to the outside;
- Data Object: an information that is transmitted from one activity (the producer of such information) to another (the consumer) within the workflow;

[3] FAIR Principles: Findability, Accessibility, Interoperability, and Re-Usability https://www.go-fair.org/fair-principles/

[4] Object Management Group https://www.omg.org

[5] ISO/IEC 19510:2013 https://www.iso.org/standard/62652.html

[6] ESTECO Cardanit https://www.cardanit.com



	Code-heavy	MuPIF	MUSICODE
Model APIs	🟠	🟠	🟢 DMS and API Generator
Workflow editing	🟠	🟡 MuPIF Workflow editor	🟢🔵 BPMN Workflow Editor
Execution script	🟠	🟢 MuPIF	🟢 MuPIF

	Code-heavy	MuPIF	MUSICODE
Data availability	🟠	🟡 MuPIF DB	🟢 DMS
HPC link availability	🟠	🟠	🟢

🟠 No direct tool support, requires coding	🟡 Basic tool support
🟢 Tool support, no or minimal coding required	🔵 Standard based

Fig. 6 From MODA to executable workflows: characteristics of different approaches

- Data Store: an information that is persisted and remains available across different executions of the workflow.

Only the first three types of data elements were necessary in the MUSICODE implementation.

As we will show in this section, each one of the four MODA template workflows mentioned above can be translated to a BPMN workflow.

Elementary Units

The translation from MODA to BPMN implies the preliminary translation of the elementary units that compose a MODA workflow into one or more elements of a BPMN workflow architecture. Such units are the operative units, i.e., the models and the data, the latter being the inputs and the outputs. The MODA models shall be translated to BPMN Tasks. Tasks are a type of Activity and represent a job that needs to be carried out by a system or a human operator. In general, each MODA model is translated to two BPMN Tasks:

- Model Task: It represents the process execution step. It solves a physics problem by resorting to an external computational model and produces new entities as output (e.g., the structure of a new molecule). It is basically a wrapper for such model and provides it the required input for the computation;
- Post-processing Task: It manipulates the Model Task outputs to extract properties from the raw data.

For each template, the data elements are translated as follows:

- The user case input is translated to a Data Input consumed by a Model Task;
- The raw output is translated to a Data Object produced by a Model Task and consumed by the downstream post-processing Task;

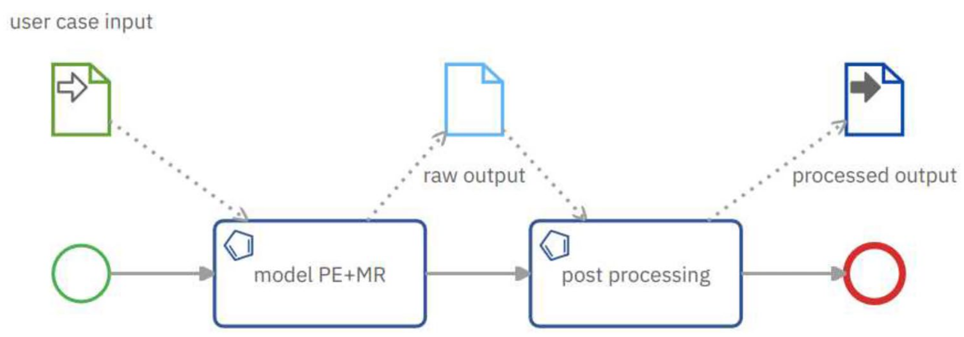


Fig. 7 BPMN translation for the stand-alone MODA model template

- The processed output is translated to a Data Object or Data Output produced by the post-processing Task.

Template Workflows

In this section, we present the procedure for translating each MODA template workflow to BPMN. While all of the possible MODA models are enumerated here, the implementation of the MUSICODE platform specifically required the translation of only the first three (see Section Implementation in MUSICODE). Starting from the elementary units, the specific translation procedure for each MODA template workflow can be carried out as follows:

1. Stand-alone model (see Fig. 7): the only model present in the workflow is translated to the above-mentioned pair of Tasks;
2. Chain of linked models (see Fig. 8): each stand-alone model is translated separately, and then, the models are connected together with a Sequence Flow transferring the control from the post-processing Task of one model to the Model Task of the next;
3. Chain of iterative coupled models (see Fig. 9): the translation is analogous to the one for linked models, but with a further specification concerning the data flows, the processed output of one model serves as input to the next. Additionally, two Exclusive Gateways are employed to model the iteration loop;
4. Tightly coupled models (see Fig. 10): the models are grouped into an Ad Hoc Sub-Process to mark that such Tasks can be executed in parallel. No data flows are currently documented for Ad Hoc Sub-Processes, thus leaving space for interpretation according to the semantics of the MODA model.

The procedure to translate the four MODA templates to BPMN processes is straightforward. At the same time, given the intermediate level of formalization of MODA and its ambiguities, translating some MODA workflows might require slightly different steps or the use of additional BPMN elements. The high number of workflow control-flow patterns[7] supported by BPMN suggests that any workflow defined with MODA can be translated to an executable BPMN process.

Implementation in MUSICODE

In the MUSICODE platform, modelers use a BPMN editor based on Cardanit to map their MODA workflows to BPMN processes. Tasks within these processes represent physics-based simulation models available in the OIPMM, while data elements describe the data exchanges between them. Each type of MODA model is translated to a new BPMN element named Model Task. Model Tasks are defined via the standard's extension mechanism. Both MODA models and Model Tasks represent distinct, atomic parts in the respective workflows, and therefore, they retain their semantic value in the translation. This allows for a static, beforehand translation of each MODA model into a corresponding Model Task.

The metadata of MODA models for MUSICODE are contained in JSON files stored inside the platform's Data Management System (DMS), based on ANSYS Granta MI.[8] The translation consists in extracting information from a JSON representation of the MODA model to shape the Data Inputs and Data Outputs of the Model Task associated with it (see Fig. 11).

Three MODA model templates are supported in MUSICODE to satisfy the needs of the project user cases: the stand-alone, the chain of linked models, and a simplified version of the iterative model. In such version, the executions are iterated on a single model, according to the Multi-Instance Characteristics pattern of the BPMN standard (see Fig. 12). The number of iterations depends on the number of values contained in a selected Data Input (e.g., the number of values of an array), and the execution of the model

[7] Workflow Patterns Initiative, control-flow patterns http://workflowpatterns.com/patterns/control/

[8] ANSYS Granta MI https://www.ansys.com/it-it/products/materials/granta-mi

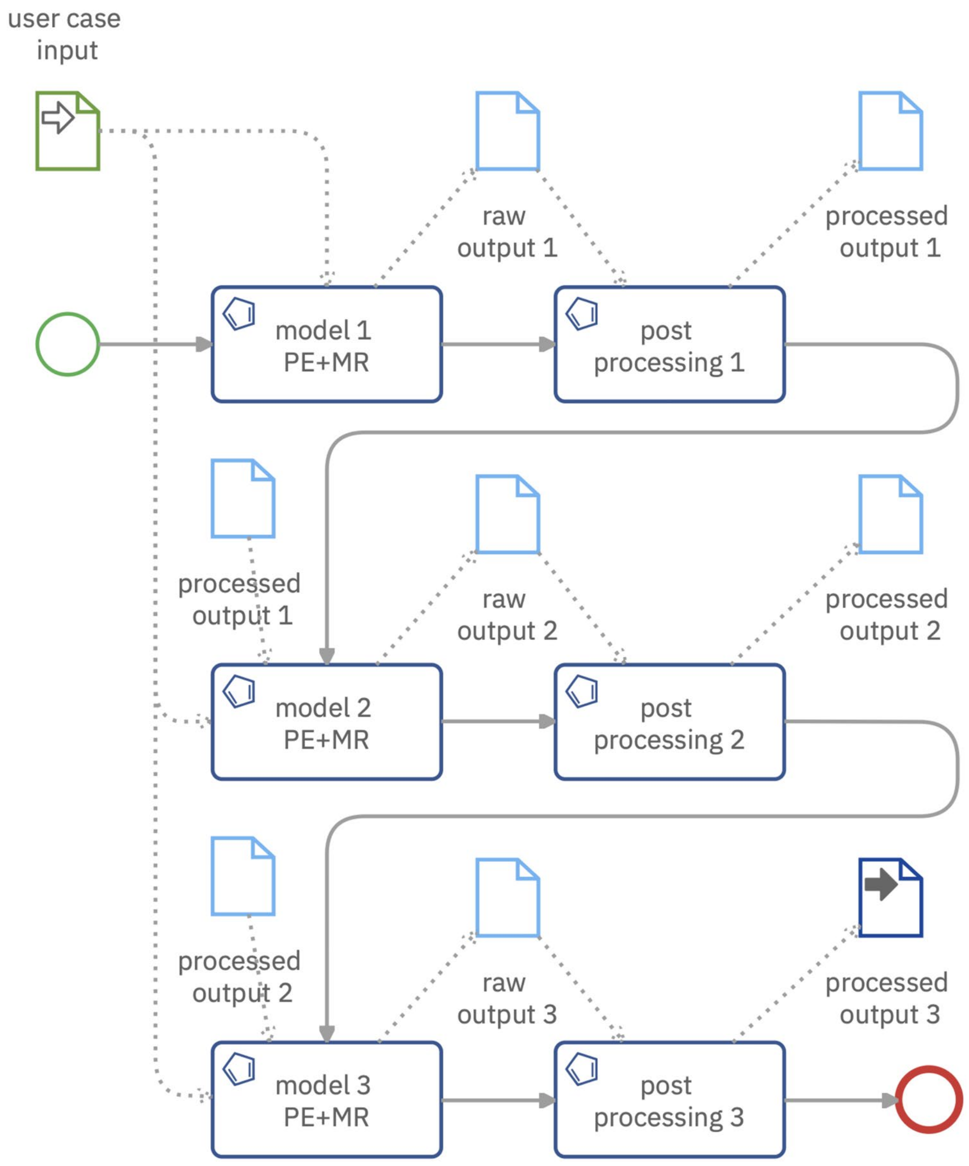


Fig. 8 BPMN translation for the chain of linked MODA models template

instances can be carried out in parallel in a (or multiple) connected HPC facility(ies).

As long as the models available within the OIPMM are sufficient to implement a MODA workflow, no additional coding is required to generate an executable representation of a given MODA workflow. However, if the necessary models are not yet integrated into the OIPMM, the user only needs to implement the missing models and their corresponding MuPIF model APIs.

From BPMN to MuPIF

To obtain an executable workflow, the BPMN process must be converted into a MuPIF-compliant workflow. Alongside the visual block representation, the BPMN standard provides an XML representation of its elements. Therefore, a BPMN workflow is essentially an XML file. Conversely, a MuPIF-compliant executable workflow is a function written in Python language. The translation is performed by converting the BPMN XML into a JSON representation of a MuPIF workflow, and then resorting to a library that is part of the MuPIF ecosystem to generate the Python code, as depicted in Fig. 13.

The JSON representation of the workflow describes the execution and data flow models. The execution model is built from execution blocks. The data model determines data dependencies between the input/output slots of the workflow and the input/output slots of the individual execution blocks. Each I/O slot is identified by its name and annotated with the slot datatype (determining the physical meaning of the slot) and the flag indicating whether the slot is optional or

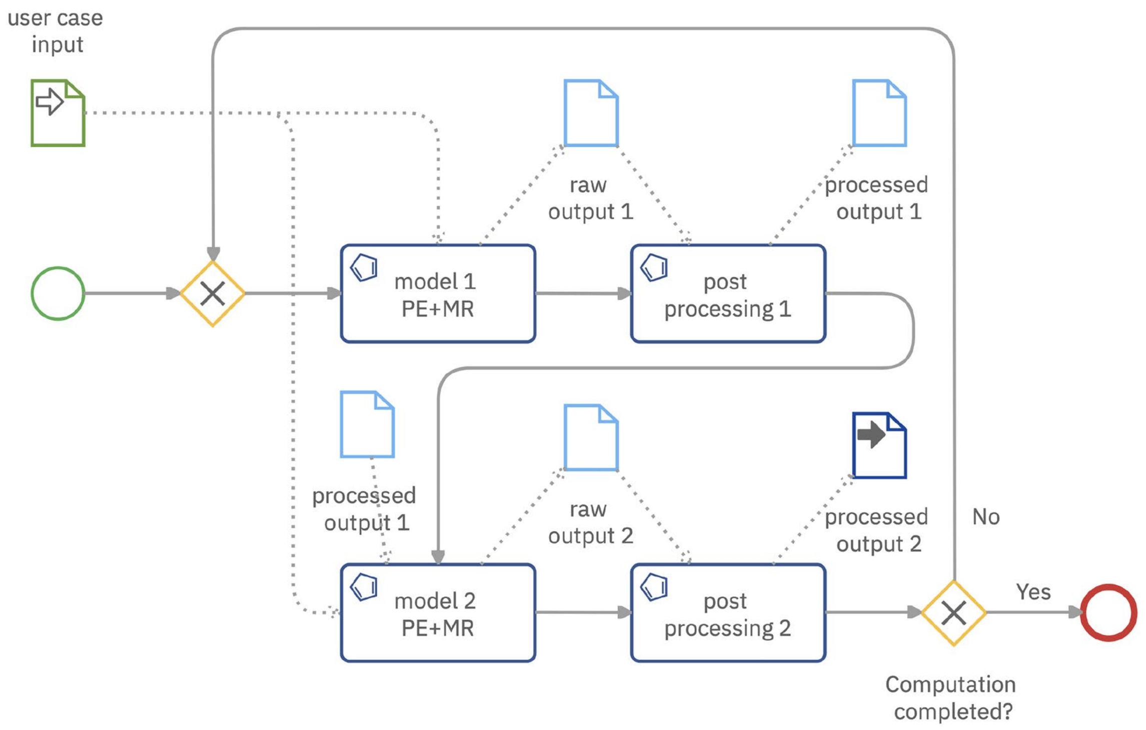
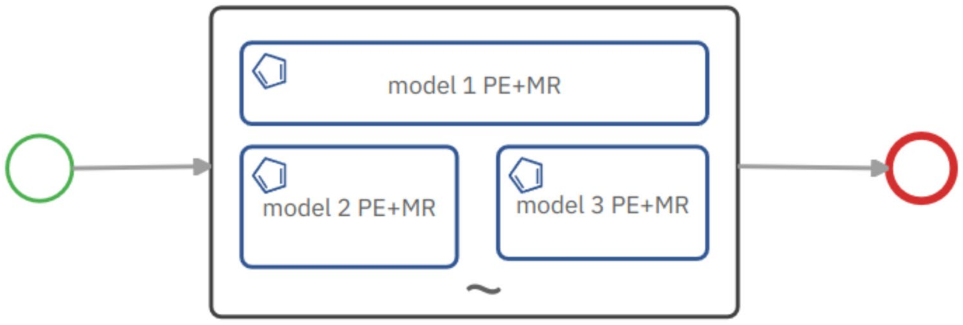


Fig. 9 BPMN translation for the chain of iterative coupled MODA models template

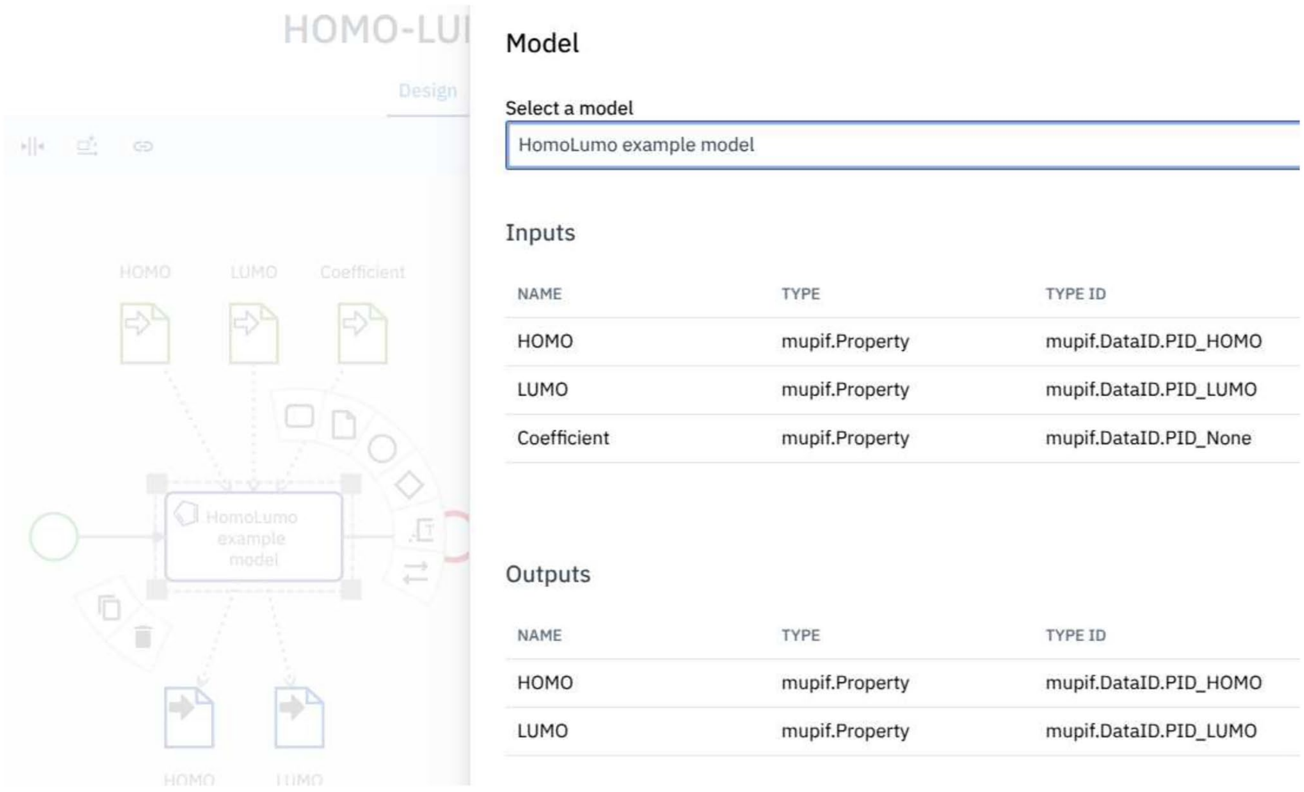
Fig. 10 BPMN translation for the tightly coupled MODA models template



not. The data model consists of a list of pairs, containing source and target slots.

Each execution block represents a specific execution logic (such as sequential block, loop block, if-then-else block, or simulation model execution block) and can contain nested execution blocks. Each block has methods that generate block initialization, code execution, and finalization in Python. The initialization code typically resolves execution block inputs (determined by the data model). For example, the loop block initialization code initializes the loop counter, and the execution code loops over nested blocks. Furthermore, it calls the initialization and the code generation methods of such nested blocks; finally, it checks if loop termination condition is met. In short, the execution model combines pre-configured execution blocks to represent target workflow execution logic. The modular design of the generator allows introducing custom-based execution blocks to meet the specific needs.

The translation from BPMN to MuPIF-compliant code requires one last specification: The translator needs to map the BPMN process Data Inputs and Data Outputs (i.e., the classes of data elements that define the inputs and outputs of each Model Task) into attributes of the material data that were previously defined in the MODA (e.g., “Grains” consisting of “Molecules” for some representative MUSICODE use cases). Each input and output is supplied with attributes defining its physical meaning, measurement unit, etc. The data mapping is validated to minimize the errors and ensure valid code generation.



Model

Select a model

HomoLumo example model

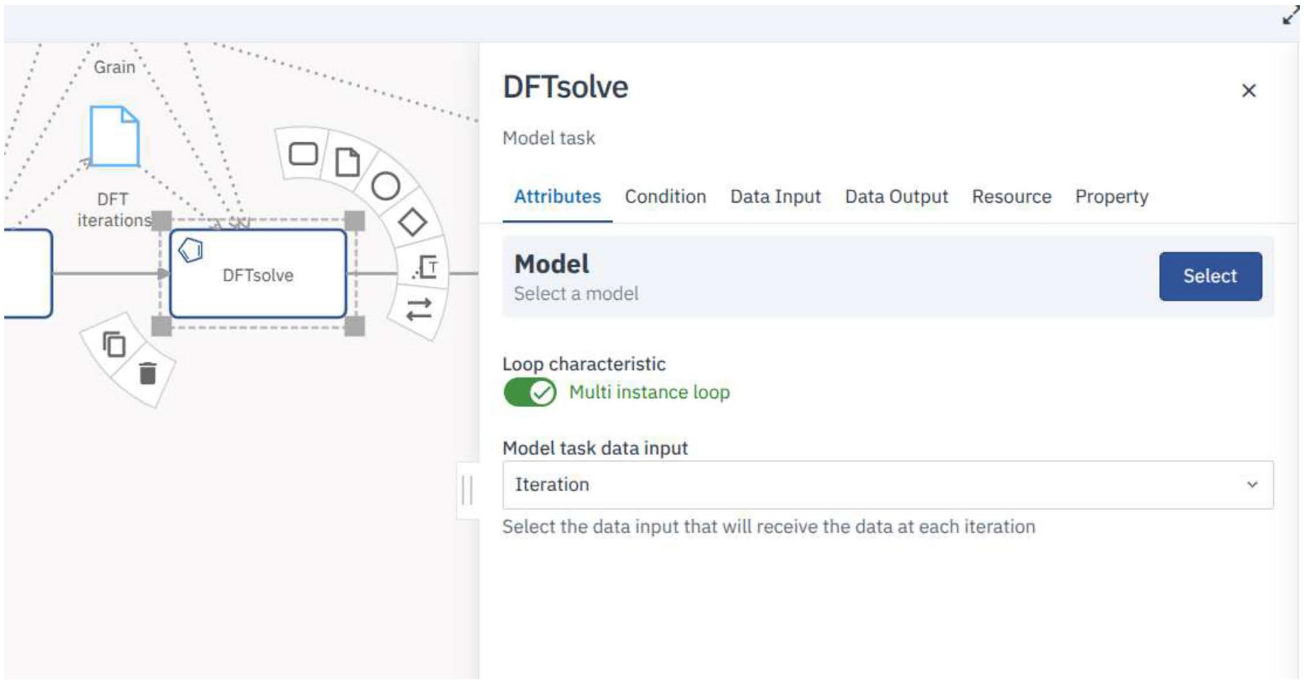
Inputs

NAME	TYPE	TYPE ID
HOMO	mupif.Property	mupif.DataID.PID_HOMO
LUMO	mupif.Property	mupif.DataID.PID_LUMO
Coefficient	mupif.Property	mupif.DataID.PID_None

Outputs

NAME	TYPE	TYPE ID
HOMO	mupif.Property	mupif.DataID.PID_HOMO
LUMO	mupif.Property	mupif.DataID.PID_LUMO

Fig. 11 When the translator selects a MODA model, its inputs and outputs are mapped into BPMN Data Inputs and Data Outputs



DFTsolve

Model task

Attributes Condition Data Input Data Output Resource Property

Model
Select a model

Loop characteristic
Multi instance loop

Model task data input
Iteration
Select the data input that will receive the data at each iteration

Fig. 12 A multi-instance loop is set for the "DFT solve" model task

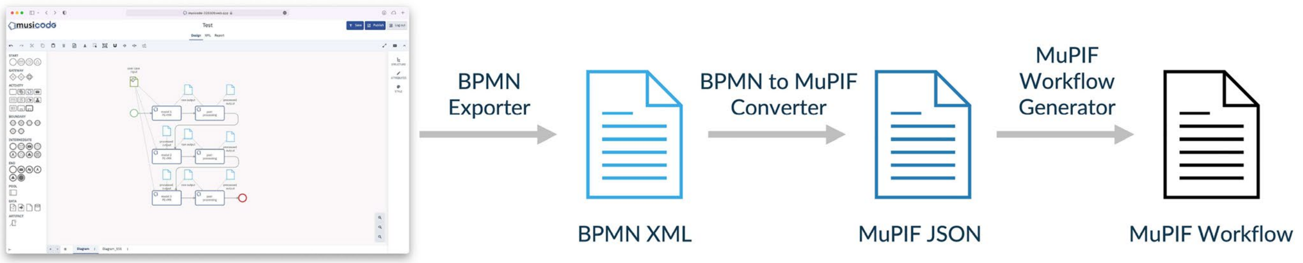


Fig. 13 The automated steps to translate a BPMN process into a MuPIF workflow

The material data are stored inside the MUSICODE DMS as tables; therefore, the translator is required to scan such tables and select the desired attribute for each data element (see Fig. 14). The rest of the translation is performed automatically by the MuPIF library. The result is Python code ready to be executed. Although in the MUSICODE platform the execution is meant to be performed by the distributed architecture of MuPIF (employing input data extracted from the DMS), the generated Python code can run on any computing system, from a software developer's local machine to a HPC cluster.

Results

In this section, we demonstrate the MUSICODE BPMN-based graphical workflow editor used on a simplified use case relevant to organic electronic materials. Specifically, we address the electronic state distributions in a sample of disordered organic molecules at a target temperature T. By electronic states here, we simply mean the highest-occupied/lowest-unoccupied molecular orbital energies (HOMO/LUMO), which are important in determining absorption bands and charge transport from molecule to molecule. Most organic electronic materials are generally found in a disordered state (amorphous or partially amorphous), and thus, the HOMO-LUMO values will be different for each molecule due to being at a different local deformation (due to its immediate neighborhood) and due to the thermal motion. The distribution of HOMO-LUMO values gives information about the average size of the electronic band gap (i.e., $\langle E_g \rangle = \langle LUMO - HOMO \rangle$) and the energetic disorder (i.e., σ, the standard deviation of LUMO states for electron transport and of HOMO states for hole transport), which determine the optical absorption edge and the density of traps, respectively. To achieve this, an atomistic molecular dynamics (MD) simulation is needed to relax the molecular assembly at the target temperature T and then perform separate electronic density functional theory (DFT) calculations in every molecule in the assembly to get the electronic states. Thus, the output of the MD simulation is the input of the parallel DFT simulations, whose raw output is in turn the input to the post-processing module that will extract the $\langle E_g \rangle$ and σ values. In the following, we will describe the process of translation from a MODA workflow to a BPMN workflow and generation of MuPIF executable code, along with the necessary links to and from the DMS for the input and output data.

Use Case Workflow and its MODA Representation

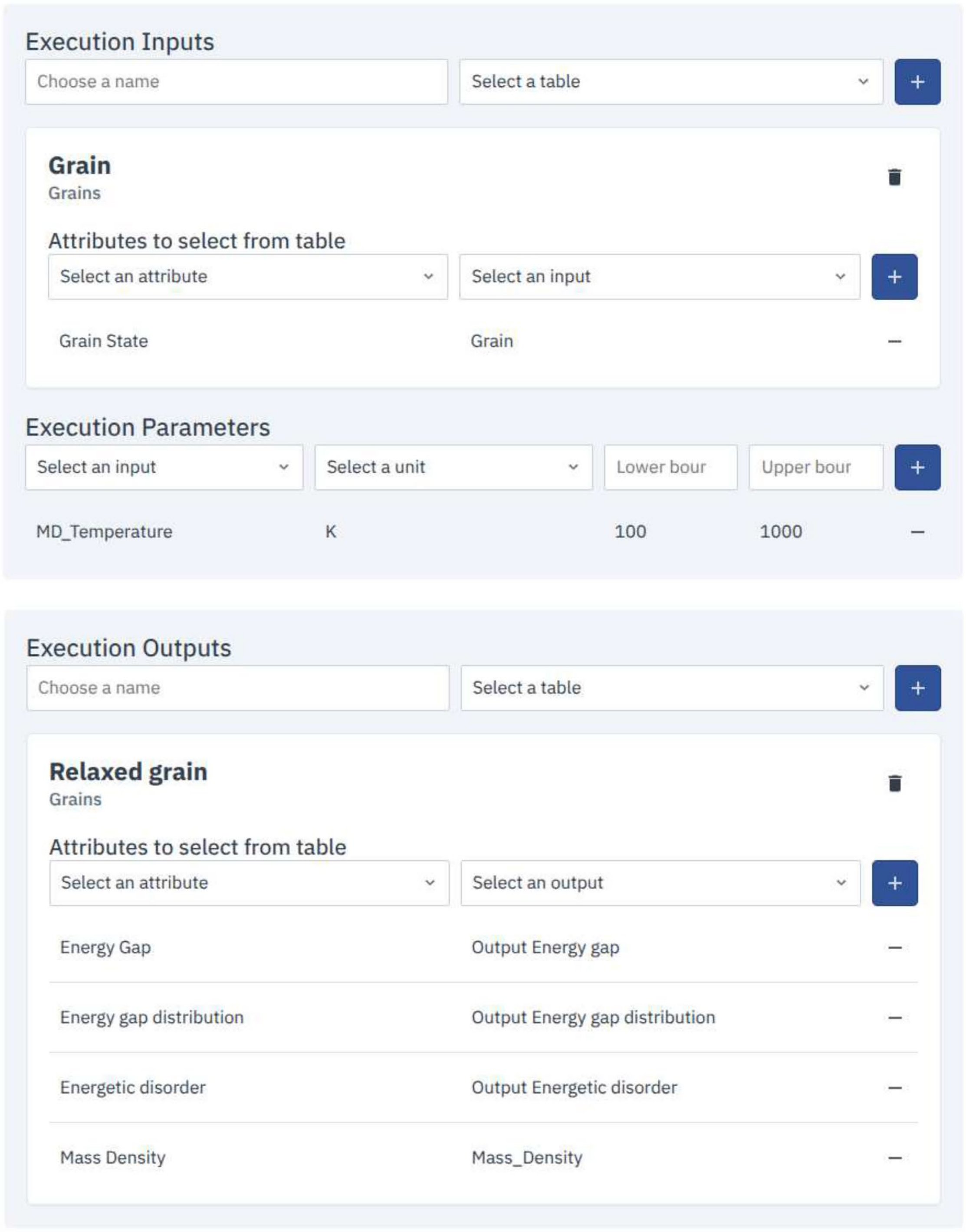
In the context of MUSICODE data representations, a "grain" is a molecular ensemble, i.e., a computational box consisting of a number of molecules. Information about this grain (e.g., cell size, atomic information, atomic positions, masses, partial charges, interaction potentials, boundary conditions, etc.) is all suitably stored in a single HDF5 container in the DMS, searchable and findable by the workflow editor. Suppose that such a grain of interest is indeed available in the DMS, but was thermodynamically relaxed at a temperature T_0 which is different than the target temperature T. In our use case scenario, the user wants to relax the grain in the new temperature T, extract one by one all molecules and perform electronic structure calculations (ideally in parallel and not serially one after the other), and then post-process the raw output to extract the following quantities: mass density, average band gap, and disorder in states (see Fig. 15).

In more detail, such a workflow scenario requires the following modeling steps:

- A call to a MD model, for which the open solver NAMD[9] is used, to elevate the temperature from T_0 to T and equilibrate the system at the higher temperature using a suitable thermostat. Given the large size of the molecular system, a parallel execution with CHARM++[10] is needed on an HPC cluster;

[9] NAMD https://www.ks.uiuc.edu/Research/namd/

[10] CHARM++ https://charm.cs.illinois.edu/


Execution Inputs

Choose a name | Select a table | +

Grain
Grains

Attributes to select from table

Select an attribute | Select an input | +

Grain State	Grain	–

Execution Parameters

Select an input | Select a unit | Lower bour | Upper bour | +

MD_Temperature	K	100	1000	–

Execution Outputs

Choose a name | Select a table | +

Relaxed grain
Grains

Attributes to select from table

Select an attribute | Select an output | +

Energy Gap	Output Energy gap	–
Energy gap distribution	Output Energy gap distribution	–
Energetic disorder	Output Energetic disorder	–
Mass Density	Mass_Density	–

Fig. 14 The interface that enables the translator to map BPMN Data Inputs and Outputs into material data attributes

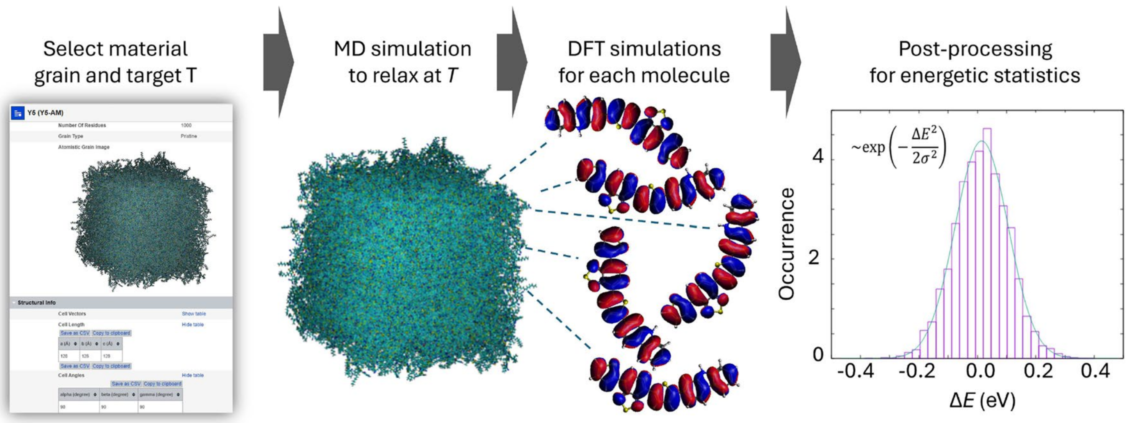
- Multiple (parallel) calls to electronic structure calculations, for which the Gaussian[11] DFT solver is used. Each run, in turn, is by itself executed in parallel with LINDA[12] on an HPC cluster;
- Post-processing to gather the results and compute statistics.

The models required by the workflow are the following:

- A NAMD model to perform the equilibration;
- A pre-processor to determine and prepare the DFT runs;

[11] Gaussian https://gaussian.com/

[12] LINDA https://gaussian.com/lindaprod/


Select material grain and target T
MD simulation to relax at T
DFT simulations for each molecule
Post-processing for energetic statistics
$\sim\exp\left(-\frac{\Delta E^2}{2\sigma^2}\right)$
Occurrence
0
2
4
-0.4
-0.2
0.0
0.2
0.4
ΔE (eV)

Y5 (Y5-AM)

Number Of Residues	1000
Grain Type	Pristine
Atomistic Grain Image	

Structural Info

Cell Vectors Show table

Cell Length Hide table

Save as CSV Copy to clipboard

a (Å)	b (Å)	c (Å)
128	128	128

Save as CSV Copy to clipboard

Cell Angles Hide table

Save as CSV Copy to clipboard

alpha (degree)	beta (degree)	gamma (degree)
90	90	90

Fig. 15 The simple use case scenario demonstrated here: a “grain” record is found and retrieved from the DMS; MD simulation is performed to relax in new temperature T; DFT simulations for each molecule in the grain; and post-processing to extract useful information

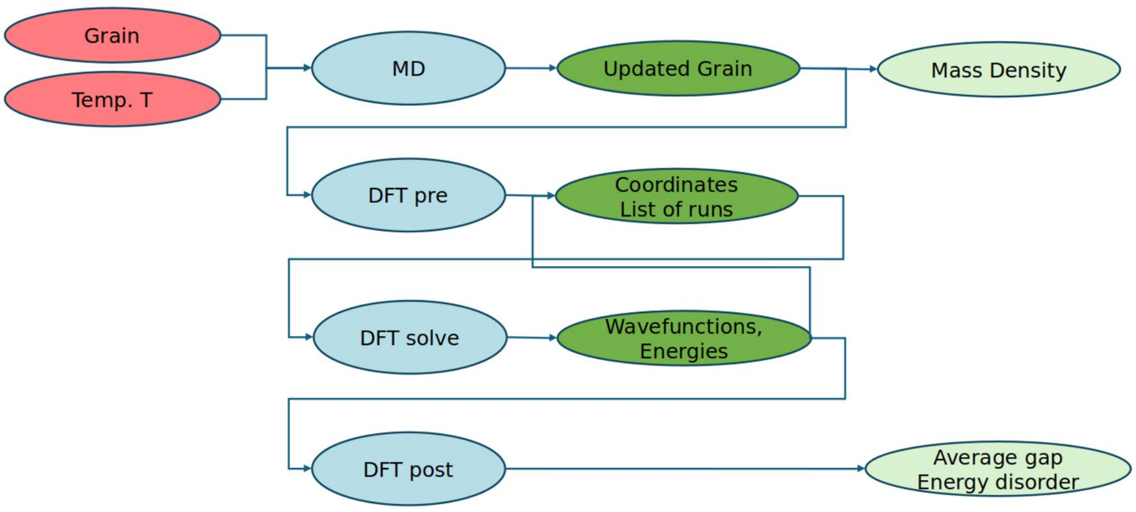


Fig. 16 MODA workflow for the grain relaxation workflow

- A Gaussian model to perform the DFT runs in parallel;
- A post-processor to gather and prepare the output.

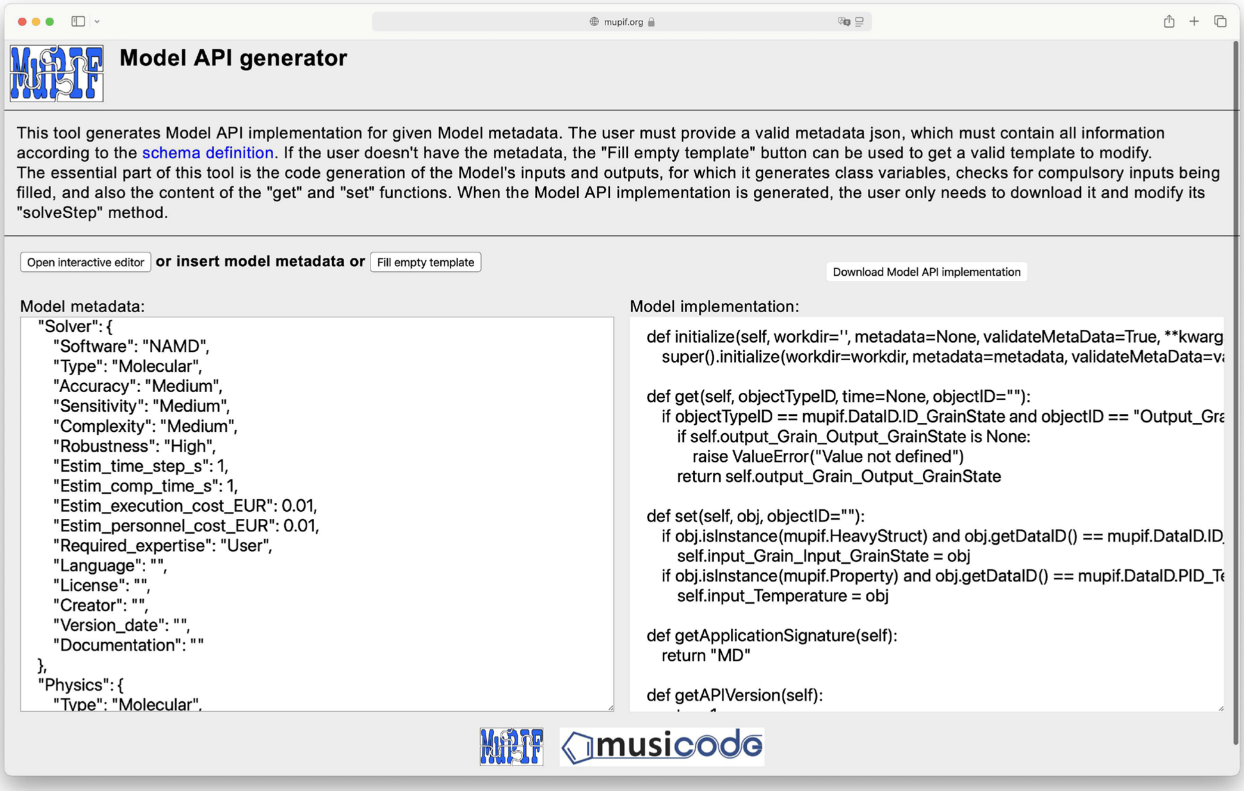
The required data elements are the following:

- Two inputs: the grain at temperature T_0, and the desired temperature T;
- Three outputs: mass density, average band energy gap, and energetic disorder.

The resulting MODA workflow is depicted in Fig. 16.

MuPIF Models

For each MODA model in MUSICODE, a JSON file is stored into the DMS. The content of such JSON file is referred to as “model metadata” and is used not only to translate the MODA model into BPMN, but also to detail the translation of the BPMN workflow into MuPIF-compliant Python code. For the implementation of a model toward executable workflows, the creation of an API is required. Each API provides the possibility to request actual executions of the selected solver, alongside with data pre- and post-processing. The API enables the users to operate models using an abstract, generic interface. The interface allows to map model inputs and outputs, request model update for a specific time step, etc. The implementation of such APIs is



Model API generator

This tool generates Model API implementation for given Model metadata. The user must provide a valid metadata json, which must contain all information according to the schema definition. If the user doesn't have the metadata, the "Fill empty template" button can be used to get a valid template to modify.
The essential part of this tool is the code generation of the Model's inputs and outputs, for which it generates class variables, checks for compulsory inputs being filled, and also the content of the "get" and "set" functions. When the Model API implementation is generated, the user only needs to download it and modify its "solveStep" method.

or insert model metadata or

Model metadata:

```
  "Solver": {
    "Software": "NAMD",
    "Type": "Molecular",
    "Accuracy": "Medium",
    "Sensitivity": "Medium",
    "Complexity": "Medium",
    "Robustness": "High",
    "Estim_time_step_s": 1,
    "Estim_comp_time_s": 1,
    "Estim_execution_cost_EUR": 0.01,
    "Estim_personnel_cost_EUR": 0.01,
    "Required_expertise": "User",
    "Language": "",
    "License": "",
    "Creator": "",
    "Version_date": "",
    "Documentation": ""
  },
  "Physics": {
    "Type": "Molecular",
```

Model implementation:

```
def initialize(self, workdir='', metadata=None, validateMetaData=True, **kwarg
    super().initialize(workdir=workdir, metadata=metadata, validateMetaData=va

def get(self, objectTypeID, time=None, objectID=""):
    if objectTypeID == mupif.DataID.ID_GrainState and objectID == "Output_Gra
        if self.output_Grain_Output_GrainState is None:
            raise ValueError("Value not defined")
        return self.output_Grain_Output_GrainState

def set(self, obj, objectID=""):
    if obj.isInstance(mupif.HeavyStruct) and obj.getDataID() == mupif.DataID.ID_
        self.input_Grain_Input_GrainState = obj
    if obj.isInstance(mupif.Property) and obj.getDataID() == mupif.DataID.PID_Te
        self.input_Temperature = obj

def getApplicationSignature(self):
    return "MD"

def getAPIVersion(self):
```

Fig. 17 MuPIF Automatic API generator (the user enters the model metadata in the left text box, and the generated model implementation appears in the right text box)

one of the few software development efforts required by the framework presented in this paper. Both domain and coding expertise are necessary to complete such a task. To alleviate it, MUSICODE partners developed a tool for generating model APIs, the Automatic API Generator[13] (see Fig. 17), which creates a MuPIF-compliant skeleton of the API implementation from the model metadata. For additional details on the implementation of a Model API and an example using the Gaussian model, we refer the reader to the electronic supplementary material.

BPMN Representation of Workflow

Once the APIs for the four models required in the workflow have been implemented (here named MD_API, DFTpre, DFTsolve, and DFTpost, respectively), the models are stored into a model registry managed by MuPIF. Hence, they become visible to the BPMN visual editor of the MUSICODE platform, named the Workflow Editor. Model Tasks are meant to represent the core of the workflow: after dragging a new Model Task into the drawing canvas, the translator selects which one of the available models (four in this case) it will represent. Data Inputs and Data Outputs for the Model Task are created automatically according to the selected model. The creation of the BPMN workflow is simplified by features of the Workflow Editor such as the automatic drawing of data flows and the easy selection of the input data for a loop architecture.

The resulting BPMN workflow is shown below (see Fig. 18). In this example, while the MODA inputs are mapped one-to-one to the BPMN Data Inputs, an additional BPMN Data Output accounts for the energy gap distribution. Data Objects are added to trace the flow of data between the Model Tasks. Finally, the Multi instance loop option is selected for the Model Task referring to the solver ("DFTsolve"), and an input for the loop is selected among the Model Task's Data Inputs, thus specifying the variable to be looped through. The whole BPMN workflow undergoes an automatic validation procedure, after which the translator is notified of possible formal violations to the BPMN standard that need to be corrected.

[13] MuPIF Automatic API Generator https://mupif.org/tools/api_generator.html

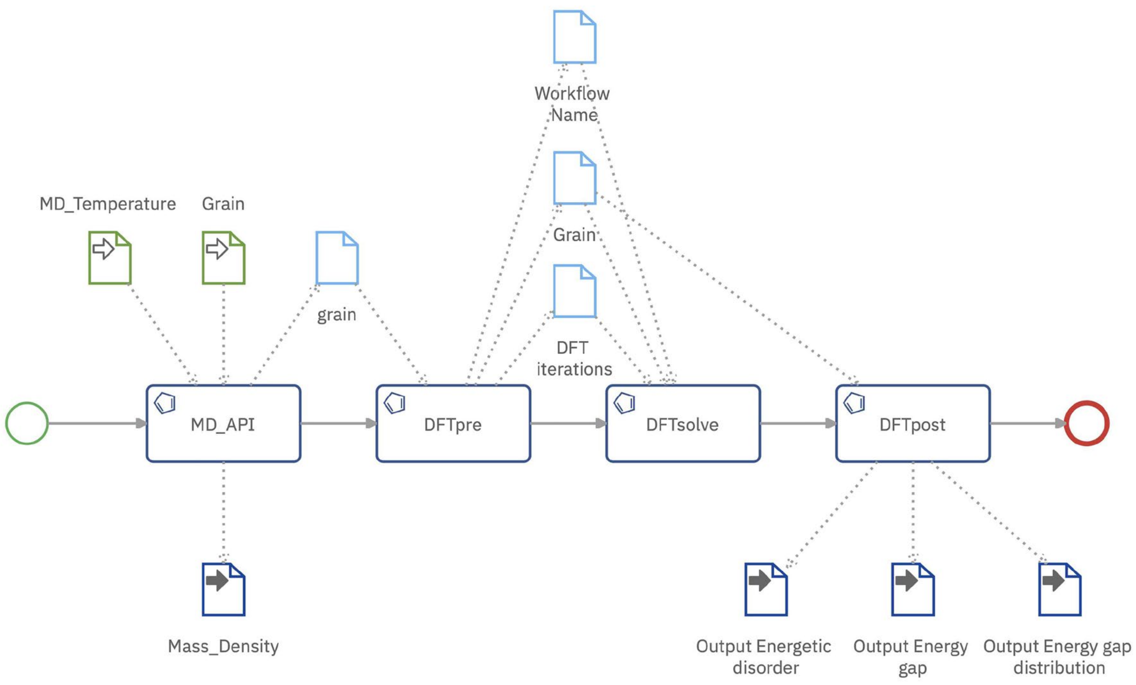


Fig. 18 BPMN representation of the MODA workflow

MuPIF Workflow and Execution

The Workflow Editor's user interface guides the translator through the last phase, which is to complete the mapping procedure, as mentioned in Section From BPMN to MuPIF. The BPMN Data Inputs and Data Outputs need to be mapped to specific data contained in the DMS, namely attributes of the material data. Alternatively, a Data Input can be marked as a parameter to be entered manually when launching a simulation execution. The mapping procedure is preceded by a validation step: The Workflow Editor checks the formal correctness of the BPMN workflow (e.g., data type matching) automatically and asks the translator to fix the errors, if any. After the validation is successful, the interface for the mapping is shown to the translator (see Fig. 14). One or more tables referring to material modeling entities (e.g., molecules, devices, etc.) can be selected, and each input can be mapped to one attribute of one of such tables.

Once the mapping procedure is completed, the workflow is ready to be published. The publishing phase implies the automatic translation of the workflow into executable code. Such code, named a template, is then stored into the DMS together with the source BPMN workflow and a JSON file summarizing the mappings. Once the template has been created, no further knowledge of the working principles of the translation from MODA to BPMN to code is required. The execution can be requested by domain experts (without coding expertise) that based on their knowledge of material modeling and of the desired outcome of the model, can select reasonable inputs and parameters for the template. Every template can be run an indefinite amount of times. Although the execution on local computer is possible, the code typically runs on a HPC facility. Execution requests are accepted via a polling mechanism and dispatched to the HPC. Once the results are ready, they are stored into the DMS. Results can be either visualized from the DMS interface or downloaded.

Conclusions

This paper presented a low-code approach for the transformation of MODA workflows into executable multi-scale workflows. This approach, based on the use of the BPMN standard and the MuPIF integration framework, has been implemented in the MUSICODE OIPMM and demonstrated through a real-world use case.

Summarizing the procedure that leads from a set of requirements to the actual execution of a simulation, the translator is the professional who performs the majority of the steps. The translator operates for the most part on visual interfaces (i.e., those of the MUSICODE Workflow Editor) to convert the MODA workflow into BPMN, and from BPMN into executable code. The implementation of the API, alongside with the implementation of the models that are required but not yet part of the OIPMM, is the only task that demands actual programming. Thus, the low-code nature of the framework presented in this paper is confirmed, as opposed to alternatives like custom coding to link together solvers and data sources or the use of an integration framework. Once the translator's work is completed, domain experts can execute the resulting template multiple times. Such experts, e.g., the investigators who formulated the user case, need to have knowledge of material modeling discipline and the related physics principles; conversely, they are not required to possess any coding skills, or any knowledge about the whole translation process.

The proposed approach shows that BPMN can act as a bridge between a MODA workflow definition and its executable encoding, merging them into a single entity while ensuring FAIR compliance and opening the possibility for reuse and generation of additional workflow implementations. This single entity represented by a BPMN model has a formal graphical representation, a well-defined execution semantics, and is machine readable and actionable. It thus overcome several of MODA shortcomings and initiatives that tried to address them.

While the expressivity of BPMN suggests that any workflow defined with MODA can be translated to an executable BPMN process, the MUSICODE OIPMM currently supports only three MODA templates. Future research directions could focus on the analysis and support of additional MODA templates and explore the use of BPMN for the encoding of all the sections of a MODA template.

Finally, we have demonstrated the translation and implementation of executable multi-scale workflows using a very simple example relating to the OLAE domain. We note that within the MUSICODE OIPMM several high-level multi-scale workflows, which link different modeling scales from electronic (DFT) to continuum (CFD), have been created and validated with the Workflow Editor. Currently, our workflows are designed to support two main OLAE industry innovation needs: (a) modeling workflows for materials discovery (like the one demonstrated here), predicting the properties of new organic materials, doped materials, material combinations and formulations, and (b) modeling workflows for production optimization, simulating wet- and gas-phase processing and predicting the effects of processing conditions on the material properties. However, they are not limited to that; more research critical directions can be supported through the OIPMM, such as inter-layer material interactions, materials stability, defect formations, and device and module aging. We stress that such highly complex and expansive multi-scale workflows are in general extremely hard to keep track of even for expert users, not to mention traceability and reusability. We hope that the MUSICODE low-code approach for simulation workflows and executable MODAs will offer a great platform to create fully documented, traceable, and reusable workflows, which will be easily managed by domain experts even with no coding expertise. Of course, the BPMN editor is quite generic and versatile, so that other industries (besides OLAE) using organic materials can immediately benefit from the OIPMM infrastructure, such as pharmaceutical, packaging, construction, and advanced composites.

Editor's Video Summary The online version of this article (https://doi.org/10.1007/s40192-025-00395-5) contains an Editor's Video Summary, which is available to authorized users.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s40192-025-00395-5.

Funding Open access funding provided by HEAL-Link Greece. This research was part of the MUSICODE project. The MUSICODE project was funded by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 953187. Conceptional contributions are provided by the authors MS and BN through support of the Helmholtz association within the program "KNMFi, no. 43.31.01" which is gratefully acknowledged.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest. The BPMN model described in Section BPMN Representation of Workflow can be downloaded at the following link: https://doi.org/10.5281/zenodo.14283144. The MUSICODE platform can be accessed at the following link: https://musicode.grantami.com/grantami. The authors of this paper can be contacted to receive access credentials.

References

1. European Commission and Directorate-General for Research and Innovation, Baas, A (2017) What makes a material function? - Let me compute the ways - modelling in H2020 LEIT-NMBP programme materials and nanotechnology projects - sixth version. Publications Office of the European Union, Luxembourg. https://doi.org/10.2777/417118
2. CEN-CENELEC (2018) Materials modelling - terminology, classification and metadata. CEN Workshop Agreement CWA 17284
3. Horsch MT, Schembera B, Preisig HA (2023) European standardization efforts from fair toward explainable-ai-ready data documentation in materials modelling. In: 2023 3rd International conference on applied artificial intelligence (ICAPAI), pp. 1–6. https://doi.org/10.1109/ICAPAI58366.2023.10193944
4. Horsch MT, Niethammer C, Boccardo G, Carbone P, Chiacchiera S, Chiricotto M, Elliott JD, Lobaskin V, Neumann P, Schiffels P, Seaton MA, Todorov IT, Vrabec J, Cavalcanti WL (2020)

Semantic interoperability and characterization of data provenance in computational molecular engineering. J Chem Eng Data 65(3):1313–1329. https://doi.org/10.1021/acs.jced.9b00739
5. CEN-CENELEC (2022) ModGra: a graphical representation of physical process models. CEN Workshop Agreement CWA 17960
6. Patzák B, Rypl D, Kruis J (2013) Mupif - a distributed multiphysics integration tool. Adv Eng Softw 60–61:89–97. https://doi.org/10.1016/j.advengsoft.2012.09.005
7. Pizzi G, Cepellotti A, Sabatini R, Marzari N, Kozinsky B (2016) Aiida: automated interactive infrastructure and database for computational science. Comput Mater Sci 111:218–230. https://doi.org/10.1016/j.commatsci.2015.09.013
8. OMG (2013) Business process model and notation (BPMN) version 2.0.2
9. Klein P, Konchakova N, Hristova-Bogaerds DG, Noeske M, Simperler A, Goldbeck G, Höche D (2021) Translation in materials modelling - process and progress. Technical report, OntoTrance, FORCE
10. EMMC (2017) Moda - modelling data generalisation. EMMC international workshop. https://emmc.eu/wp-content/uploads/2021/05/EMMC_IntWorkshop_Vienna2017_MODA_Talk.pdf (accessed on 26/08/2024)
11. Wilkinson M, Dumontier M, Aalbersberg Iea (2016) The fair guiding principles for scientific data management and stewardship. Sci Data 3:160018. https://doi.org/10.1038/sdata.2016.18
12. Campagna D, Costanzo S, Kavka C, Turco A, Poloni C (2015) Leveraging the bpmn standard to govern engineering processes in a collaborative environment. In: 2015 IEEE international symposium on systems engineering (ISSE), pp. 318–323. https://doi.org/10.1109/SysEng.2015.7302776
13. Campagna D, Kavka C, Turco A, Pogace B, Poloni C (2016) Solving time-dependent coupled systems through fmi co-simulation and bpmn process orchestration. In: 2016 IEEE international symposium on systems engineering (ISSE), pp. 1–8. https://doi.org/10.1109/SysEng.2016.7753140

Authors and Affiliations

Dario Campagna[1] · Alan Del Piccolo[1] · Konstantinos Kaklamanis[2] · Stanislav Šulc[3] · Mattia De Bernardi[1] · Flavio Ellero[1] · Saeideh F. Kalourazi[6] · Klaus Reimann[5] · Maria Andrea[2] · Konstantinos Kordos[2] · Michael Selzer[6] · Britta Nestler[6] · Dimitrios G. Papageorgiou[2] · Aron Kneer[5] · Davide Di Stefano[4] · Bořek Patzák[3] · Elefterios Lidorikis[2]

✉ Elefterios Lidorikis
elidorik@uoi.gr

Dario Campagna
campagna@esteco.com

Alan Del Piccolo
delpiccolo@esteco.com

Konstantinos Kaklamanis
k.kaklamanis@uoi.gr

Stanislav Šulc
stanislav.sulc@fsv.cvut.cz

Mattia De Bernardi
debernardi@esteco.com

Flavio Ellero
ellero@esteco.com

Saeideh F. Kalourazi
saeideh.kalourazi@kit.edu

Klaus Reimann
k.reimann@tinnit.de

Maria Andrea
mara.andre25@gmail.com

Konstantinos Kordos
k.kordos@uoi.gr

Michael Selzer
michael.selzer@kit.edu

Britta Nestler
britta.nestler@kit.edu

Dimitrios G. Papageorgiou
dpapageo@uoi.gr

Aron Kneer
a.kneer@tinnit.de

Davide Di Stefano
davide.distefano@ansys.com

Bořek Patzák
borek.patzak@fsv.cvut.cz

[1] Research and Development Department, ESTECO SPA, 34149 Trieste, Italy

[2] Department of Materials Science and Engineering, University of Ioannina, 45110 Ioannina, Greece

[3] Department of Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague, 16000 Praha 6, Czech Republic

[4] Ansys, Cambridge CB1 7EG, UK

[5] TinniT Technologies GmbH, 76131 Karlsruhe, Germany

[6] Institute of Applied Materials-Microstructure Modelling and Simulation, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany