Delivery date: 31.12.2014

Authors:

Henrik Rusche
Wikki
E-mail : h.rusche@wikki.co.uk

Heinz Preisig
NTNU
E-mail :
heinz.preisig@chemeng.ntnu.no

MoDeNa

Deliverable 5.2 Orchestrator v0: First release of orchestrator, database structure and interface library for the MoDeNa software

WP's leader: Wikki

Principal investigators:
Henrik Rusche, **Wikki** (UK)

Project's coordinator:
Heinz Preisig, **NTNU** (N)

1. Description of deliverable

An initial implementation of the software framework has been carried out. The software is based entirely on open-source software. A set of initial adaptors and recipes and formed a suitable basis for the software development tasks planned within MoDeNa (task 5.1).

2. Summary of contribution of involved partners

Wikki carried out the software implementation of the software framework. Initial adaptors and recopies have been submitted by VSCHT, POLITO, TUE, US, UNITS. VSCHT provided the Fortran module definition.

3. Introduction

MoDeNa aims at developing, demonstrating and assessing an easy-to-use multi-scale software-modelling framework application under an open-source licensing scheme that delivers models with feasible computational loads for process and product design of complex materials. The use of the software will lead to novel research and development avenues that fundamentally improve the properties of these nanomaterials.

The concept of MoDeNa is an interconnected multi-scale modelling-software framework. Four scales are linked together by this framework namely the nano-, micro-, meso-, and macroscale. This unifying software framework will allow for enhanced product- and process design across these scales. As is shown in Figure 1, the modelling framework is intimately coupled with the software framework. In this project, the software framework will facilitate and greatly enhance the modelling activities. The orchestrator enables the linking of all scales which is a necessary condition to obtain an integral approach, in contrast to a series of disconnected phases.

The orchestrator is the backbone of our software suite in that it logically links the specific task-solvers (or applications). These application-specialised codes mostly existing already and will be connected across the scales. The orchestration software is very much like a work-flow generation interface, in which the different task-solvers acting at each single scale are ‘orchestrated’ by the framework. The orchestrator calls the external software and obtains the necessary information, which is analysed, pre-processed and passed to the next task-solver through a suitable protocol.

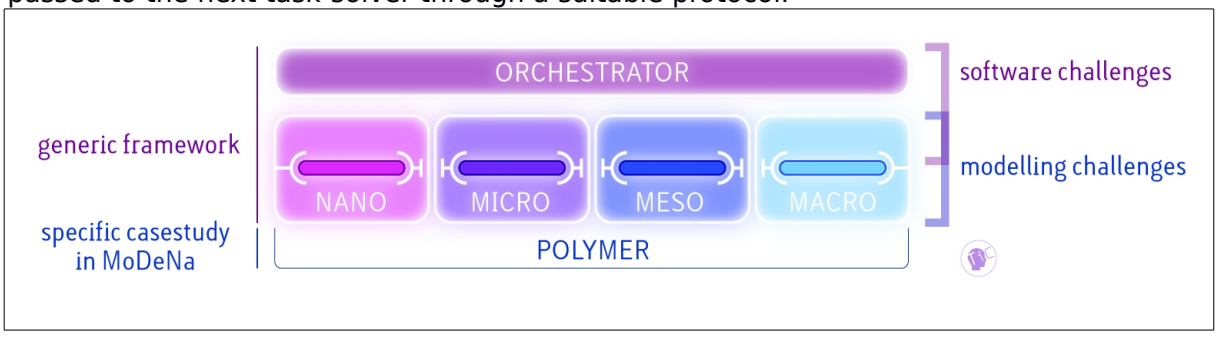

Figure 1: Conceptual structure of MoDeNa.

This coupling will allow for the application to product and process design as well as the integration of the various computational tasks (see Figure 2) through linking of models, associated parameters, data, and the production process of a multi-scale material can be accurately simulated.

Within the project an open-source software-suite is constructed that logically interlinks scale and problem specific software of our university groups, using a software orchestrator that communicates information utilizing our proposed new communication standard in both directions, namely upwards to the higher scale and downwards to the lower scale. This feature is unique, enabling the solution of complex material design problems.

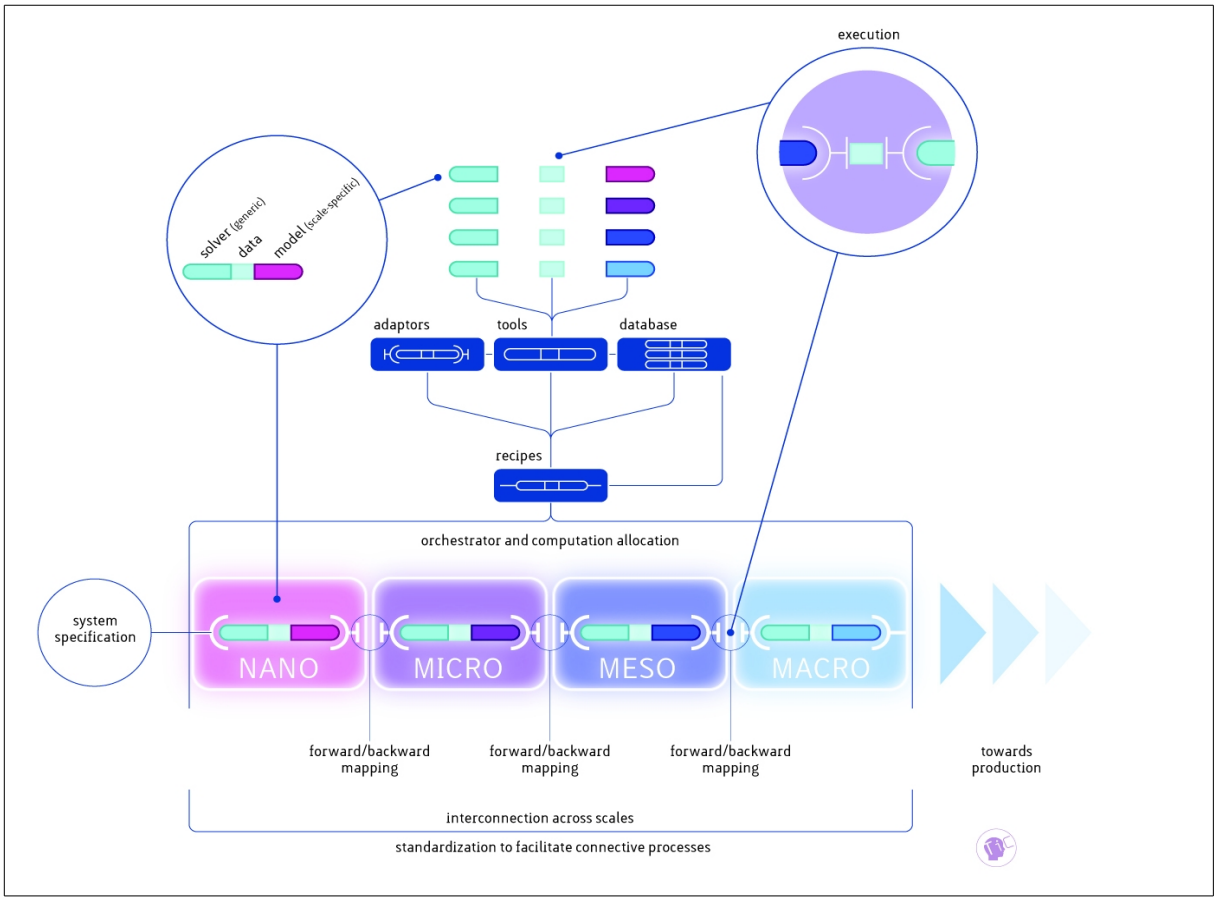

Figure 2: Conceptual structure of MoDeNa, coupling solvers and models into tools, which form sequences through recipes and the orchestrator. The sequence from nano-scale to macro-scale signifies the range of scales.

Multi-scale coupling as proposed in MoDeNa requires the exchange of information between software instances developed for specific scales in a consistent way. In order to achieve this, generating consistent representations for models and data that is based on a solid theoretical framework is necessary. The information exchange is governed by protocols and may occur in two ways, namely:

- 'forward mapping' (passing information from the fine to the coarse scale, upward direction)
- backward mapping' (passing information from the coarse to the fine scale, downward direction)

'Forward mapping' is relatively straightforward, while *'backward mapping'* inevitably requires iteration since changing the operating conditions at the fine level changes the feedback to the coarse level. 'Backward mapping' can be realised in two ways: *'Two-way coupling'* and *'model fitting'* through a sequence of model based design of experiments and parameter estimation. The first approach usually requires exchange of large amounts of data during runtime that may be expensive either due to the complexity of the data exchange or the computational cost associated with executing the fine scale software. In such cases, surrogate models, which are 'parameter fitted models' presents the only viable alternative.

The software framework under development combines complex features and IT infrastructure such as job scheduling, distributed computing, code coupling, model database, (model based) design of experiments and parameter estimation in a unique

and unified way. It is unrealistic to develop all of these features from scratch within in the time and budget constraints. Hence the development must start from already existing software.

In the next sections we present the software evaluation for each component.

4. Interaction of the Software Components

The interaction of the software components is shown in Figure 3 for a backward mapping. The execution of the macroscopic and microscopic codes is depicted by the white arrows on the right and left hand side while parameter fitting and design of experiments take place within the arrows on the top and bottom. The first two operations are handled by user of the software framework through suitable recipes and scale-specific codes while the latter operations are handled by the MoDeNa software framework. The green and red arrows stand for communication interfaces with the central database.

In the MoDeNa framework, all scale interactions are handled through surrogate models which are invoked within in the macroscopic code. Respective code blocks for

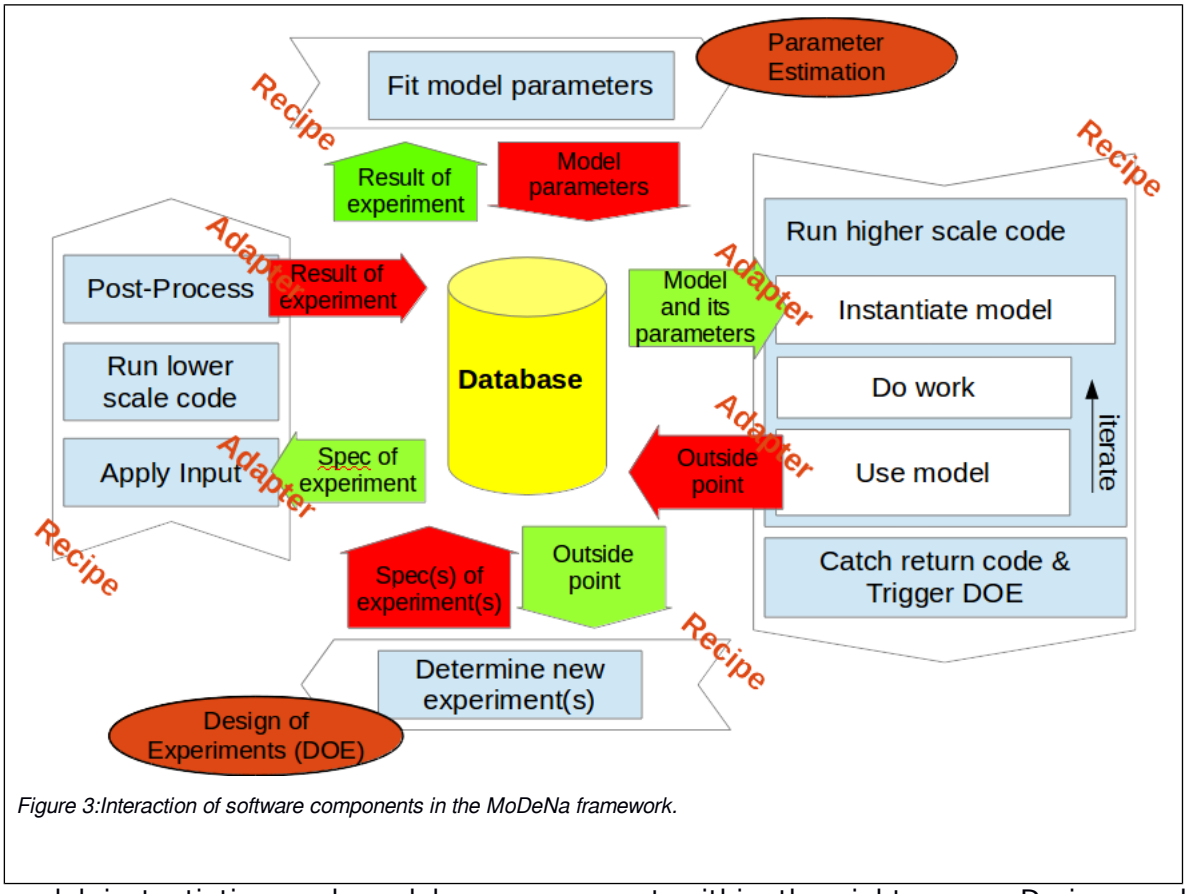

Figure 3:Interaction of software components in the MoDeNa framework.

model instantiation and model usage are put within the right arrow. During model instantiation, all necessary data such as model parameters is obtained from the database. After instantiation, the model can be called and its results are used within the macroscopic code. During invocation of the surrogate model, the inputs given to the model are checked against the range of validity recorded in the database. If the model is used out of bounds, the violating point (outside point) is then recorded in the database and an exception is raised which must then be handled appropriately by the embedding codes and recipes. In the simplest case, violation of the model bounds will result in a controlled termination of macroscopic code and invocation of the design of experiments module. This module then determines a set of new experiments to be executed on the microscopic level. After the microscopic recipe is run for each experiment, a new set of model parameters is obtained by the parameter fitting module and the macroscopic code is restarted. Here, different strategies are possible which will be subject to future research.

It should be noted that a microscopic code may act as a macroscopic code by embedding a surrogate for a yet another, even lower scale. This scenario will be supported in future version of the software framework.

It is evident that the order of execution of the recipes changes dynamically and that the generation of multiple points by the design of experiments module offers potential for parallelism. This complexity is handled by the orchestrator.

5. Software Stack

The software stack is shown in Figure 4. The colours symbolise computer languages used in the implementation. Since orchestration of the recipes is handled through FireWorks (blue box) all recipes are implemented as FireTasks (red boxes). FireTasks are derived python classes which either invoke an executable/external (shell) script or directly call functionality coded in python. The first method will typically be used

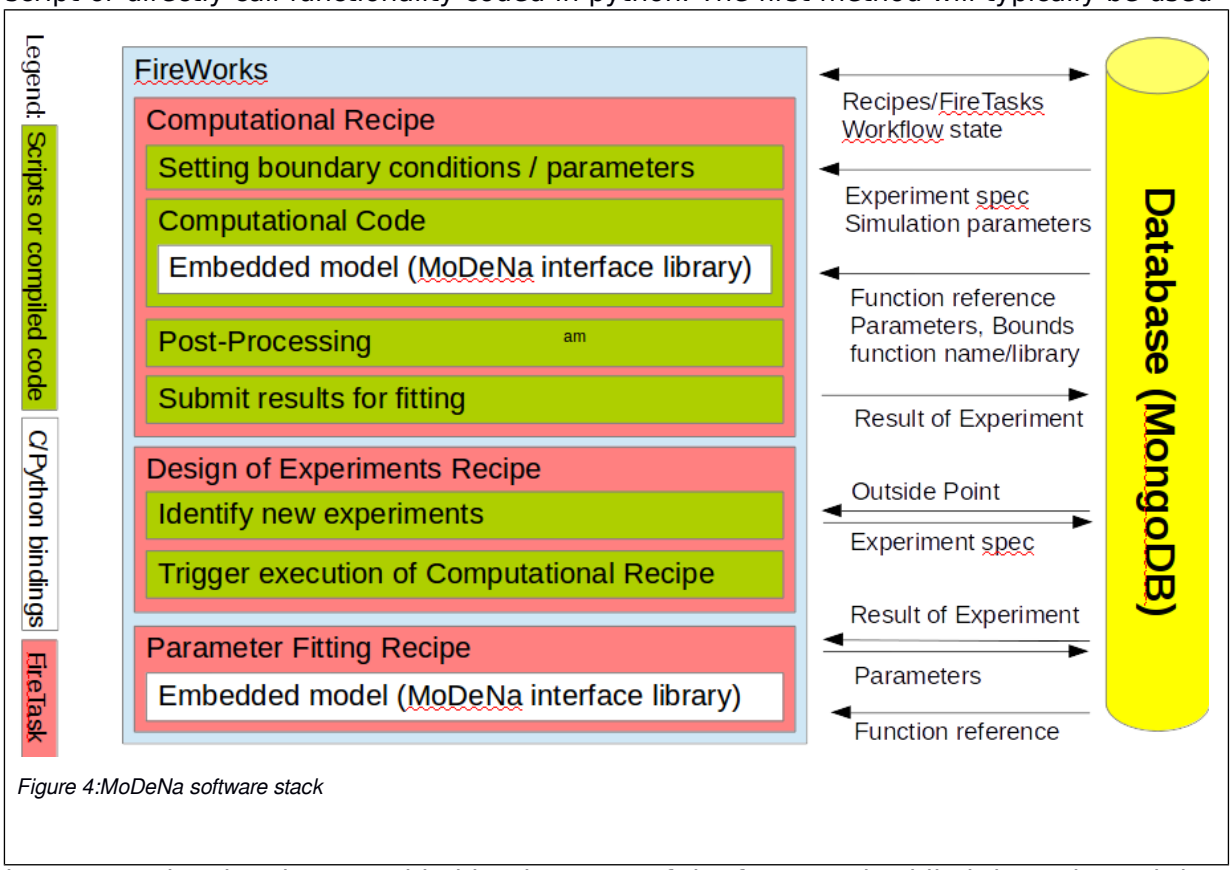

Figure 4:MoDeNa software stack

in computational recipes provided by the users of the framework while it is envisaged the parameter fitting and design of experiment modules are coded in python. Embedded surrogate models appear in two places: within the computational codes and parameter fitting while the MoDeNa interface library ensures that the correct models are used in both. This core component is written in pure C. Language interoperability is ensured by SWIG python wrappers and a Fortran 90 module definition. Consequently, the surrogate models implemented in MoDeNa can be called from C, C++, Fortran 90 and python.

6. Two Tank Tutorial

This tutorial has been created to show the features of the MoDeNa software framework for a trivial example. The problem modelled is that of the discharge of air from one tank into another (macroscopic problem) through a nozzle (microscopic problem). Depending on the flow conditions and shape of the nozzle, the nozzle flow may be difficult to compute and require a full 3D CFD calculation. In the tutorial, this calculation is replaced by an engineering correlation. Assuming that the range of inputs is unknown a-priori, this is a prototypical backward mapping problem. *twoTanksMacroscopicProblem.C* uses the MoDeNa interface library to embed an even simpler model for the flow rate.

While *twoTanksFullProblem.C* implements it fully integrated. The latter executable is used solely as a reference. This prototype application of software framework has been initially distributed to project partners. The tutorial is intentionally kept to the bare minimum in order to reduce runtime and dependencies (e.g. no external codes).

A part of the source code of *twoTanksMacroscopicProblem.C* is printed in Figure 5. The code snippet creates a connection to the MongoDB database and then instantiates a surrogate model. After determining the positions of the input arguments, the input vector is filled and the model is called. Depending on the return value, the program is terminated after appropriate clean-up. At this point, additional actions may be required in order to allow restarts from the current state.

```
// Open connection to database
mongoc_client_t *client = mongoc_client_new ("mongodb://localhost:27017/");

// Instantiate a model
modena_model_t *model = modena_model_new (client, "dummy", "flowRate", "dummy");

// Allocate memory and fetch arg positions
modena_inputs_t *inputs = modena_inputs_new (model);
modena_outputs_t *outputs = modena_outputs_new (model);

size_t Dpos = modena_model_inputs_argPos(model, "D");
size_t rho0Pos = modena_model_inputs_argPos(model, "rho0");
size_t p0Pos = modena_model_inputs_argPos(model, "p0");
size_t p1Pos = modena_model_inputs_argPos(model, "p1");

while(t + deltat < tend + 1e-10)
{
    t += deltat;

    if(p0 > p1)
    {
        // Set input vector
        modena_inputs_set(inputs, Dpos, D);
        modena_inputs_set(inputs, rho0Pos, rho0);
        modena_inputs_set(inputs, p0Pos, p0);
        modena_inputs_set(inputs, p1Pos, p1);

        // Call the model
        int ret = modena_model_call (model, inputs, outputs);

        // Terminate, if requested
        if(ret != 0)
        {
            modena_inputs_destroy (inputs);
            modena_outputs_destroy (outputs);
            modena_model_destroy (model);
            mongoc_client_destroy (client);

            return ret;
        }

        // Fetch result
        double mdot = modena_outputs_get(outputs, 0);

        // Update states of the tanks
        m0 -= mdot*deltat;
        m1 += mdot*deltat;
    }
    else
    {
        // Symmetric code, omitted for brevity
    }
}
```

Figure 5: twoTanksMacroscopicProblem.C

7. Current Features

- Optional use of the interface: This feature allows models to be used from C, C++, Fortran90 or python code.
- Automatic compilation of models: This feature allows storage of C-code in the database and automatically compiles the code and links it into the respective FireTask.
- DOE fitting and parameter fitting: These features are subject to deliverable 5.3 and form a core structure in the MoDeNa orchestrator. Their implementation allows for the "daisy-chaining" of the scales as described in D5.3. Combined with a state definition of the tasks and an event handling these modules will form the core of the workflow handling.
- Platform independent make-system (CMake).
- Library compiles on Linux and Mac OS X.
- 10 mappings for PU foam and TPUs have been implemented and submitted:
    - acoustics (VSCHT),
    - bubbleGrowth (VSCHT),
    - thermalConductivity (VSCHT),
    - CFD_tool (POLITO),
    - coalescenceKernel_Tool (TUE),
    - rheology_Tool (TUE),
    - DFT_Tool (US),
    - EOS_Tool_PolymerDensity (US),
    - EOS_Tool_Solubilities (US),
    - density (UNITS)

8. Development Status

The consortium members are in the process of implementing the recipes and adaptors for their respective parts of the demonstration problems. The scope of the problems is purposely kept limited so as to learn about the problems that arise from the user's side. The current approach is that an application starts with the existing primary example and extends it to the desired scope. This then forms a package which is aimed to evolve into one component in the overall workflow.

10 mappings have so far been submitted and we also are working on an extended tutorial example that will demonstrate more of the features mainly with respect to the splitting of the DOE and model fitting facilities. As mentioned in D5.2, the target is to increase the flexibility of the DOE and fitting component by using the statistical programming language R as the basis. This will enable us to generate recipes in R, which is an extremely rich environment for statistical data analysis.

9. Future Work

Future work will focus on testing and applying the current framework to the PU foams and TPUs. In the process, we will improve the usability of the library based on the comments and suggestions which we receive from the user base. This will include a class hierarchy for models and recipes such that only a minimum of coding is required by the user for typical application while the framework is kept open for future extensions and specialisations. Further improvements will be done to the database storage layout to store fitting data and models separately as well as to store the model dependency in scenarios where surrogate models are used within the microscopic code.

Options for surrogate model import, for example, Automatic generation of C-code from unified model description and semi-automatic wrapper generation for polymerisation kinetics as produced by Predici.

The generalisation and flexibility of the DOE/parameter fitting are under development. Two larger problems have been defined as next targets for the implementation of PU-relevant workflows. In crude terms the first represents the interaction and agglomeration of the lower scales, whilst the second is aiming at the

larger scales and the integration of surrogate models for the lower scales. This development is planned as a collaboration of WIKKI and NTNU.

Global state of the computational units and the centralised event handling are at a high priority as they will be essential for the "daisy-chaining" of the scale-related computational tasks.