Developing Software Tools Facilitating Interdisciplinary Modelling

Sigve Karolius[1] Heinz Preisig[1]

[1]Chemical Engineering, Norwegian University of Science and Technology, Norway, sigve.karolius@ntnu.no, Heinz.Preisig@chemeng.ntnu.no

Abstract

Computer-aided modelling has focused on developing domain-specific frameworks. Despite being powerful stand-alone tools they can be challenging to incorporate into a multi-scale model whose inherent interdisciplinary nature leads to a heterogeneous set of languages and tools. However, tools and models can be made easier to adopt into future projects by making conscientious development choices based on software development techniques. This paper shares the experiences and software design options that were considered and employed in the development of the MoDeNa multi-scale modelling framework.
Keywords: multi-scale simulations, software design

1 Introduction

The focus on interdisciplinary collaboration is likely to increase within the computer-aided modelling community as the demand for simulation-guided design and discovery requires increasingly accurate predictive models. The trend is already clear within material-science where both the European Materials Modelling Council (EMMC) and National Aeronautics and Space Administration (NASA) have formulated long-term strategies aimed at developing solutions facilitating software interoperability for domain-specific simulation platforms, as can be seen in the technical report (Liu et al., 2018).

This is obviously not the first time that the need for standardisation has been realised and acted upon. One example from chemical engineering is the CAPE-OPEN project presented in (Belaud and Pons, 2002). The goal of the project was to create a common interface for process models to facilitate interoperability between well entrenched proprietary simulation environments, such as the Advanced System for Process Engineering (ASPEN) outlined in (Evans et al., 1979), as well as developing third-party extensions. Another example is the Modelica project, described in (Erik Mattsson et al., 1998), which aimed to provide a generic language for developing physics-based models in any domain.

The challenge in the integrated materials computational engineering (ICME) domain is that the models requires intertwining descriptions of systems on multiple scales and are therefore based on different physical principles. The project Morphology Development of Micro- and Nano-structures (MoDeNa) was one effort aimed at creating a platform coupling models together, but used simplified versions in-place of the detailed models. The project was a major software-development effort that involved implementing the software framework in collaboration with model-developers. This paper describes software design principles that were used in order to achieve coupling between a large number of models presented in (Karolius et al., 2017), as well as extensions that would have to be made in order for the framework to also support developing and simulating fully coupled models.

2 Constituent Parts and Coupling

All modelling activities involve describing systems and the relationship between them. In the multi-scale simulation domain the focus is mainly on coupling models that describe physical systems on vastly different scales as illustrated in Figure 1.

In order to develop tools that support the development of multi-scale systems it is necessary to identify the relationships between the elements that make up the overall multi-scale model as well as the type of coupling that must be supported. The project Morphology Development of Micro- and Nano-structures (MoDeNa) took aim at one area of multi-scale simulation, namely sequentially coupled models.

2.1 Ontology

The *ontology* provides a formal description of the constituent parts and the relationship between them. This is currently one of the major efforts of the materials modelling experts in the EMMC , but the topic is still relevant for the software design.

The work by (Yang and Marquardt, 2009) provides an extensive abstract ontology for multi-scale models from an engineering perspective. It could be of particular interest to study the overlap between ontologies from different domains and attempt to create a correlation between them for the purpose of facilitating communication between ontologies.

An alternative approach to defining an ontology and demanding developers to adhere to it is to develop a graphical modelling environment, such as (Elve and Preisig, 2017), that strictly adheres to a pre-defined ontology. Consequently, the model developer has no choice but adhering to the ontology.

Considering that the relationships between the entities in the ontology is directly related to scale-bridging in a multi-scale model the approach could be of great aid in de-

signing a software framework capable of handling a wide variety of scale-coupling strategies.

2.2 Scale-bridging Strategies

There are several approaches for coupling the scale-specific simulation models in a multi-scale model, both rigorous theoretical frameworks and practical approaches can be found in recent literature such as (Weinan, 2011).

2.2.1 Fully Coupled Models

The overall multi-scale model can, seen from the top-down perspective, be described as an onion. At the engineering-level one is therefore always making implicit assumptions about the scales below. Bearing in mind that

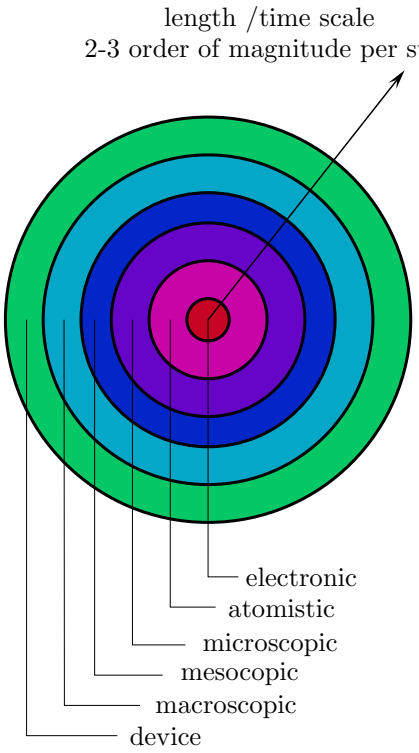

Figure 1. Illustration of a multi-scale model as the layers of an onion, there each layer represents a modelling domain.

continuum-based models have been successfully used for decades it could be tempting to question the need for the scale-coupling illustrated in Figure 2. However, the scale-

Figure 2. Illustration of a model in which the dynamics of the scale-specific models overlap. Th multi-scale description consequently requires that the models are fully coupled.

coupling is often important, such as when considering with surface effects in low-density systems, such as rarefied gases (Docherty et al., 2014). Moreover, when the coupling does occur it quickly becomes a dominating factor that characterises the dynamics of the system.

Fully coupling models is therefore necessary in order to capture physical phenomena that arises as a direct consequence of the multi-scale nature of the system. However, the computational cost associated with running concurrent simulations limits the number of models that can be integrated in this manner before the computational effort outweighs the benefit.

2.2.2 Scale-Separated Models

The scale-separation assumption is an argument that decouples the dynamics of the individual scale-specific models as illustrated in Figure 4. This is the assumption on which the MoDeNa software framework was based. The use of surrogate-models in the MoDeNa project is shown in Figure 3, they are used in-place of the detailed model for the lower scale. In this way the scale-separation is

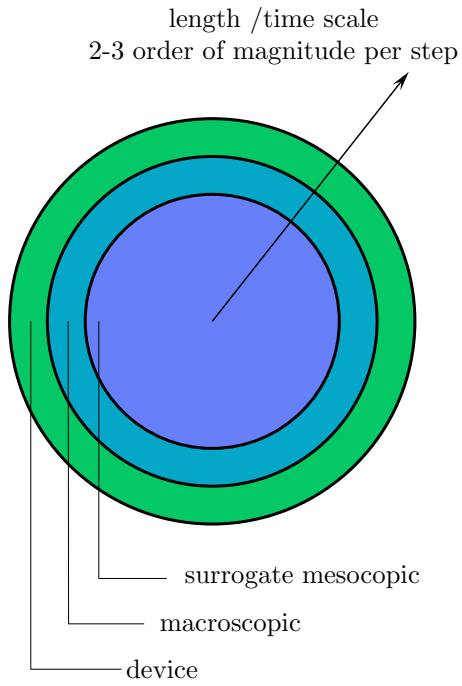

Figure 3. Illustration of a multi-scale model as the layers of an onion, there each layer represents a modelling domain.

exploited, typically for the purpose of speeding up the higher scale model. As mentioned in the previous section, scale separation is very common and the use of low-level models to calculate material properties for exotic alloys is highly interesting for engineers. In contrast to the fully

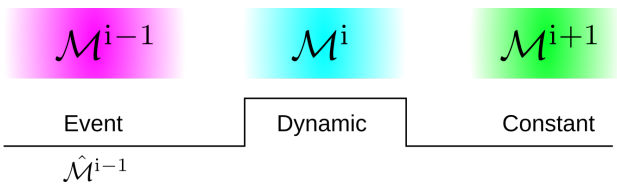

Figure 4. Illustration of the relationship between the dynamics of a model on a specific scale to that of the scale below and above.

coupled models in the previous section one can use a lot

of surrogate-based scale-separated models as long as they are never used outside of the domain in which the parameters were validated.

3 Software Design Concepts

The process of developing generic software is intrinsically iterative and different modelling paradigms often have significantly different requirements. However, it can be considerably simplified by making conscious decisions that clearly separate individual components of the software and define to what extent it supports generic concepts.

3.1 Abstraction

For the past two decades so-called *object-oriented*, or data-centric, design has been the prevailing principle for software development. The earliest formal representation of an object-oriented paradigm is often accredited to the SIMULA languages, whose history is presented in (Nygaard and Dahl, 1981), which was designed to facilitate development of discrete event systems (DESs).

Nowadays, most modern programming languages support object-oriented (OO) programming to a greater or lesser extent. It is common for modelling frameworks to take advantage of the abstraction capabilities that object-orientation provides, which in practice means to design well-defined generic implementations that are capable of describing special cases of an appropriate type. The advantage often lies in that multiple solution strategies can be defined for the generic description and any of them can be imposed on a particular problem.

The Open Field Operation and Manipulation (OpenFOAM) project founded by (Weller et al., 1998) is an excellent example of the power of OO design principles. At its core the OpenFOAM framework is a generic implementation of finite volume method (FVM) and provides abstract data-types designed specifically for the purpose of making the computer implementation of partial differential equations (PDEs) mimic mathematical notation. The listing 1 shows an excerpt from the source code for the OpenFOAM solver "`laplacianFoam`", an algorithm for solving the heat equation.

Listing 1. A slightly modified excerpt from the source code of the `laplacianFoam` solver in the OpenFOAM project. The code block is the entire functional part of the main loop of the solver and illustrates how well-designed abstraction of operators and data-containers simplifies the implementation.

```
while ( simple.loop() ) {
  while ( simple.correctNonOrthogonal() ) {
    solve(
      fvm::ddt(T) - fvm::laplacian(DT, T)
    );
  }
}
```

When comparing the code-block with the colour coding in equation 1 the resemblance is striking.

$$\frac{\partial T}{\partial t} - \nabla \left[D \nabla T \right] = 0 \tag{1}$$

However, even though the abstraction of the differential operators makes the overall implementation look deceptively simple, there is obviously a lot of software engineering taking place in the background.

This raises the question of what happens to the terse implementation if the solver should be modified to include temperature dependency of the thermal diffusivity $D(T)$ by employing a simulation-model from a lower scale, e.g. using molecular dynamics (MD) and the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) described in (Plimpton, 1995).

There is effectively two possible choices at this point:

Keep the original abstraction
: In the case of OpenFOAM it is possible to implement a specialised thermo-physical model which handles the coupling with the lower-scale model; thus, the original solver remains largely unchanged.

Modify the solver
: The easiest strategy is to implement the coupling to the MoDeNa framework directly in the main loop of the solver. This will render the original implementation virtually unrecognisable, but also clearly show the coupling.

There are positive and negative aspects to both strategies. As an example consider the sample code for implementing the MoDeNa software framework directly into the `laplacianFoam` solver.

Listing 2. Excerpt from the source code of the `modenaLaplacianFoam` solver. The main loop effectively split into two parts where one is a loop that requests information about the thermal diffusivity from the MoDeNa software framework before continuing to the solver equivalent to the original *laplacianFoam* implementation in Listing 1

```
while ( simple.loop() ) {
  while (simple.correctNonOrthogonal()) {
    try { // ----------------- MoDeNa Block
      forAll(T, celli) {
        DT[celli] = model(Tpos, T[celli]);
      }
    }catch(const modenaException& e){
      return e.errorCode();
    } // ---------------------- Call Solver
    solve(
      fvm::ddt(T) - fvm::laplacian(DT, T)
    );
  }
}
```

When compared to the original solver in Listing 1 the modified implementation in Listing 2 has major changes. However, the changes are fairly straight forward to implement for someone who is somewhat familiar with the programming language. Moreover, since the implementation

interrupts the main loop it would be possible to perform synchronization with an external application; thus, potentially allowing for concurrent computational fluid dynamics (CFD) and MD simulations that exchange data on-the-fly.

3.2 Modularity

In the overall picture of a simulation that employs multiple models it is intuitive to consider each individual simulation-model as a software module. The module is commonly further divided into components that perform independent tasks, such as pre/post processing or providing abstraction, which is the case for the operators in the example in Listing 1.

3.2.1 Coupling

In general the term *coupling* refers to the relationship between software modules in a project. The idea motivating the focus on coupling in software design is that internal changes within one module should affect other modules to the least extent possible.

Loose coupling

Loose coupling refers to a situation where there is little interaction between the modules that makes up the software. Loosely coupled models can for instance be running a simulation in order to generate initial conditions for a subsequent simulation.

Tight coupling

Modules that are tightly coupled are highly intertwined and the connection may require in-depth knowledge of the internal components of a different module. Simulation models that requires run-time exchange of information, maybe even synchronised execution, are tightly coupled.

For the purpose of maintainability it is desirable to strive towards achieving loose coupling, illustrated in Figure 5, between modules. Loose coupling simplifies both development and simulation of models compared to tightly coupled models illustrated in 6. In the case of interdisciplinary modelling efforts tight coupling is particularly problematic. The reason for this is that it is important that

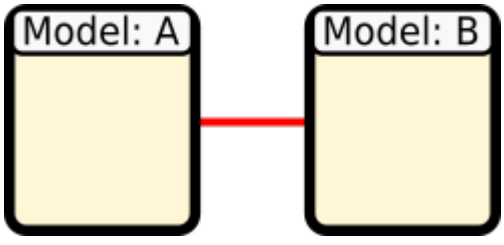

Figure 5. Illustration of two loosely coupled models where the interface effectively decouples the inner-workings of each model. This makes it possible for one model to make changes to internal components as-necessary without requiring the other model to change.

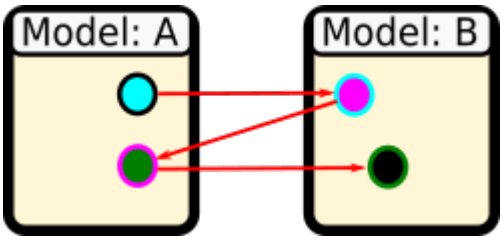

Figure 6. Illustration of tight coupling between two models where both models need some information about individual components of the other. Consequently, internal changes in one model may cause the other model to adapt accordingly.

models can be tested and validated in relative isolation, i.e. without connections to other models.

However, the question still remains with how to deal with models that are not loosely coupled by nature and requires exchange of information at run-time. The strategy here is to employ an external framework, such as the MoDeNa software framework presented in (Karolius et al., 2016), in order to decouple the connection between the individual models as shown in Figure 7.

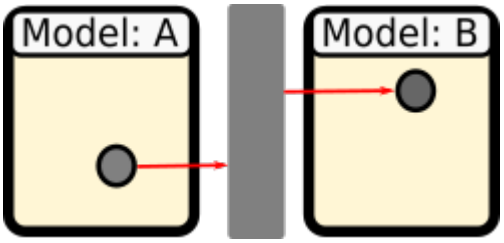

Figure 7. Illustration of a situation where two models that are inherently coupled, but the use of a external framework is used for the purpose of decoupling the connection.

3.2.2 Cohesion

Cohesion is an intra-module concept and refers to the relationship between the internal components of a module. It is a measure of the degree to which the internal components of a module focuses on a single task.

It is common to refer to the degree of cohesion within a module as *low* or *high*.

Low cohesion

Low cohesion effectively implies that the implementation of the software module, or simulation model, is disorganised. In a module that suffers from low cohesion each module may try to perform several unrelated tasks and still require importing functionality from other components in the module.

High cohesion

Models that have high cohesion are organised into components that focus on specific tasks with little or no interaction with the other components in the module.

Since cohesion is an intra-module concept the practical importance of cohesion becomes less relevant if the coupling between modules is loose. However, as comparing the illustrations of modules with high and low cohesion, in Figure 8 and respectively, it can be argued that the former will lead to fewer sources for errors and less maintenance. In contrast to coupling there is not much that an external

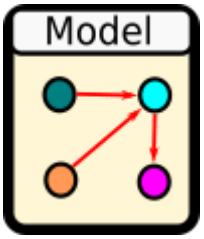

Figure 8. Illustration of a module with high cohesion.

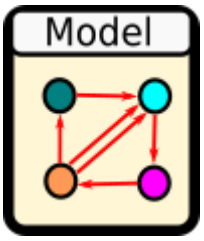

Figure 9. Module whose functionality is implemented in multiple components that require cross- and circular-imports in the module.

framework can do in order to enforce cohesion within a simulation model. However, it is possible to promote cohesion by making it possible to pre-define a workflow that should be executed when running the application.

3.3 Workflows

Most scientific simulations follow a pre-defined workflow: *initialisation*, *execution* and *post-processing*. However, for the purpose of exploring a large design space for simulation-guided studies it necessary to automate the execution and tracking of a large number of simulation runs.

In its simplest form a computational workflow consists of *atomic*, i.e. non-interruptable, tasks. Software tools, such as FireWorks (Jain et al., 2015) and are treats the workflow as a directed acyclic graph (DAG) is designed to execute whatever it can until there are no more tasks left in the queue.

3.3.1 Adaptivity and Distributed Tasks

In order to make workflows that can be employed in a machine-learning concept it is necessary to incorporate some adaptivity into the execution of the computational workflow. In practice this idea is rather simple: make the simulation-models capable of telling the workflow generator what to do in a particular context.

The MoDeNa software framework used the features of FireWorks to implement an adaptive workflow that would perform run-time parameter estimation and validation of using model-based design of experiments (Franceschini and Macchietto, 2008).

Some simulations are simply too complex to be executed in a normal desktop environment and requires the workflow to migrate to facilities capable of running applications that require heavy computing.

3.3.2 Concurrency

As long as simulation tasks are *atomic* concurrency is not a big issue and parallel execution is easily facilitated using DAG s. However, when simulations have to exchange information while they are running there is a problem, namely how to ensure that the processes are synchronised.

One approach could be to employ the message passing interface (MPI) toolkit, which is a standardised way of handling inter-process communication. However, with this approach one could end up attempting to run multiple MPI processes inside the main MPI process. An alternative way of modelling concurrency is to use Petri-nets (Peterson, 1977), as illustrated in Figure 10 A basic model

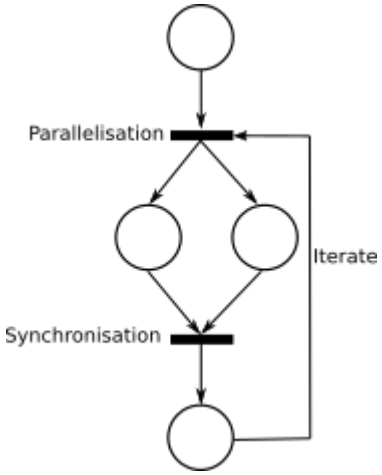

Figure 10. Simple illustration of a naive concurrency model for two models that runs in parallel and synchronises between every iteration allowing for information to be exchanged between the models.

for concurrency would not require implementing complex MPI-style communication mechanisms.

4 Conclusion

Designing a generic software framework that facilitates development of multi-scale models requires making pragmatic choices in the software design. The main focus of the framework is to support the modelling and simulation effort, not necessarily providing framework that uses abstraction techniques to hide the implementation from the users.

Instead, the design should aim at enforcing modularity of the scale-specific models and enabling flexible and distributed computational workflows. There are also significant lessons to be learned from

the use of *semantic interoperability* within web-development as well as the *gene ontology* project.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° [604271].

References

Jean-Pierre Belaud and Michel Pons. Open software architecture for process simulation: The current status of cape-open standard. In Johan Grievink and Jan van Schijndel, editors, *European Symposium on Computer Aided Process Engineering-12*, volume 10 of *Computer Aided Chemical Engineering*, pages 847 – 852. Elsevier, 2002. doi:https://doi.org/10.1016/S1570-7946(02)80169-9. URL http://www.sciencedirect.com/science/article/pii/S1570794602801699.

Stephanie Y. Docherty, Matthew K. Borg, Duncan A. Lockerby, and Jason M. Reese. Multiscale simulation of heat transfer in a rarefied gas. *International Journal of Heat and Fluid Flow*, 50:114 – 125, 2014. ISSN 0142-727X. doi:http://dx.doi.org/10.1016/j.ijheatfluidflow.2014.06.003. URL http://www.sciencedirect.com/science/article/pii/S0142727X14000836.

Arne Tobias Elve and Heinz A. Preisig. From ontology to executable program code. In Antonio Espuña, Moisés Graells, and Luis Puigjaner, editors, *27th European Symposium on Computer Aided Process Engineering*, volume 40 of *Computer Aided Chemical Engineering*, pages 2317 – 2322. Elsevier, 2017. doi:https://doi.org/10.1016/B978-0-444-63965-3.50388-3. URL http://www.sciencedirect.com/science/article/pii/B9780444639653503883.

Sven Erik Mattsson, Hilding Elmqvist, and Dynasim Ab. Modelica - an international effort to design the next generation modeling language. 30, 01 1998.

L.B. Evans, J.F. Boston, H.I. Britt, P.W. Gallier, P.K. Gupta, B. Joseph, V. Mahalec, E. Ng, W.D. Seider, and H. Yagi. Aspen: An advanced system for process engineering. *Computers & Chemical Engineering*, 3(1):319 – 327, 1979. ISSN 0098-1354. doi:https://doi.org/10.1016/0098-1354(79)80053-8. URL http://www.sciencedirect.com/science/article/pii/0098135479800538.

Gaia Franceschini and Sandro Macchietto. Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, 63(19):4846 – 4872, 2008. ISSN 0009-2509. doi:http://dx.doi.org/10.1016/j.ces.2007.11.034. URL http://www.sciencedirect.com/science/article/pii/S0009250907008871. Model-Based Experimental Analysis.

Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015. ISSN 1532-0634. doi:10.1002/cpe.3505. URL http://dx.doi.org/10.1002/cpe.3505. CPE-14-0307.R2.

Sigve Karolius, Heinz A. Preisig, and Henrik Rusche. Multiscale modelling software framework facilitating simulation of interconnected scales using surrogate-models. In Zdravko Kravanja and Miloš Bogataj, editors, *26th European Symposium on Computer Aided Process Engineering*, volume 38 of *Computer Aided Chemical Engineering*, pages 463 – 468. Elsevier, 2016. doi:http://dx.doi.org/10.1016/B978-0-444-63428-3.50082-5.

Sigve Karolius, Heinz A. Preisig, and Henrik Rusche. Sequential multi-scale modelling concepts applied to the polyurethane foaming process. In Antonio EspuÃśa, MoisÃĺs Graells, and Luis Puigjaner, editors, *27th European Symposium on Computer Aided Process Engineering*, volume 40 of *Computer Aided Chemical Engineering*, pages 487 – 492. Elsevier, 2017. doi:https://doi.org/10.1016/B978-0-444-63965-3.50083-0. URL http://www.sciencedirect.com/science/article/pii/B9780444639653500830.

Xuan Liu, David Furrer, Jared Kosters, and Jack Holmes. Vision 2040: A roadmap for integrated, multiscale modeling and simulation of materials and systems. Technical Report 20180002010, National Aeronautics and Space Administration (NASA), 03 2018.

Kristen Nygaard and Ole-Johan Dahl. The development of the simula languages. In Richard L. Wexelblat, editor, *History of Programming Languages I*, pages 439–480. ACM, New York, NY, USA, 1981. ISBN 0-12-745040-8. doi:10.1145/800025.1198392. URL http://doi.acm.org/10.1145/800025.1198392.

James L. Peterson. Petri nets. *ACM Computing Surveys*, 9:223–252, 1977.

Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. ISSN 0021-9991. doi:http://dx.doi.org/10.1006/jcph.1995.1039. URL http://www.sciencedirect.com/science/article/pii/S002199918571039X.

E. Weinan. *Principles of Multiscale Modeling*. Cambridge University Press, 1 edition, 2011. ISBN 978-1-1070-9654-7.

H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput. Phys.*, 12(6):620–631, November 1998. ISSN 0894-1866. doi:10.1063/1.168744. URL http://dx.doi.org/10.1063/1.168744.

Aidong Yang and Wolfgang Marquardt. An ontological conceptualization of multiscale models. *Computers & Chemical Engineering*, 33(4):822 – 837, 2009. ISSN 0098-1354. doi:https://doi.org/10.1016/j.compchemeng.2008.11.015. URL http://www.sciencedirect.com/science/article/pii/S0098135408002524.