Knowledge Amalgamation for Computational Science and Engineering

Theresa Pollinger, Michael Kohlhase, and Harald Köstler

Computer Science, FAU Erlangen-Nürnberg

Abstract. This paper addresses a *knowledge gap* that is commonly encountered in computational science and engineering: To set up a simulation, we need to combine domain knowledge (usually in terms of physical principles), model knowledge (e.g. about suitable partial differential equations) with simulation (i.e. numerics/computing) knowledge. In current practice, this is resolved by intense collaboration between experts, which incurs non-trivial translation and communication overheads.
We propose an alternate solution, based on *mathematical knowledge management (MKM)* techniques, specifically *theory graphs* and *active documents*: Given a theory graph representation of the domain, model, and background mathematics, we can derive a targeted knowledge acquisition dialogue that supports the formalization of domain knowledge, combines it with simulation knowledge and – in the end – drives a simulation run – a process we call MoSIS ("Models-to-Simulations Interface System"). We present the MoSIS prototype that implements this process based on a custom Jupyter kernel for the user interface and the theory-graph-based Mmt knowledge management system as an MKM backend.

1 Introduction and Motivation

Computational science and engineering (CSE) deals with the development and application of computational models and simulations, often coupled with high-performance computing, to solve complex physical problems arising in engineering analysis and design (computational engineering) as well as natural phenomena (computational science). CSE has been described as the "third mode of discovery" (next to theory and experimentation). Computer simulation provides the capability to enter fields that are either inaccessible to traditional experimentation or where carrying out traditional empirical inquiries is prohibitively expensive.

However, CSE as an interdisciplinary field requires a mixture of three fields of expertise, a skillset that is difficult to acquire and for which university programs are only now being established [Rü+16]. Thus at the heart of CSE resides a *knowledge management problem* where

1. **domain knowledge** – information and intuition about real-world processes – has to be combined with
2. **model knowledge** – i.e. how to express and describe the underlying relationships and processes in the domain with mathematical constructs, e.g. partial differential equations and

3. **simulations knowledge** on how to get accurate and efficient numerical approximations;

The disciplinary boundaries in CSE – see the clover leaf in Figure 1 for an illustration – do not fully align with these knowledge categories. Indeed, for many problems, CSE practicioners[1] will act as interpreters for application domain experts who usually know the problem to be simulated well and are mathematically literate. They coordinate with the domain experts on and via the models and provide the simulations expertise needed for the particular application.

We address the mathematical knowledge management (MKM) problem involved with "Mathematical Modeling and Simulation" (MMS) – a synonym for CSE that puts more emphasis on the modeling part. Following the MaMoReD (Math Models as Research Data) approach [KKMT17], we meta-model CSE/MMS knowledge using logical constructions such as as theory graphs and CSE/MMS practices like knowledge application as actions (e.g. pushout constructions).

Contribution In this paper, we show how existing MaMoReD theory graphs (see Section 2.1) can be utilized to solve the knowledge amalgamation problem inherent in the collaboration of domain experts and computational engineers. We discuss how theory graph structure can be exploited to automate the application knowledge acquisition processes ("Models-to-Simulations Interface System") to drive simulation tools directly – which is why MoSIS is situated at exactly the intersection of Application and Simulations Expertise in Figure 1. We distinguish persistent background knowledge that can be collected and curated in knowledge bases by the wider CSE community from ephemeral, application-specific knowledge that does not transfer to other situations.

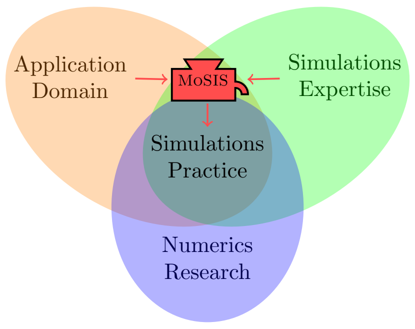


Fig. 1. Disciplines in Simulations Practice

To show the feasibility of the MoSIS approach, we implemented MoSIS 1.0, a simple interactive interface to the domain-specific language ExaSlang for the specification of PDEs and stencil-based algorithms. MoSIS 1.0 allows the user to generate both a flexiformal representation of their PDE model as well as its numerical solution. Furthermore, we explore other possible applications of knowledge management tools in mathematical modeling.

Overview This paper is a refined and condensed version of [Pol17], to which we refer for details and code. Section 2 sets up the running example, introduces the

[1] For this paper we disregard numerics research, which is advancing the available methods, as this largely is an off-line process that is motivated by concrete applications, but not invoked on a per-problem basis.

MaMoRed paradigm of "Mathematical Models as Research Data", and discusses the knowledge gap encountered with the solver ExaStencils. Section 3 presents a theory graph for the knowledge of our running example discusses the MoSIS prototype. Section 4 wraps up our findings and provides an outlook.

2 Meta-Modeling and Simulation

2.1 The MaMoReD Approach

We follow the MaMoReD approach – "Mathematical Models as Research Data" [KKMT17; Kop+18] – of explicitly representing mathematical models to enable computer support of CSE practices. Concretely, the model, all its assumptions, and the mathematical background in whose terms the model is defined, are represented as as a flexiformal theory graph (see Section 2.3). This can serve as a flexiformal documentation of the ideas and maths used in a project and can also be processed and compared computationally. However, it is fair to ask whether the benefits really outweigh the costs of creating the theory graph representation in practice. We work towards an affirmative answer by showing that MaMoReD theory graphs support added-value services for CSE: bridging the gap between problems and simulations.

To fortify our intuition, consider the following problem, which we will use as a running example throughout the paper:

Running Example (One-Dimensional Heat Conduction Problem)
Jen, an engineer, would like to simulate the heat conduction through the walls of her house. Jen knows basic physics in particular that heat distribution throughout a heated wall with constant surrounding temperatures is described by Poisson's equation when simplified to the static case. In the illustration, Figure 2, we see heating pipes going through the wall – which is made from materials of varying thermal conductivity k_1, k_2 – as well as the warm air on its side a, and cold on b. Jen has not worked with (thermal) simulations so far, but is aware of the pitfalls involved. Luckily she has a friend James who is a computational engineer. They discuss the model (see Section 2.2), James inquires about the specific parameters, and eventually uses them to set up an ExaStencils script, runs the simulation, and together they interpret the results.

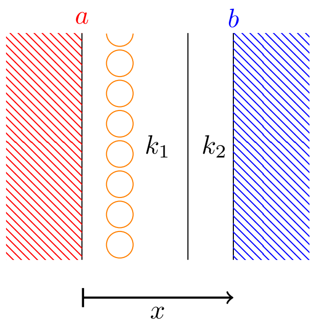


Fig. 2. Wall Schematics

The problem can be extended to include time or coupled systems later on, but Jen restricts herself to the static problem for the moment.

2.2 Poisson's Equation

Poisson's equation occurs in nature e.g. in electrostatics or in static heat conduction problems. Usually defining an unknown scalar function $u : \Omega \mapsto \mathbb{R}$ and an integrable source term $f : \Omega \mapsto \mathbb{R}$ on the domain $\Omega \subset \mathbb{R}^n$, it can be represented as

$$-\Delta u = f \ \text{ in } \Omega, \tag{1}$$

where the second-order differential operator Δ, the **Laplace operator**, is defined as the sum of the second order partial derivatives

$$\Delta = \partial_{x_1 x_1} + \partial_{x_2 x_2} + \cdots + \partial_{x_d x_d} \tag{2}$$

in d dimensions. As we will visualize in the next paragraph, u is the unknown temperature in our running example, and f is given as the heat transported into the wall divided by the thermal conductivity.

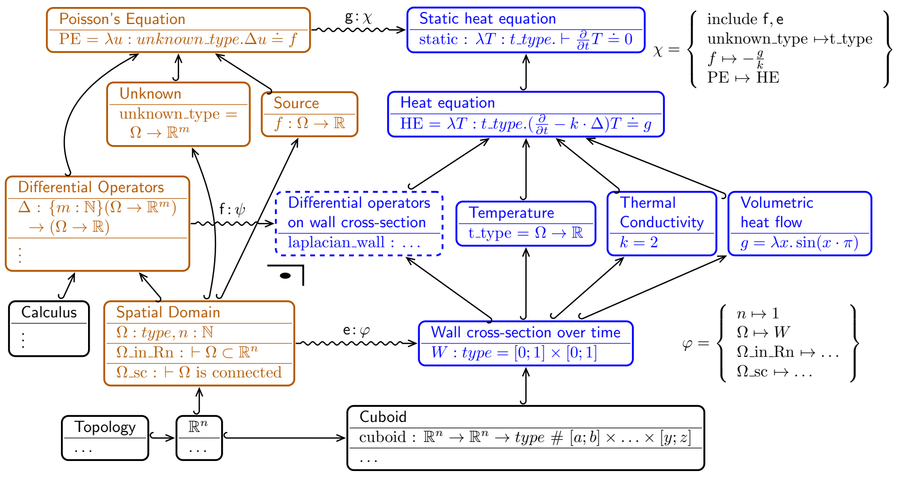
To allow the Poisson equation to be a (uniquely) solvable **boundary value problem**, we need to add **boundary conditions** imposed on u. For example, u may need to be equal to some other function b

$$u = b \ \text{ in } \partial\Omega, \tag{3}$$

as a **Dirichlet boundary condition** on all of the boundary $\partial\Omega$. We can then uniquely determine u as an element of the Sobolev space $H^1_b(\Omega)$.

2.3 Theory Graphs

Theory graphs [RK13] are a conceptual format for representing mathematical knowledge in a structured, modular form. They consist of **theories**, small units of knowledge, and **morphisms** – truth-preserving mappings – between them.



Poisson's Equation
PE = λu : unknown_type.Δu ≐ f
g : χ
Static heat equation
static : λT : t_type. ⊢ ∂/∂t T ≐ 0
χ = { include f, e; unknown_type ↦ t_type; f ↦ −g/k; PE ↦ HE }
Unknown
unknown_type = Ω → ℝ^m
Source
f : Ω → ℝ
Heat equation
HE = λT : t_type.(∂/∂t − k · Δ)T ≐ g
Differential Operators
Δ : {m : ℕ}(Ω → ℝ^m) → (Ω → ℝ)
f : ψ
Differential operators on wall cross-section
laplacian_wall : ...
Temperature
t_type = Ω → ℝ
Thermal Conductivity
k = 2
Volumetric heat flow
g = λx. sin(x · π)
Calculus
Spatial Domain
Ω : type, n : ℕ
Ω_in_Rn : ⊢ Ω ⊂ ℝ^n
Ω_sc : ⊢ Ω is connected
e : φ
Wall cross-section over time
W : type = [0; 1] × [0; 1]
φ = { n ↦ 1; Ω ↦ W; Ω_in_Rn ↦ ...; Ω_sc ↦ ... }
Topology
ℝ^n
Cuboid
cuboid : ℝ^n → ℝ^n → type # [a; b] × ... × [y; z]

Fig. 3. A Theory Graph Example: The Static Heat Equation (for Poisson's Equation)

The most important theory morphisms for this paper are the **inclusion** ($\hookrightarrow$) and **view** ($\rightsquigarrow$) relations. **Inclusions** are equivalent to unnamed imports in many programming languages, allowing to re-use symbols exactly as they were defined. **Views**, however, map symbols to apply different concepts of reasoning to the given situation, which in programming would be equivalent to Python-style duck typing, a common example being the concept of "example" itself.

For instance, in Figure 3 the Wall cross-section over time can be viewed as a Spatial Domain, as all necessary symbols are mapped in the view e, effectively making Wall cross-section an example of a Spatial Domain. Subsequently, every item of knowledge that can be generally formulated in terms of the one can then be applied to the other in particular, for example the Differential Operator Δ. This mode of knowledge generation is called a **pushout** and is ubiquitous in mathematical reasoning. In Figure 3, we see the pushout for Differential operators on wall cross-section denoted as a right angle with dot. Pushouts are a convenient construction whenever they occur, as they can be automatically generated [CMR17]. This allows us to walk up the graph further, such that finally a view g can be applied to see the Static heat equation as an instance of Poisson's Equation, which includes the views established earlier on.

To make theory graphs computationally usable, we express them in the MMT language (Meta Meta Toolset) [MMTa]. Concretely we base our formalizations on the **Math-in-the-Middle** (MitM) foundation, a feature-rich higher-order logic with dependent function and record types as well as predicate subtypes and general subtyping. This choice of primitives is geared towards ease of representation of mathematical knowlege without losing structure.

2.4 MoSIS: Creating ExaSlang Layer 0

In the ExaStencils project [EXA], the external, multi-layered domain-specific language (DSL) ExaStencils [Len+14; Kös+17] is developed in order to support automatic code generation of scalable and efficient numerical PDE solvers. The code generator is implemented in Scala and outputs e.g. parallel C++ or CUDA code [KK16]. Algorithms that can be expressed as stencils include most finite difference methods (FDM), which can be used to numerically solve e.g. Poisson's equation on a structured grid – computational grids are restricted to structured ones that lead to highly performant stencil codes.

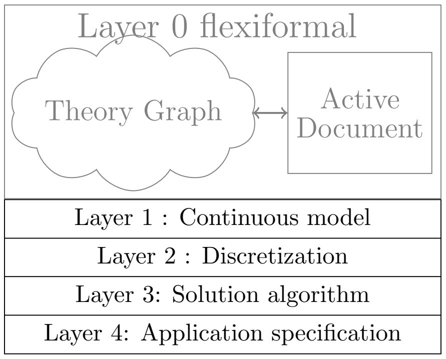

Layer 0 flexiformal
Theory Graph
Active Document
Layer 1 : Continuous model
Layer 2 : Discretization
Layer 3: Solution algorithm
Layer 4: Application specification

Fig. 4. MoSIS as ExaSlang Layer 0

ExaSlang is built from different layers, cf. Figure 4, which represent a transformation starting from an abstract continuous mathematical model (in Layer

1) down to the specifics of the application to be generated (in Layer 4). Especially Layer 1 is aimed at providing an intuitive interface for the mathematically versed user.

But there are two obstacles for ExaSlang to be an interface for MMS practicioners. Firstly, there is no interactive feedback when entering the PDE model; problems are only encountered once everything is set up and the user tries to compile and run the configuration.

Secondly, there is no input checking to ensure that the model is consistent in itself and neither under- nor over-specified as to allow for a unique solution of the PDE in the weak sense – or whether the (non-)existence of a unique solution may even be provable. This kind of consideration is part of the typical CSE/maths knowledge and is not easily represented on a low level of abstraction, such as C++ code.

The underlying problem in both cases is that the transformation process from – explicit or tacit – knowledge in the experts to program code is not structure-preserving and not easily reversible; a problem that the language hierarchy in ExaSlang is designed to mitigate. But practical experience shows that (at least) another stepping stone is needed.

3 MoSIS: Combining MaMoReD and ExaStencils

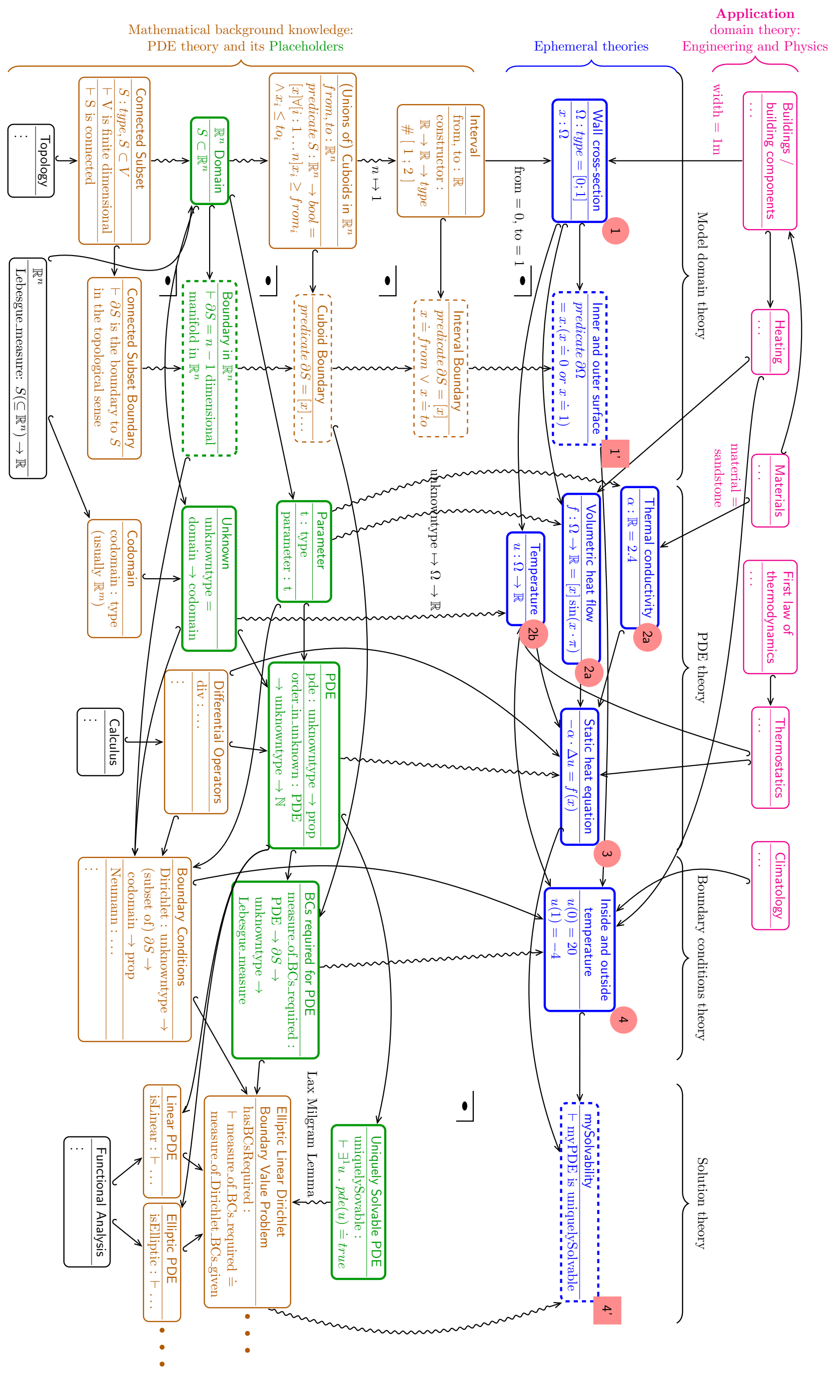
We propose a "Layer 0" for ExaSlang that uses an **active document** [Koh+11] for user interaction. It can use the **theory graph**, containing background knowledge to guide the user through the model description process. The resulting program can then translate the abstract model to ExaSlang Layer 1 code, which is still human-readable and can be translated to highly performant solver code via the ExaStencils tool chain. As a "by-product", a high-level representation of the mathematical model under consideration is generated.

In the following, we will call this idea MoSIS, the "Models-to-Simulations Interface System". The implementation we report on in this paper uses a Mmt-based **theory graph** and the **Jupyter notebook** [JN] as basic active document format. Building on the MaMoReD approach, we meta-model the process of establishing a mathematical model as an actual dialog carried out between a domain expert and a simulations expert. The main prerequisites for this are as follows.

3.1 A Theory Graph for PDE knowledge

To evaluate MoSIS in our running example we Mmt-formalized a simple meta-model of PDEs into a theory graph, cf. Figure 5.

At the base we see the more general mathematical concepts, such as the Euclidean space $\mathbb{R}^n$, arithmetics, calculus, etc. The specifics of PDEs come above that, answering questions like "*What do we expect of a domain?*" and "*What are the types of unknowns and the PDE itself?*". The lower right corner contains



Mathematical background knowledge:
PDE theory and its Placeholders
Ephemeral theories
Application
domain theory:
Engineering and Physics
Topology
Connected Subset
$S : type, S \subset V$
$\vdash$ V is finite dimensional
$\vdash$ S is connected
$\mathbb{R}^n$ Domain
$S \subset \mathbb{R}^n$
(Unions of) Cuboids in $\mathbb{R}^n$
$from, to : \mathbb{R}^n$
$predicate\ S : \mathbb{R}^n \to bool = [x] \forall i : 1 \ldots n[x_i \geq from_i \wedge x_i \leq to_i$
Interval
from, to : $\mathbb{R}$
constructor :
$\mathbb{R} \to \mathbb{R} \to type$
# [1 : 2]
$n \mapsto 1$
Wall cross-section
$\Omega : type = [0; 1]$
$x : \Omega$
from = 0, to = 1
Buildings / building components
width = 1m
Model domain theory
Lebesgue_measure: $S(\subset \mathbb{R}^n) \to \mathbb{R}$
Connected Subset Boundary
$\vdash \partial S$ is the boundary to S in the topological sense
Boundary in $\mathbb{R}^n$
$\vdash \partial S = n-1$ dimensional manifold in $\mathbb{R}^n$
Cuboid Boundary
$predicate\ \partial S = [x] \ldots$
Interval Boundary
$predicate\ \partial S = [x]$
$x \doteq from \vee x \doteq to$
Inner and outer surface
$predicate\ \partial\Omega = x.(x \doteq 0\ or\ x \doteq 1)$
Heating
Materials
material = sandstone
Codomain
codomain : type
(usually $\mathbb{R}^m$)
Unknown
unknowntype = domain $\to$ codomain
Parameter
t : type
parameter : t
unknowntype $\mapsto \Omega \to \mathbb{R}$
Temperature
$u : \Omega \to \mathbb{R}$
Volumetric heat flow
$f : \Omega \to \mathbb{R} = [x]\sin(x \cdot \pi)$
Thermal conductivity
$\alpha : \mathbb{R} = 2.4$
First law of thermodynamics
PDE theory
Calculus
Differential Operators
div : ...
PDE
pde : unknowntype $\to$ prop
order_in_unknown : PDE $\to \mathbb{N}$
Static heat equation
$-\alpha \cdot \Delta u = f(x)$
Thermostatics
Boundary Conditions
Dirichlet : unknowntype $\to$ (subset of) $\partial S \to$ codomain $\to$ prop
Neumann : ...
BCs required for PDE
measure_of_BCs_required : PDE $\to \partial S \to$ unknowntype $\to$ Lebesgue_measure
Inside and outside temperature
$u(0) = 20$
$u(1) = -4$
Climatology
Boundary conditions theory
Functional Analysis
Linear PDE
isLinear : $\vdash$ ...
Elliptic PDE
isElliptic : $\vdash$ ...
Elliptic Linear Dirichlet Boundary Value Problem
hasBCsRequired :
$\vdash$ measure_of_BCs_required $\doteq$ measure_of_Dirichlet_BCs_given
Lax Milgram Lemma
Uniquely Solvable PDE
uniquelySolvable :
$\vdash \exists^1 u.\ pde(u) \doteq true$
mySolvability
$\vdash$ myPDE is uniquelySolvable
Solution theory
1
1'
2a
2a
2b
3
4
4'

Fig. 5. Theory Graph for PDEs, Applied to 1-D Heat Conduction Problem: The ephemeral (blue) subgraph is forged between the background and application knowledge and provides an outline for the interview

theories about combinations of properties that make the PDE solvable[2], here only showing the parts needed to understand Poisson's equation.

The knowledge enoded the lower part of the graph is background knowledge that can be used to generally describe a PDE model and acts as an immutable common ground for multiple modeling/simulation tasks. For one particular such task (e.g. Jen's Wall), we need to generate application-specific theories, which we call **ephemeral**, since they can be discarded after the simulation. Ephemeral theories are not universally true– and as such need not become part of the **persistent** (non-ephemeral) knowledge base – but they are true for Jen's model.

Shown as the uppermost layer in the theory graph, the ephemeral theories are always connected to their background counterpart by way of views. The concrete ephemeral subgraph in Figure 5 is chosen for our running example: the static temperature distribution throughout the walls of a house, with fixed inside and outside temperatures. This can be mapped to be an elliptic linear Dirichlet boundary value problem, and is therefore provably uniquely solvable in the weak sense.

Figure 5 only shows the part of the MitM Ontology [Mitb] that is relevant to our – didactically chosen and thus relatively simple – running example. MitM itself is under constant development and covers other modeling case studies as well, e.g. the van Roosbroeck models discussed in [KKMT17; Kop+18]. It is our experience that the general topology of the theory graph – which is the only part that is actually used in MoSIS – does not fundamentally change for more complex examples, e.g. in higher dimensions.

Finally, note that the possible notation "∂" to denote the boundary is the same that is often used to describe the partial derivative. In practice, this only rarely causes confusion; which one is meant usually becomes clear from both the context and the type of object that the operator is applied to. The same is true for the theory graph representation here. In addition, a production MoSIS interface should allow for reformulation of notations – as long as no ambiguities are introduced – in order to support various mathematical traditions (e.g. f'_x, f_x, $\partial_x f$, $D_x f$, $D_1 f$, $\frac{\partial}{\partial x} f$, or $\frac{\partial f}{\partial x}$ as notations for the partial derivative used in different communities).

3.2 Automating Model Knowledge Amalgamation

The knowledge gap we discussed in the introduction is a tangible one: People – that are competent in their own domain – set up numerical tool kits themselves by diving into the technical documentation, just to end up with the wrong results (possibly without noticing and understanding why).

Luckily, our engineer Jen knows about the pitfalls and decides to talk to her friend James, a computational engineer. James asks her about the property she wants to determine – the temperature curve in the wall – and the situation at hand: concretely he needs the *i*) coordinates of the wall cross-section, *ii*) materials

[2] a lot of mathematical insight about weak solutions is represented by the view labeled "Lax-Milgram Lemma"

and their thermal properties, *iii*) the layout of heat sources in the wall, *iv*) physics of the wall (here given by the static heat equation), *v*) boundary conditions (inside and outside temperatures).

In the course of the discussion – we think of this as the **knowledge acquisition process** to be solved by James – Jen writes down some assignments and formulas. Only when everything is defined, and the meaning is perfectly clear and plausible to both of them, James tells Jen that he thinks the PDE problem is solvable – in this case even uniquely – and starts a suitable simulation (which may further be influenced by Jen's requirement on accuracy and time) and proudly presents Jen with the results: a nice visualization of the temperature distribution. But not every domain specialist has a friend or colleague who is a computational engineer; so automation of knowledge acquisition – and more generally of knowledge amalgamation, like the interaction between Jen and James – is desirable.

Fortunately, the sequence of interactions necessary to be able to fully specify the simulation can be read off the theory graph in Figure 5. Essentially, the shape of the green part is a template to the (blue) subgraph of ephemeral theories that forces the interview: We would require that at least one item is defined that can be understood as a Domain. We can allow users to use all constants defined in theories that can themselves be viewed as a Domain, such as the Interval constructor. Establishing a (transitive) view between Domain and Wall cross-section as a 1st user input immediately returns knowledge about the domain boundary, denoted by 1'.

Next, we would like to know about (optional) Parameters and at least one Unknown. Based on the dependencies given by the include structure, we notice that the order between the inputs 2a and 2b is arbitrary. What is peculiar at this point is that only the type but no concrete definition must be given for the Unknown, as otherwise it would in fact be known!

The inputs 3 and 4 define the PDE and Boundary Conditions as the central part of the model specification. If we now push out[3] 3 and 4 with theories PDE and Elliptic Linear Dirichlet Boundary Value Problems, we get the output theory mySolvability, which states that a solution $u : \Omega \to \mathbb{R}$ can be computed.

The theory graph in Figure 5 stops at "mathematical solvability theory", but could be extended to include numerical solvability by discretization on a grid and even solvability in particular simulation methods. Such extensions would greatly enhance practical coverage, but are beyond the scope of this paper.

Note that the ephemeral theories (in blue in Figure 5) necessarily combine information about the background mathematical (in orange) with knowledge of the physics of the problem (theories at the top in magenta). For specifics of this combination we refer the reader to [KKMT17; Kop+18], where we have elaborated on this in the case of the "van Roosbroeck Model" from quantum electronics. Here we focus on the application of such models to a specific situation, which we had previously only touched upon.

[3] The category of MMT theories and morphisms has co-limits – and thus pushouts, which can be calculated canonically [CMR17] in the MMT system.

Note furthermore that we have abbreviated the ephemeral theories and their provenance from physics; we give their constants concrete values by using full MMT declarations of the form by $c : \tau = \delta$, where c is the constant name, τ its type, and δ its definition. Actually, in the pure MaMoReD approach we would have divided them into a physical background theory with "undefined" constant $c : \tau$ and only instantiating c to δ in the corresponding ephemeral theory. In Figure 5 we have not elaborated the physics layer, because we have a different – but related – way of automating, which we discuss now.

The green theories (thick border) in the persistent part of the graph in Figure 5, e.g. Boundary in $\mathbb{R}^n$ and PDE are exactly the ones we want to map ephemerally in order to get the simulation information in the steps 1 to 4. We call them **placeholder** theories. The placeholder theories can be determined by looking for those constants in the persistent graph that do not have a definition (yet).

The order in which the placeholders needs to be filled in can be obtained by the persistent theory morphisms. Consequently, the ephemeral theories can be generated as fill-ins to the placeholder "form", carrying the same mirrored theory morphisms, and the exact same outer dependencies. This instantiation is the heart of the knowledge acquisition process.

Now one might say that it would be possible to just generate all ephemeral theories in the beginning and have all the constants assigned by the user. This is true if we generate the dialogue from a known model template as given in the application domain theory. The notable difference here is that we do not know how often Jen is going to fill certain parts into the generalized "model". For instance, she may define arbitrarily many parameters – Thermal conductivity and Volumetric heat flow in our running example – several unknowns and exactly as many determining equations, here one, but we still need only one solvability theory for the whole problem.

A slightly modified approach to generate interviews is offering a keyword for each placeholder. For instance, Jen might want to define a new constant by typing parameter my_favourite_room_temperature = 25 at any given time. The dependencies in the theory graph should then make sure that this happens only when appropriate, i. e. after a domain for x was defined. This feature would correspond to the user's habit of introducing parameter functions e.g. to keep equations more readable, or to extend the model to coupled physics, e.g. if Jen wanted to find out how much the wall will expand when heated. To this end, there is a lot that can be done with simplifications such as in-situ computations [ODK17] that simplify the model in real time, and the visualization of these changes. Adding keywords to the sequential interview effectively mimics mathematical model amalgamation as a mixed-initiative dialogue.

3.3 Implementation: Realizing MoSIS via Jupyter & MMT

Building on the ideas introduced in the last section, we have implemented MoSIS 1.0 in Jupyter [Jup]. This is an open-source web application that plays **Jupyter notebooks**: interactive documents that contain live code: equations, visualizations and narrative text. The code is executed in a backend kernel process, here running the MMT system. In contrast to normal Jupyter notebooks that use a fixed sequence of (documented) instructions in a read-eval-print-loop (REPL), MoSIS implements an ephemeral dialogue using a simple state machine with states for each part (1-4) of the theory graph to be set up. MoSIS uses a MMT kernel and communicates through the HTTP API of the MMT server. We have extended the MMT server to support the creation (and if needed the storage) of ephemeral theories (see [MMTb]).

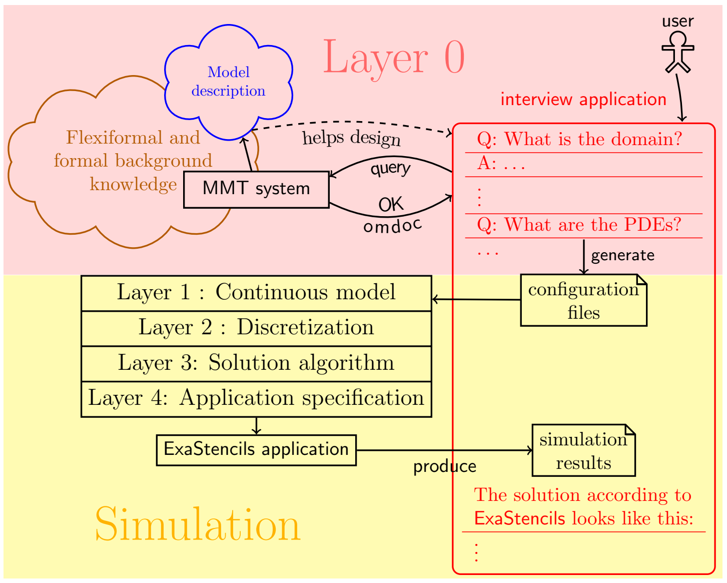
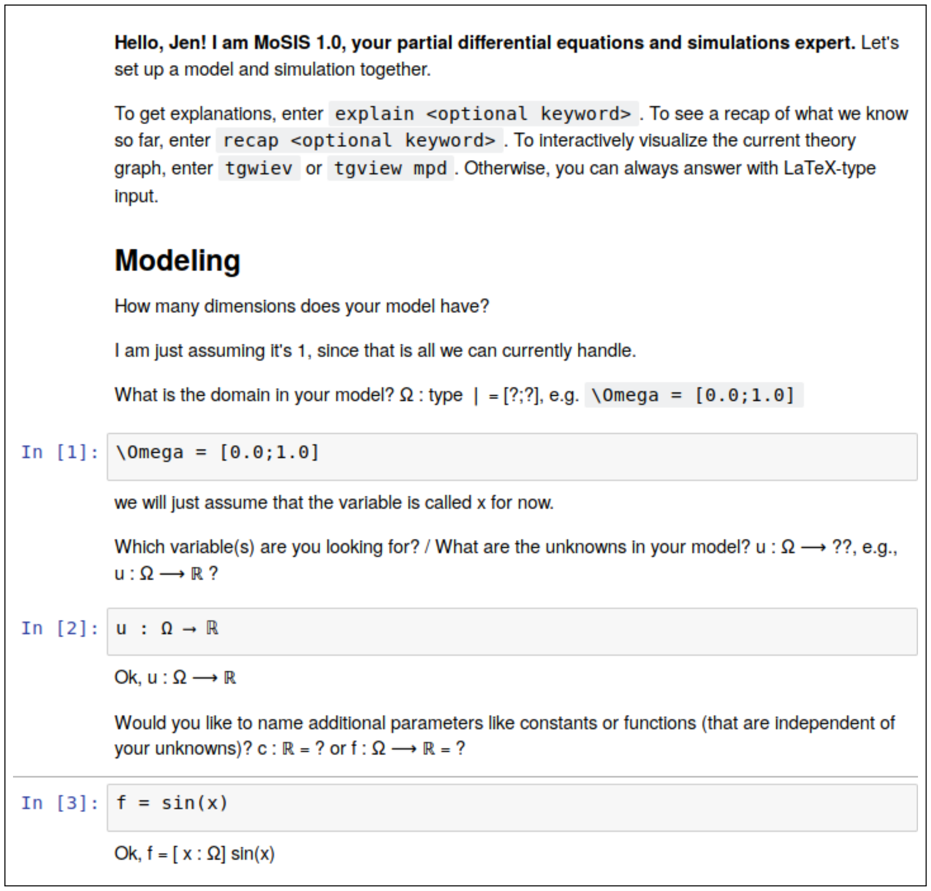


Fig. 6. MoSIS System Architecture

Figure 6 shows the MoSIS architecture. The left side is a concretization of the "Layer-0-design" from Figure 4, where the MMT system takes the place of the theory graph interface. The right hand side shows the interview – a Jupyter notebook – as the active document and how it interacts with the kernel level. In particular, the user only sees the notebook; answers the knowledge acquisition questions presented by MoSIS, until MoSIS can generate a ExaStencils configuration file to be shipped to ExaStencils, which transforms it into efficient code through the ExaSlang layers, computes the results and visualizations, that MoSIS in turn incorporates into the notebook. Note that the user (in our example the domain expert Jen) can run simulations without having to interact with the simulation system and learn its peculiarities (or consulting James).

On the front end, MoSIS 1.0 inherits the communication and representation capabilities from the Jupyter web application and notebook format. Questions and state information required by MoSIS are presented as Markdown cells. The user's replies can be written as "maths" encoded in MMT surface syntax employing MMT notations that were defined in the background knowledge theory graph beforehand. Following scientific practice, LaTeX representation can be used for non-standard Unicode characters. A side benefit of passing user inputs through MMT is that these are type-checked by the system. Already in our simple example this eliminates a considerable source of errors before they ever reach ExaStencils.


Hello, Jen! I am MoSIS 1.0, your partial differential equations and simulations expert. Let's set up a model and simulation together.

To get explanations, enter `explain <optional keyword>`. To see a recap of what we know so far, enter `recap <optional keyword>`. To interactively visualize the current theory graph, enter `tgwiev` or `tgview mpd`. Otherwise, you can always answer with LaTeX-type input.

Modeling

How many dimensions does your model have?

I am just assuming it's 1, since that is all we can currently handle.

What is the domain in your model? Ω : type ❘ = [?;?], e.g. `\Omega = [0.0;1.0]`

```
In [1]: \Omega = [0.0;1.0]
```

we will just assume that the variable is called x for now.

Which variable(s) are you looking for? / What are the unknowns in your model? u : Ω ⟶ ??, e.g., u : Ω ⟶ ℝ ?

```
In [2]: u : Ω → ℝ
```

Ok, u : Ω ⟶ ℝ

Would you like to name additional parameters like constants or functions (that are independent of your unknowns)? c : ℝ = ? or f : Ω ⟶ ℝ = ?

```
In [3]: f = sin(x)
```

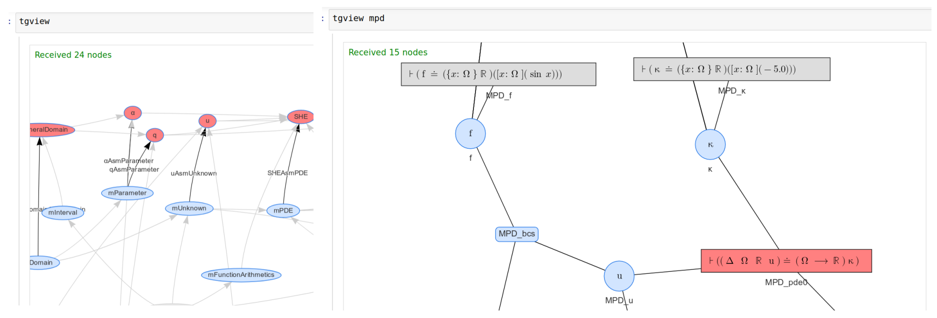
Ok, f = [x : Ω] sin(x)

Fig. 7. Beginning of a Dialogue in MoSIS

For the translation into ExaSlang Layer 1 code, we use the views into the ephemeral theories, e.g. from Domain to Wall cross-section – theory graph morphisms can be composed and transform into Domain syntax via MMT notation definitions. This is also where a central benefit of the theory graph representation comes into play: The theory graph always contains the same or more information than any specific programming language representation of the same model, and can therefore be used to translate to different changing formats once a translation is available. This is especially interesting in the context of the **Math-in-the-Middle approach** [Deh+16] of aligning mathematical open source tools through a common flexiformal mathematical reference point: the MitM Ontology [Koh+17].

MoSIS uses ExaStencils to generate the solver code for this particular problem and runs it – and the user is presented with a Bokeh [BO12] visualization of the solution. But what is more, they now also have a re-usable representation of how they got to the result, both as a Jupyter notebook (for narration) and an OMDoc file (for further computation). Figure 7 shows (the beginning of) an actual MoSIS dialogue for our running example; see [MoS] for a live demo.

In addition to the dialogue turns, the user can orient herself using special MoSIS actions: getting a recap of all the ephemeral information stored so far, undoing the last step(s) and having a theory graph or Model Pathway Diagram (MPD) [KKMT17] of the current working state displayed back through the theory graph viewer TGView [RKM17], cf. Figure 8. Our user can discover, inspect and "debug" the structure of the model captured. Given a Model/MPD graph following [KKMT17; Kop+18], we can use the MPD view as a more intuitive user interface for inspecting the PDE.



```
tgview
```

Received 24 nodes

```
tgview mpd
```

Received 15 nodes

α
u
SHE
neralDomain
q
αAsmParameter
qAsmParameter
uAsmUnknown
SHEAsmPDE
mParameter
mUnknown
mPDE
mInterval
Domain
mFunctionArithmetics
$\vdash(f \doteq (\{x\colon\Omega\}\,\mathbb{R})([x\colon\Omega](\sin x)))$
MPD_f
$\vdash(\kappa \doteq (\{x\colon\Omega\}\,\mathbb{R})([x\colon\Omega](-5.0)))$
MPD_κ
f
f
κ
κ
MPD_bcs
u
MPD_u
$\vdash((\Delta\ \Omega\ \mathbb{R}\ u) \doteq (\Omega \longrightarrow \mathbb{R})\,\kappa)$
MPD_pde0

Fig. 8. The Theory Graph Viewer TGView Embedded into Jupyter

The code for MoSIS kernel can be found in [Pol]; see [Mita] for formalizations. In addition, the MoSIS 1.0 kernel is set up in a JupyterHub environment under [MoS]. JupyterHub [JHub] is a Jupyter notebook server that is accessible to many users who can independently work in their own notebooks in the browser and execute code on the server, such that no software needs to be installed locally.

4 Conclusion and Future Work

This paper addresses the knowledge gap in modeling and simulations practice: People who want to derive or work with new application models know that at the end, a PDE has to be solved, but usually no background or interest in the required computation process. But an efficient solver depends on the PDE and thus detailed system knowledge or intense discussions with simulation experts are necessary, or else errors are bound to occur.

In the ExaStencils project, this problem has so far been approached with the development of a dedicated domain-specific language, ExaSlang, cf. Section 2.4. The knowledge gap discussed above corresponds to the problem of specifying the model on ExaSlang's most abstract layer, layer 1.

MoSIS fills the gap by providing an architecture and components for it by combining represented knowledge with an active document that the user can interact with. We established the feasibility of MoSIS by implementing MoSIS 1.0 based on existing components: Mmt for the representations of formal (and flexiformal) knowledge as a theory graph (cf. Section 2.3) and an interactive Jupyter kernel with a Mmt backend through which it can access PDE knowledge.

We currently have only developed the theory graph to the extent necessary for our running example. For more realistic simulations we will need to extend it to n-dimensional calculus and *partial* differential equations. For that we will need to extend Mmt and the MitM foundation from a purely logical system to one that can also "understand" equations as **quotations**. In our models, the names of the unknowns and variables in the equation actually matter, such that e.g. alpha-renaming these parts is not desirable (even though they are not formally bound in the model). One possible approach of dealing with this could be the internal enumeration of coordinates and variables.

To enhance the interactivity of the MoSIS front-end we are working on integrating Jupyter widgets [IPyWid15] into MoSIS, e.g. for selection inputs.

The technical terms used in the Markdown content can come with hoverable or clickable explanations as we already know them from the semantic glossary SMGloM [Gin+16]. Co-highlighting the aforementioned terms together with all the corresponding ephemeral and persistent mathematical symbols – in the best case also in the theory graph viewer and other possible tools – would greatly support the effect of visualizing what belongs together.

Acknowledgments The authors acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541); Project ExaStencils received funding within the DFG priority programme 1648 SPPEXA. We gratefully acknowledge fruitful discussions with the KWARC group, especially Dennis Müller and Florian Rabe for support with the Mmt implementation, to Thomas Koprucki and Karsten Tabelow at WIAS Berlin; and to Sebastian Kuckuk (LSS chair, FAU). Last but not least, Kai Amman and Tom Wiesing have helped with the Jupyter frontend and deployment of MoSIS on JupyterHub.

References

[BO12] *bokeh: Interactive Web Plotting for Python.* Mar. 26, 2012. URL: `https://github.com/bokeh/bokeh` (visited on 04/18/2018).

[CMR17] Mihai Codescu, Till Mossakowski, and Florian Rabe. “Canonical Selection of Colimits”. In: *Recent Trends in Algebraic Development Techniques*. Ed. by Phillip James and Markus Roggenbach. Springer, 2017, pp. 170–188.

[Deh+16] Paul-Olivier Dehaye et al. “Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach”. In: *Intelligent Computer Mathematics 2016*. Ed. by Michael Kohlhase et al. LNAI 9791. Springer, 2016. ISBN: 978-3-319-08434-3. URL: `https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf`.

[EXA] *Advanced Stencil-Code Engineering (ExaStencils)*. URL: `http://exastencils.org` (visited on 04/25/2018).

[Gin+16] Deyan Ginev et al. “The SMGloM Project and System. Towards a Terminology and Ontology for Mathematics”. In: *Mathematical Software - ICMS 2016 - 5th International Congress*. Ed. by Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew Sommese. Vol. 9725. LNCS. Springer, 2016. DOI: `10.1007/978-3-319-42432-3`.

[IPyWid15] *ipywidgets: Interactive widgets for the Jupyter Notebook*. Apr. 17, 2015. URL: `https://github.com/jupyter-widgets/ipywidgets` (visited on 04/18/2018).

[JHub] *JupyterHub — JupyterHub documentation*. URL: `https://jupyterhub.readthedocs.io/en/latest/` (visited on 04/18/2018).

[JN] *Jupyter Notebook*. URL: `http://jupyter-notebook.readthedocs.org/en/latest/notebook.html#notebook-documents` (visited on 08/22/2017).

[Jup] *Project Jupyter*. URL: `http://www.jupyter.org` (visited on 08/22/2017).

[KK16] Sebastian Kuckuk and Harald Köstler. “Automatic Generation of Massively Parallel Codes from ExaSlang”. In: *Computation* 4.3 (Aug. 4, 2016), p. 27. ISSN: 2079-3197. DOI: `10.3390/computation4030027`. (Visited on 10/09/2017).

[KKMT17] Michael Kohlhase, Thomas Koprucki, Dennis Müller, and Karsten Tabelow. “Mathematical models as research data via flexiformal theory graphs”. In: *Intelligent Computer Mathematics (CICM) 2017*. Ed. by Herman Geuvers et al. LNAI 10383. Springer, 2017. ISBN: 978-3-319-62074-9. DOI: `10.1007/978-3-319-62075-6`.

[Koh+11] Michael Kohlhase et al. “The Planetary System: Web 3.0 & Active Documents for STEM”. In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhisa Sato et al. Finalist at the Executable Paper Grand Challenge, pp. 598–607. DOI: `10.1016/j.procs.2011.04.063`.

[Koh+17] Michael Kohlhase et al. “Knowledge-Based Interoperability for Mathematical Software Systems”. In: *MACIS 2017: Seventh International Conference on Mathematical Aspects of Computer and Information Sciences*. Ed. by Johannes Blömer, Temur Kutsia, and Dimitris Simos. LNCS 10693. Springer Verlag, 2017, pp. 195–210. URL: `https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/MACIS17-interop/crc.pdf`.

[Kop+18] Thomas Koprucki et al. “Model pathway diagrams for the representation of mathematical models”. In: *Journal of Optical and Quantum Electronics* 50.2 (2018), p. 70. DOI: `10.1007/s11082-018-1321-7`.

[Kös+17] Harald Köstler et al. “A Scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms”. In: *International Journal of Computational Science and Engineering* 14.2 (2017), pp. 150–163.

[Len+14] Christian Lengauer et al. “ExaStencils: advanced stencil-code engineering”. In: *European Conference On Parallel Processing.* Springer. 2014, pp. 553–564.

[Mita] *MitM / MoSIS*. URL: `https://gl.mathhub.info/MitM/MoSIS` (visited on 04/18/2018).

[Mitb] *MitM: The Math-in-the-Middle Ontology*. URL: `https://mathhub.info/MitM` (visited on 02/05/2017).

[MMTa] *MMT – Language and System for the Uniform Representation of Knowledge*. project web site. URL: `https://uniformal.github.io/` (visited on 08/30/2016).

[MMTb] *UniFormal/MMT – The MMT Language and System*. URL: `https://github.com/UniFormal/MMT` (visited on 10/24/2017).

[MoS] *JupyterHub - MoSIS Demo*. URL: `http://mosis.mathhub.info` (visited on 04/15/2018).

[ODK17] Michael Kohlhase and Tom Wiesing. *In-place computation in active documents (context/computation)*. Deliverable D4.9. OpenDreamKit, 2017. URL: `https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP4/D4.9/report-final.pdf`.

[Pol] Theresa Pollinger. *interview_kernel*. URL: `https://gl.kwarc.info/theresa_pollinger/MoSIS` (visited on 04/11/2018).

[Pol17] Theresa Pollinger. “Knowledge Representation for Modeling and Simulation – Bridging the Gap Between Informal PDE Theory and Simulations Practice”. Master’s Thesis. Informatik, FAU Erlangen-Nürnberg, 2017. URL: `https://gl.kwarc.info/supervision/MSc-archive/blob/master/2017/tpollinger/thesis.pdf`.

[RK13] Florian Rabe and Michael Kohlhase. “A Scalable Module System”. In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: `http://kwarc.info/frabe/Research/mmt.pdf`.

[RKM17] Marcel Rupprecht, Michael Kohlhase, and Dennis Müller. “A Flexible, Interactive Theory-Graph Viewer”. In: *MathUI 2017: The 12th Workshop on Mathematical User Interfaces*. Ed. by Andrea Kohlhase and Marco Pollanen. 2017. URL: `http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf`.

[Rü+16] Ulrich Rüde et al. “Research and Education in Computational Science and Engineering”. In: *arXiv:1610.02608 [cs, math, stat]* (Oct. 8, 2016). arXiv: `1610.02608`. URL: `http://arxiv.org/abs/1610.02608` (visited on 02/23/2018).