FEA solver integration framework

Jerome Szarazi
(Koneksys, United Kingdom);
Conrad Bock
(US NIST, United States);

Abstract

Integrating finite element analysis (FEA) with systems engineering (SE) would improve traceability, consistency, interoperability and collaboration between SE and FEA activities in multiple engineering disciplines. The first step in achieving this is a software-independent description of FEA models, which are characterized by numerical approximations of partial differential equations (PDEs) derived from physical laws, and finite elements representing unknown physical quantities. In previous work, we presented a finite element mathematics specification that is formal and understandable by most engineers. It provides all information needed for generation of shape functions for physical quantities. In this work, we propose a specification of physics in FEA to complement our earlier mathematics specification.

We first compare existing FEA physics descriptions and their software implementations to highlight the benefits of domain-independent model descriptions used by PDE solvers. A significant drawback of PDE representations is they do not show all physical quantities from which they are derived. To tackle this, we represent the physical laws and derivations needed for FEA PDEs in human- and machine-readable graphs. Instead of classifying physics problems by the kind of PDE, as in PDE solver packages, we formalize problems as paths through these graphs. This increases transparency by capturing modelling decisions currently done on paper or in electronic documents.

We combine the graph-based specification of FEA physics above with the finite element mathematics specification developed earlier to generate linear system of equations (algebraic FEA models) for solving the problem numerically. This combination will enable FEA engineers to design their own libraries (potentially automatically) if they choose, or associate existing solvers. It also generalizes mappings from physics to FEA models, a task currently repeated across specific disciplines. The framework could be standardized and integrated with SE modeling languages, improving interoperability and collaboration between systems and FEA engineers.

1. Introduction and motivation

Simulation-driven design has increased efficiency of product development using computer aided design (CAD) and repeated evaluation of these models with simulations. Software tools for simulation, such as finite element analysis (FEA), computational fluid dynamics (CFD), multibody dynamics (MBD) fall into the category computer aided engineering software (CAE).

Among these kinds of simulations, FEA is one of the most popular numerical methods to test virtual models. For example, FEA helps find the location of maximum stress on a body and therefore assess if and where a product may fail. After identifying weak design locations in simulation results, the design can be digitally corrected using CAD software. By alternating between design and simulation, the virtual model can be refined to eventually meet product requirements. This reduces the need for expensive physical prototypes to test designs. This simulation-design loop helps improve decisions at an early stage of engineering by choosing the best solution candidates in the design space.

Deciding whether a solution meets all requirements also includes reaching agreement between stakeholders. Cross-functional development teams, parallel processes and integrated CAE are necessary to be competitive, but efficient collaboration between FEA engineers, designers and other disciplines is essential for a project success. Difficulties in sharing information and miscommunication increase the time to reach agreement over design alternatives. Such problems are further complicated by use of multiple discipline specific models, often uncoupled and regenerated throughout the design cycle.

Systems engineering (SE) is increasingly used to overcome these problems. Traditional SE provides methodologies, processes and documentation to support such cross-disciplinary development process. In recent years, model-based SE (MBSE) significantly increased efficiency by exchanging models instead of documents [1]. In an MBSE scenario, systems are described by information models, in the same way software code is described by an architecture model.

The Systems Modeling Language (SysML®) [2], an extension of the Unified Modeling Language (UML®) [3], is the common language used to describe and exchange models among system engineers. These system models are digital artefacts, similar to 3D dimensional designs or simulation models, enabling mappings between other simulation data and SysML. Such transformation or integration is facilitated by standard data exchange for simulation models in particular domains. This reduces integration to the interfaces between the domain standards and SysML. For example, Modelica [4], an open and standardized language for time-only simulations, has a standard integration with SysML [5]. Data integration of simulation models with SysML enables traceability to requirements. When information exchange at the interface between a simulation model and SysML is bidirectional, it can lead to simulation automation by synchronizing system requirements changes

with simulation parameters and testing validity of system modifications with simulations. This is also true of simulation optimization, a process critically dependent on bidirectional model mappings.

For FEA, there is, unfortunately no open language to create models or exchange them. As a result, model-based integrations of FEA are platform specific. The absence of standardization leads to point to point integration between FE solvers, often a highly laborious manual or individually scripted process that must be repeated between each pair of points, which is expensive to maintain and sensitive to software updates.

To overcome these problems, we propose a new language for future standardization that helps share reusable information about FEA library models and generate finite element code, with inputs that are understandable by most engineers. This language goes beyond the specific usage of FEA, because it can be applied to physical system modeling in general. It enables FEA engineers to understand relationships between physical quantities within a physical theory and capture modelling decisions. It formalizes the otherwise time consuming modelling process currently done on paper or electronic documentation.

Section 2 describes FEA workflow, data generated at each stage, and reviews standards or formats that support data exchange. Section 3 outlines challenges currently hampering data exchange, one related to converting CAD to mesh data in the context of simulation driven design, and another related to the describing library elements independently of tools. Section 4 presents a new FEA solver integration framework that uses graph structures to capture physical laws, modelling decisions and all model parameters required for a numerical solver. Associated with a mesh standard with virtual topology capabilities, this framework could lead to a new FEA software ecosystem that helps engineers to create and specify their library models. Section 5 concludes the paper.

2. FEA data and standardization

FEA standardization is complex as it requires harmonizing geometric data exchange between CAD and meshes, a unified description of FEA library models and a common format to share simulation results. These three data models are digital artefacts created during FEA simulation, which has three steps: 1) meshing CAD geometry, 2) solving a simulation model, and 3) processing results [6].

a. Meshing

The first step, meshing, requires access to CAD geometry. Such access is facilitated by International Organization for Standardization (ISO) 10303 STandard for the Exchange of Product model data (STEP), a CAD file format supported by all CAD tools [7]. There are many available automatic and semi-automatic meshing algorithms, classification of such algorithms can be found in [8], as well as open source code that generate meshes from STEP files. A

mesh or grid approximates a desired geometry with a collection of discrete shapes in 2D or discrete volumes in 3D used as the computational elements for FEA. Unfortunately, the STEP format for meshes is not widely adopted, the only popular formats being associated with FEA software or meshing tools [9]. In general, mesh information can be stored using a finite element (shape or volume) to node data structure, where each indexed finite element of the mesh refers only to their nodes coordinates or by storing the complete mesh topology, which adds connectivity between the elements that compose mesh shapes or volumes. The first reduces storage space, while the second optimizes data access to volumes, faces and edges for adaptive FEA algorithms such as h-p refinement [10].

b. Solving

The second step of FEA simulation is selection, parametrization and solving of FEA models. There are two groups of FEA solvers, one that abstracts numerical information, and another that abstracts physical domains, which consequently have different inputs. For both models, spatial regions of the mesh, later associated to boundary conditions values, need to be selected and stored.

The first group of FEA solvers provides domain-specific models to solve mechanical (solid or fluid), electrical or thermal problems or a combination of them (multi-physics problems). Each model, called a library element, is associated with a template where material, additional geometric parameters, and initial and boundary conditions values can be set. These models, often identified by proprietary codes, do not provide source code and numerical choices are unknown most of the time. Simulation documentation contextualize the model and guide set-up of the input deck, most often represented as an ASCII file storing mesh, library model reference codes, and boundary and initial conditions. Graphical user interfaces (GUIs) of modern FEA software help in model set-up.

The second group of FEA solvers are partial differential equations (PDE) solvers that uses the finite element method (FEM). PDE solvers, such as Diffpack [11], identify models by PDE type, while others like FENics [12] or FreeFem++ [13] create models by entering the weak or variational form of the PDE, a multilinear functional, constructed by integrating PDEs by parts. The unknown variables of the weak form, usually physical quantity kinds, are substituted by one or more interpolation functions, called finite elements. Domain-independent models are defined with symbolic expressions using a domain-specific language like FreeFem++ or multi-purpose language such as Python [14] for FEnics.

c. Post-processing

The third and final step of FEA simulation is post-processing and evaluation of results. After solving the linear equations, solutions are post-processed if other model quantities are required. For example, in mechanics,

stress fields can be post-processed from displacement fields. Domain-specific solvers store relations between model quantities in their closed source code. Writing additional post-processing code is very often required for domain-independent software.

Evaluating results is facilitated by visualizing physical field quantities by projecting results onto the meshed geometry. Most commercial software solutions provide visualization. The open source visualization platform, Paraview [15], seems to be the most popular open source solution. Its VTK format [16], serialized in a proprietary text format or XML [17], is a de facto standard to exchange simulation results.

3. Current problems and proposals

There are two main problems related to FEA data exchange. First, design-simulation loops are more frequent, but current geometry standards do not facilitate integration between disparate CAD and CAE models, i.e., design geometries and analysis meshes. The other is lack of formal identification of numerical and physics models, which could be resolved using domain-independent models.

a. CAD/CAE integration

The traditional FEA workflow (see subsection 3.b) has five types of digital assets: detailed CAD models or master CAD models, digital mockups,[1] CAD STEP files, FEA mesh proprietary files with boundary conditions, and FEA results files. These are regenerated and maintained whenever the master CAD source is modified. Strategies for CAD/CAE integration framework are detailed in [18].

An intermediary neutral CAD file format, e.g., ISO 10303, provides platform independency between CAD and CAE software, but introduces substantial work due to geometric errors and meta-data losses during the conversion process from CAD proprietary formats to the STEP standard. Geometric errors are due to proprietary modelling of tolerances by CAD systems, requiring correction of the resulting “dirty geometry” [19][20]. In addition, most STEP translators usually omit construction history, parameters and constraints [21], and do not maintain storage of other meta-data, e.g., part names or material data.

The benefit of parametric associative CAD software is that modifications of some parameters, e.g., length, will update all related geometry downstream as well as keep constraints and meta-data associations. Product functionality is maintained by separate administration of geometry and constraints. Ideally, CAE information, such as material and boundary conditions, should be

[1] FEA is usually preceded by either defeaturization or idealization of the detailed CAD design. The resulting simplified models, called digital mock-ups help evaluate system assemblies or kinematics, perform MBD or FEA simulations and visualize FEA results on assemblies or parts [23].

maintained at the CAD level in order to be reused between CAx applications, e.g., MBD and FEA. Most CAD programs provide data interfaces, e.g., product data management (PDM), to enrich CAD models with metadata.

To provide CAD/CAE integration, current trends in CAD technology lean towards solutions that combine CAD and CAE in the same environment, further reducing compatibility between platforms [22]. Before future geometry standards arrive, an intermediary solution could be developing a standard that enables CAD environments to exchange mesh data structures with virtual topology capabilities, as described in [23], by keeping topological correspondence between CAD and mesh. Metadata associated with regions of the mesh could with a standard be exchanged between FEA application and CAD. In comparison to curved geometries in CAD systems, tessellated geometries with polygons faces do not suffer the same problems with tolerances. By moving the CAD/CAE interface from CAD to mesh, workflow efficiency of parametric CAD software is still maintained and platform independence of CAE solvers could be guaranteed. A general topology mesh data structure that captures the connectivity of all mesh elements, as detailed in [25], could maintain topological correspondence between CAD and mesh, enabling mesh regions of interest to be defined, such as material, interfaces or boundary conditions, improving data access, e.g., generation of internal nodes for higher degree finite elements. The mesh could be exchanged with the JSON data format [26], a key-value pairs data structure, that would facilitate data access to the mesh and the virtual topology. The keys, acting as pointers, could link CAD metadata and simulation information to geometric information required by the solver.

b. Formal CAE model identification

Another problem is lack of formal standards that provide enough information to run a simulation with any solver interchangeably. STEP AP209, the standard for FEA model exchange [27], targets mainly domain-specific software, capturing software name, version, and software models identified with proprietary reference codes. The standard provides taxonomies to classify simulation models, but they are too informal to associate a specific solver. Model reconciliation with STEP AP209, evaluating if two models are equivalent, is not possible because source code and numerical assumptions are closed for domain-specific software. This is problematic for solution migration when companies decide to change FEA software; for supplier collaboration when two companies use different software; and for long-term archiving if the software is not supported in the future. Resolving these challenges associated with STEP AP209 is undertaken as a common effort by various groups at different levels. For example, LOTAR's (Long Archiving and Retrieval) Engineering Analysis and Simulation Workgroup (EAS) develops, publishes and maintains standards for archiving and retrieval of key FEA input/output characteristics at various stage of development in a robust and repeatable fashion [28]. For collaboration, MOSSEC (Modeling and Simulation

information in a collaborative Systems Engineering Context), develops methods for organizing and sharing Modeling and Simulation meta- data and information in a collaborative system, and for capturing context to enable traceability [29]. Tools to analyze and visualize STEP file are developed by NIST [30].

In comparison, domain-independent software solutions require details about numerical choices. Their models are reusable in multiple physical domains that have the same mathematical structure. For example, a physics problem described by a Laplace PDE can be used to solve thermal conduction as well as electrostatics problems. Symbolic code is very often used for model definition, which can be automatically compiled into a lower level C code. Symbolic code was in use in the 90s for finite difference in the Sinapse framework [31] and in the later 90s for finite elements [32].

The weak form equation, formalized in [33] as a language and extended in [34] as a unified framework for finite element assembly, complemented by a finite element and user defined function, is enough information to symbolically or numerically integrate each cell of a mesh and assemble a linear system of equations that is interpretable by a linear algebra solver (LAS). Solving the systems of equations at this stage only requires linear algebra methods, such as Cholesky or Krylov subspace methods that many frameworks support, such as Petsc [35] with parallel processing capabilities or libraries compatible to the BLAS specification [36].

However, writing symbolic expressions for PDE weak forms is uncommon for FEA engineers. Even though PDEs are widely-understood and useful for model classification, using PDEs for model identification is problematic because boundary conditions refer to variables not in the PDEs. Without documentation detailing a PDE's derivation from multiple equations, they provide only a partial view of the model, limiting post-processing of other model variables. Such post-processing requires expert knowledge of model relationships to derive finite elements of post-processed variables. In contrast, domain-specific software models hide finite element choices and variable relationships, but offer simpler post-processing capabilities.

Ideally FEA model definition should be understandable by most engineers and generate solutions with any solver. Starting with weak forms, we can reverse-engineer the workflow used to find their expressions. Weak forms, can be derived from PDEs using integration by parts, enabling weak forms to be linked to their corresponding PDEs or automatically derived. Next PDEs are derived from model equations that correspond to physical laws. Transitioning from domain-independent descriptions, the mathematical model, to domain-dependent description, the physics model, is done by substituting mathematical variables with physical quantities. The resulting physics model is understandable by engineers and can be used by numerical methods such as finite volumes (FV) or finite differences (FD). For FEA models, finite elements that describe unknown physical quantities and space-dependent parameters are needed, in combination with physics models formulated as PDE weak forms to

construct linear algebraic systems of equations that can be solved by LASs.

To achieve formal description of FEA library models, we need a formal description of finite elements as well as an efficient mechanism to describe physics equations and the corresponding derivation of PDEs. The rest of the paper discusses these two aspects in more detail.

4. New FEA solver integration Framework

In this section, we present a new platform-independent approach to integrate FEA solvers. To provide an open model specification, subsection 4.a introduces building blocks of a graph-based language to capture physical equations, as a starting point for model definition. Linking these equation graphs together captures the mathematical structure of physics describing the system being analyzed. Subsection 4.b explains how math or physics problems are defined by selecting known and unknown variables in math and physics graphs. This captures modeling decisions as functional programs, enabling extraction of all problems as math/physics evaluation subgraphs or math/physics solver subgraphs. Subsection 4.c details how boundary condition types can be automatically collected and associated to math/physics solver subgraphs. To facilitate reusability, subsection 4.d introduces the concept of common mathematical structure that can be reused for multiple domains, illustrated in subsection 4.h with the rapid extension of library elements. Once problems are defined, subsection 4.e explains how physical quantities are represented in FEA by finite elements, with examples on how to use our finite element specification. Subsection 4.f clarifies the association of finite elements to physics solver subgraphs by creating numerical solver graphs and generating input templates for solvers. In subsection 4.g, an example presents how a mesh standard with virtual topology capabilities would associate design and FEA metadata and, together with the library element specification of subsection 4.e, provide a platform-independent description of FEA solvers.

a. Equation models

Whereas time-only (a.k.a., lumped parameter, 1D, network) simulation handles simple topologies (e.g., electric circuits) of many different library elements (e.g., capacitor, resistor…), FEA simulates one or few library element(s) on space embedded topologies (meshes). FEA library elements are more complex than time-only library elements and space is multi-dimensional and multi-directional in contrast to time, which flows in one direction.

Time-only solutions have open standards, such as Modelica or proprietary languages, such as XCOS [37] and Simscape/Simulink [38], to describe libraries or models, which are defined with physics equations. For signal-based simulations, mathematical expressions of Laplace transformations are either captured in custom or standard library blocks that can be connected together.

In contrast, FEA solutions deliver libraries as complied code that are parametrized with input templates. This transparency problem can only be solved with an ecosystem shift that gives FEA engineers the same flexibility as

time-only simulations. To achieve this, we believe physics equations should also be inputs to define FEA libraries. This would help in modeling FEA libraries, which is currently done on paper or electronic documents.

Mathematical structures of space-dependent physics models (physical equations describing a domain), are captured by Tonti diagrams [39] and used in [40] to illustrate FEA variational principles. Tonti diagrams are formalized with the cell method, a discretization method based on algebraic topology, first introduced by Branin [41], that defines topological dualities for both space and time as well as cell-specific mesh discretizations.

FEA engineers are not familiar with the cell method, so we propose a more general approach, one that first captures equations as triples, two physical quantities linked by an operator. Triples are linked whenever they share the same elements; a graph is automatically created. These graphs define symbolic computation, as compared to Simulink graphs that define real number processing. With this approach, structures described by Tonti diagrams can be captured in a way more accessible to engineers.

To explain how this works, let's consider an engineer who aims to build an FEA library with this approach. His first step would be to find relevant physical equations in the literature. These could be captured using a GUI that helps select and connect necessary blocks, or by entering symbolic equations that are progressively displayed as graphs. Figure 1 shows the gradient law for relating potentials and gradients.

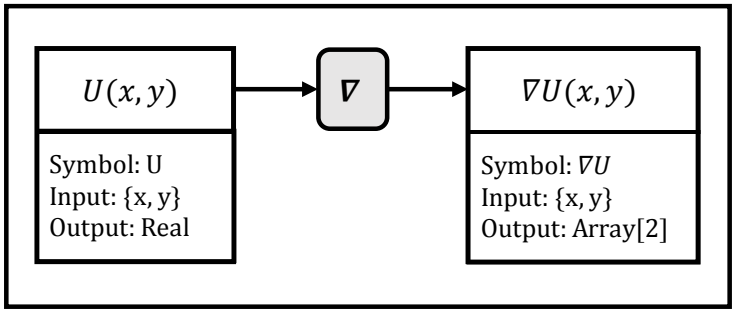


Figure 1: Description of the gradient law

Math objects in physics are tensor fields or tensors. Tensor fields are functions, in physics, they map from space coordinates to a real number or arrays of real numbers. In the first case, tensor fields are scalar functions while in the second they are arrays of functions. Each math object is defined with respect to a Euclidian space of dimension N, the space on which library elements are defined, and symbols for each dimension. (e.g., 2D space with $S = \{x, y\}$). We can restrict the space of a map by choosing a subset of the coordinate symbols. Math objects that have no input coordinates are tensors, which is a real number or an array of real numbers that only depend on the coordinate system but not the coordinates. Figure 2 shows type declarations (symbol, input, output) of some math objects and their representation. It

defines math objects for a two-dimensional space with $\{x, y\}$ as symbols for real number coordinates. For example the math object with symbol U is a scalar field that takes $\{x, y\}$ as input, and outputs a real number. Another example is the math object K, a constant, which is coordinate-independent (no input) and outputs a real number.

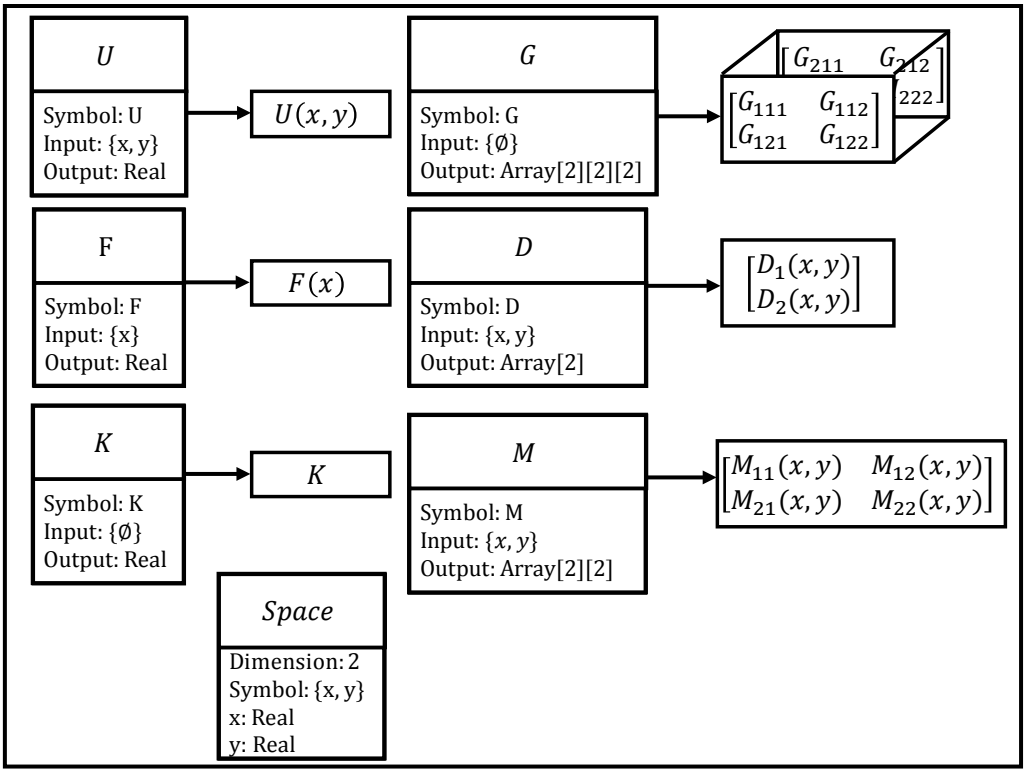


Figure 2: Declaration of math objects

Math objects are linked in a *math graph* (MG), e.g., one for the gradient law as shown in Figure 1. MGs are bi-partite and directed, which by definition are composed of two sets, one for math objects and another for operations linking two math objects. Each edge indicates how math objects relate to the operators at its ends. An arrow coming into an operator comes from a math object input to the operator, while an arrow going out of an operator leads to a math object output from the operator. The operator transforms one math object into another. Having operator as nodes in MGs enables representation of binary operators such as addition and multiplication. To support unambiguous references, each node is unique (identified, e.g., by a unique resource identifier on the web) and to support specialization, each has multiple attributes. Math objects can be augmented (*specialized*) with units and symbols following ISO [42] to produce *physics objects*. In Figure 3, the engineer entered the physical law that defines temperature gradient. For model completeness, the inputs of each math object have been added, which are the space or time coordinates or a combination of these.

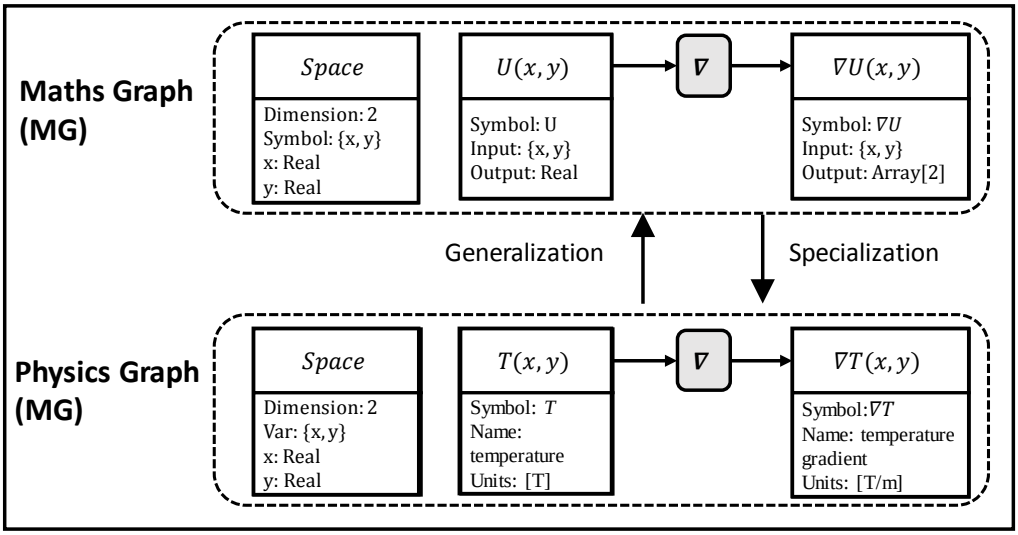


Figure 3: Specializing a physics graph (PG) from a math graph (MG)

As the number of relevant equations increases, they can be automatically combined when two equations, each a statement or triple linking two physical quantities by an operation, share the same physical quantity, as illustrated in Figure 4. Combination is possible only if each physical node is uniquely identified. For example, the Resource Description Framework [42], the web standard for linked data, could identify them with unique resource identifiers (URIs) and describe equations with two statements each, one linking a physical quantity to an operator, and another linking that operator to another physical quantity.

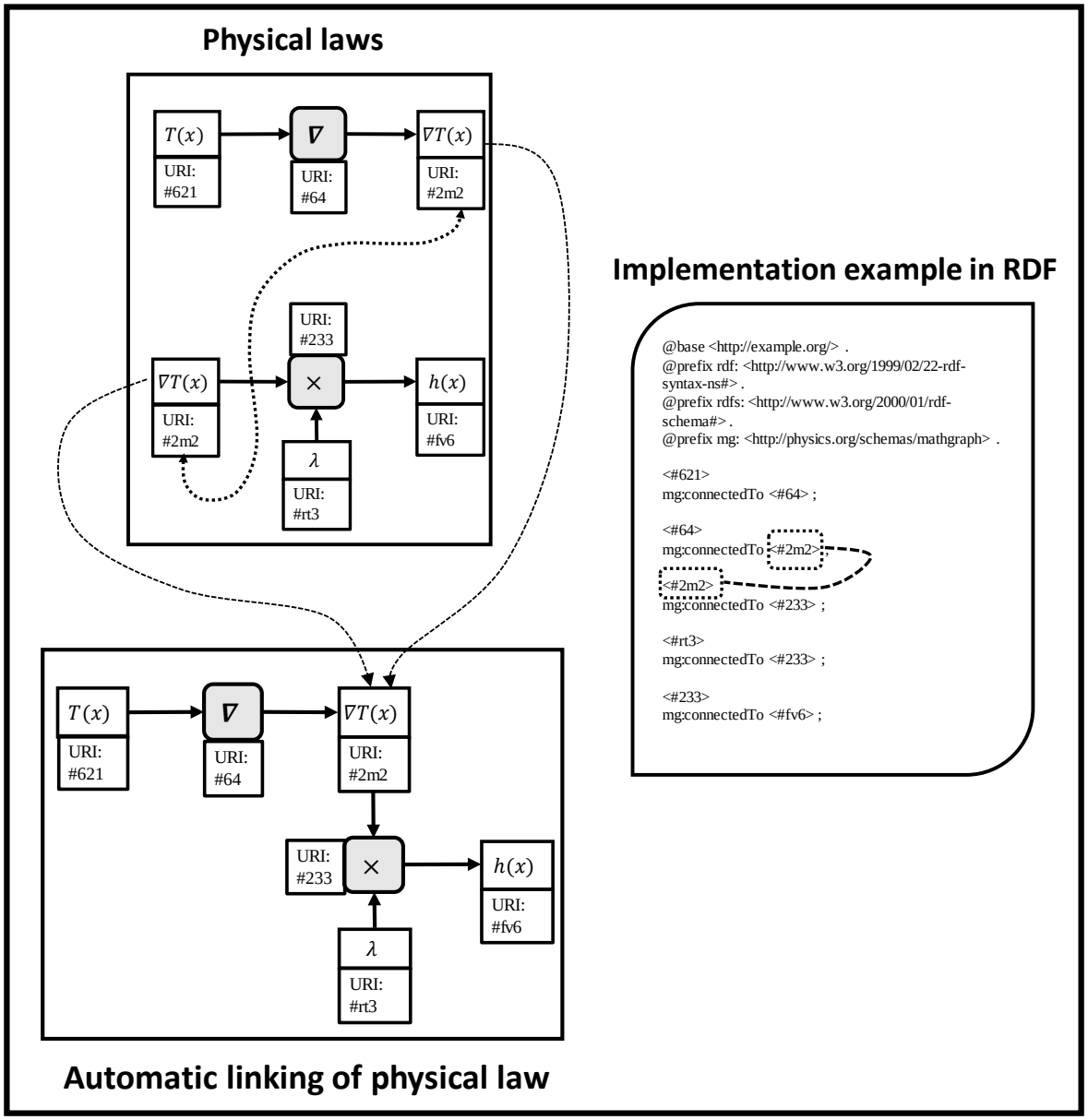


Figure 4: Automatic linking of physical equations

Combining equations, automatically or manually, produces a *physical graph* (PG), a combination of physical equations that capture the mathematical structure of a physics theory or model. We distinguish three layers of a PG as shown in Figure 5. The *functional program specification layer* defines the type of math or physics objects and operators, and the objects input/output for each operator (arrows). A path through a graph in the direction of arrows from one object to another is called a *functional program path (FPP)*. Objects between the start and end of a path are intermediary results. The second layer is the *expression layer* that declares a mathematical expression (e.g., e^{5x^2}) specifying

outputs in terms of inputs. The types of math and physics objects can be checked for consistency with the expression (e.g., gradient of scalar field produces a vector field). This facilitates expression processing when mathematical expressions, e.g., LaTeX [44], MathML [45] or expressions trees, are assigned to math or physics objects. Symbolic computation can produce the expression for a math object from the expression of another and the operation linking them, in the direction of the link. The *data layer* specifies input values mapped to output values following the rule defined by the expression, creating a function graph or plot.

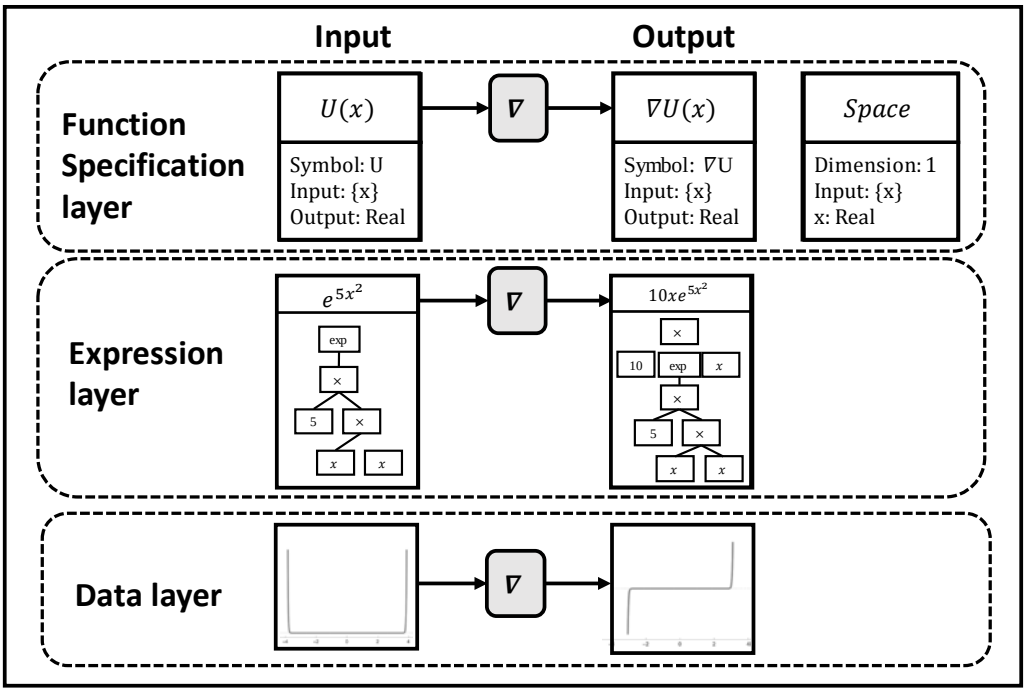


Figure 5: Layers of a math objects

b. Physics problems

After defining math and physics models, or choosing existing ones, the next step specifies problems to solve by declaring known and unknown pairs of math or physics objects in MGs and PGs. If a mathematical relation exists for a pair, then at least one FPP exists through the MG or PG with the two math or physics objects at the ends of the path. Absence of path means no solution is possible without modifying the graph. If a FPP has operations composed only of invertable functions, then an *inverse* FPP can be defined by replacing the operations by their inverses and reversing the arrows in the path.

When known and unknown are assigned as the start and end of a FPP, respectively, we call the path a *math evaluation graph (MEG) or physics evaluation graph (PEG)*. When known and unknown are assigned as the end and start of a FPP, respectively, and an inverse FPP exists, an MEG or PEG can be automatically generated for it as shown in the Figure 6.

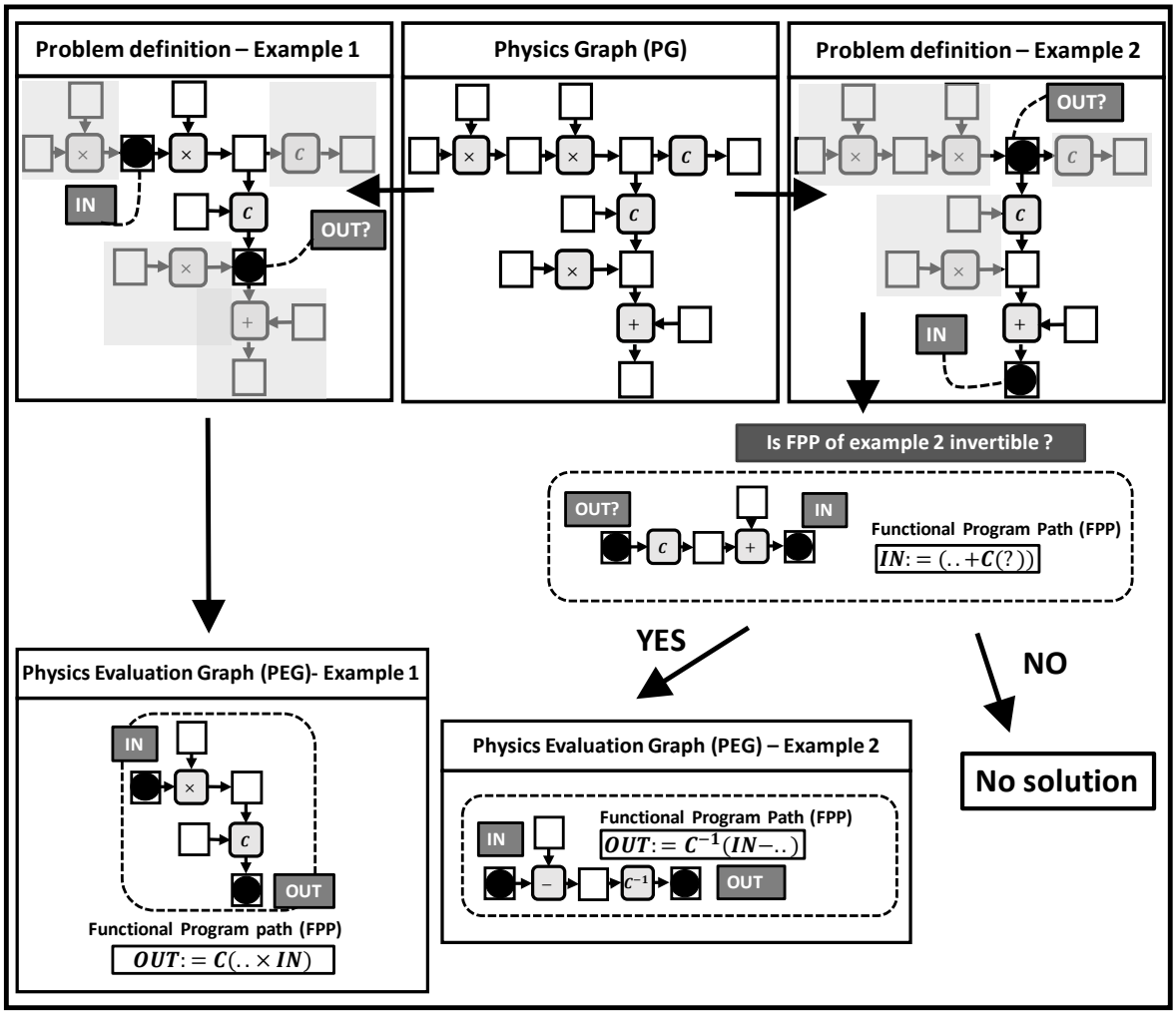


Figure 6: Solution finding in a physics graph

In physics, most laws are expressed in mathematical equations using differential operators, which can be inverted by adding boundary conditions. For example, in one dimension (e.g., motion on a line, see Figure 7), the top PEG connects a position/force pair, while the middle one inverts it by adding integration constants as boundary conditions for the anti-derivative operations. For example, the time derivative of position is velocity, on the left in the top PEG, but its inverse, integration, in the middle PEG, produces distance, rather than position. A specific position value, a boundary condition, is needed to get position from distance. Similarly, on the right, the time derivative of momentum is force, but the integral of force is impulse. A specific boundary momentum is added to get momentum from impulse.

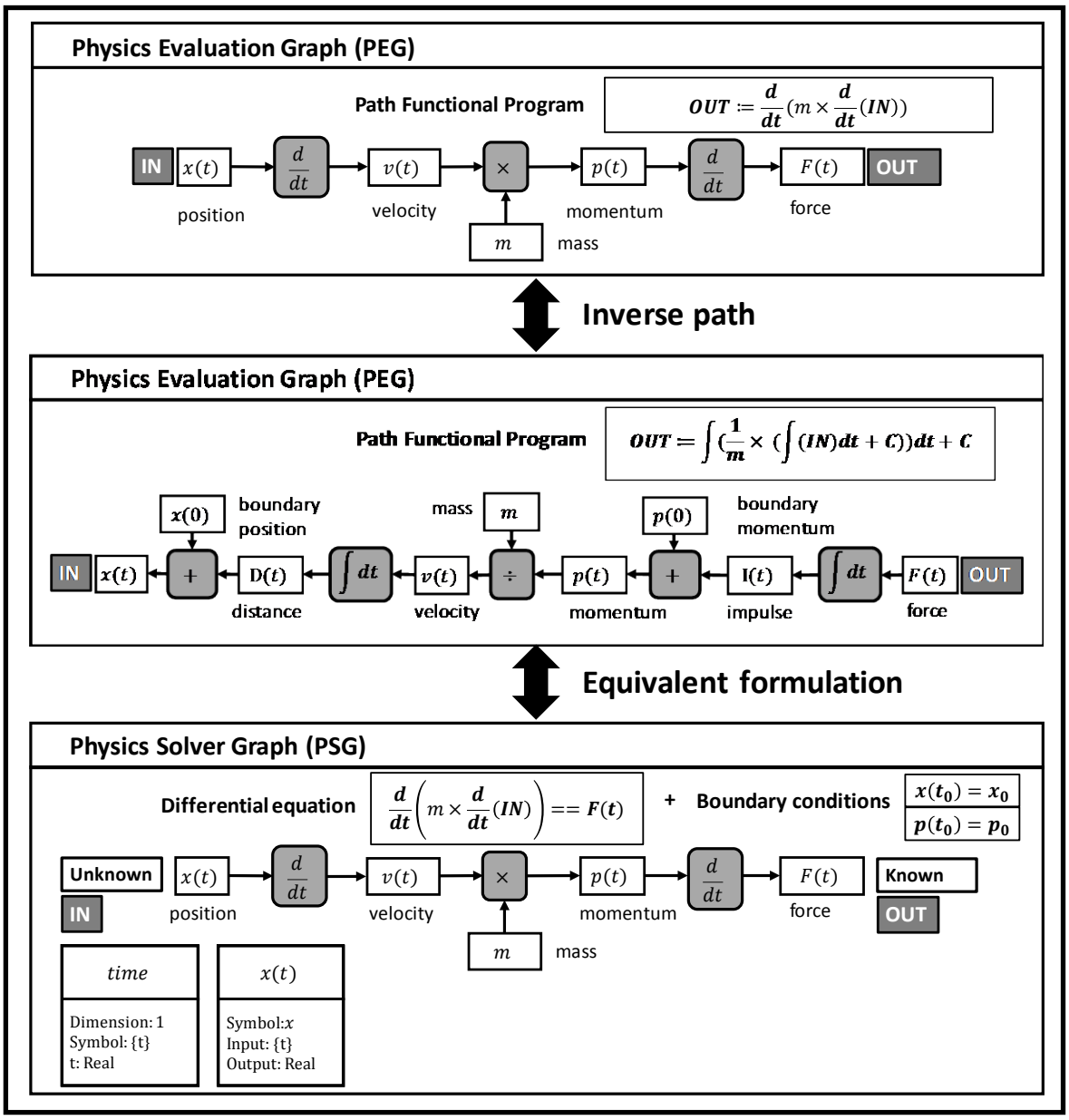


Figure 7: Inverse path and its equivalent formulation for ODEs

A more common representation of such problems is differential equations, which in one dimension, ordinary differential equations (ODEs), can be handled by symbolic solvers. When a PEG has integration operators and a known start object (e.g., force known, position unknown in the middle graph of Figure 7), its inverse PEG has differential operations and an unknown start object (e.g. position). Because the end object (e.g., force) is known and must be equal to the differential expression implied by the rest of the graph, the graph represents a differential equation. To be equivalent to the original integral PEG, we must add boundary conditions (e.g., the lower graph of Figure 7).

This is a *math solver graph (MSG)* or *physics solver graph (PSG)*. Paths through these graphs go from unknown to known under boundary conditions, representing differential equations under boundary conditions.

Most physical laws are expressed in differential form are represented as a differential operator linking two physical quantities. For multi-dimensional spaces, there is no analytic inverse to differential operators, therefore, analytic inverses of most FPPs with differential operators do not exist. The problem of finding the inverse can still be characterized by an MSG or PSG, see Figure 8. An approximate solution to the inverse FPP can be constructed either by using an equivalent formulation to the PDE (e.g. FEM) or directly discretizing the operators (e.g. FD or FV).

In our context, most graphs will have differential operators. The methodology consists of defining known and unknown object pairs, then finding a path connecting these objects. Most of the time there will be only one possible path. We can generate a MEG or PEG when the start object is known, or a MSG or PSG if the start object is unknown. The two graphs are the same except for the known/unknown choice for the start and end objects, and additional boundary conditions attached to the MSG or PSG, which will be detailed in subsection 4.c. MEGs and PEGs are used for evaluation problems or post-processing, while MSGs and PSGs are used to specify solver problems that involve ODEs, PDEs or systems of PDEs.

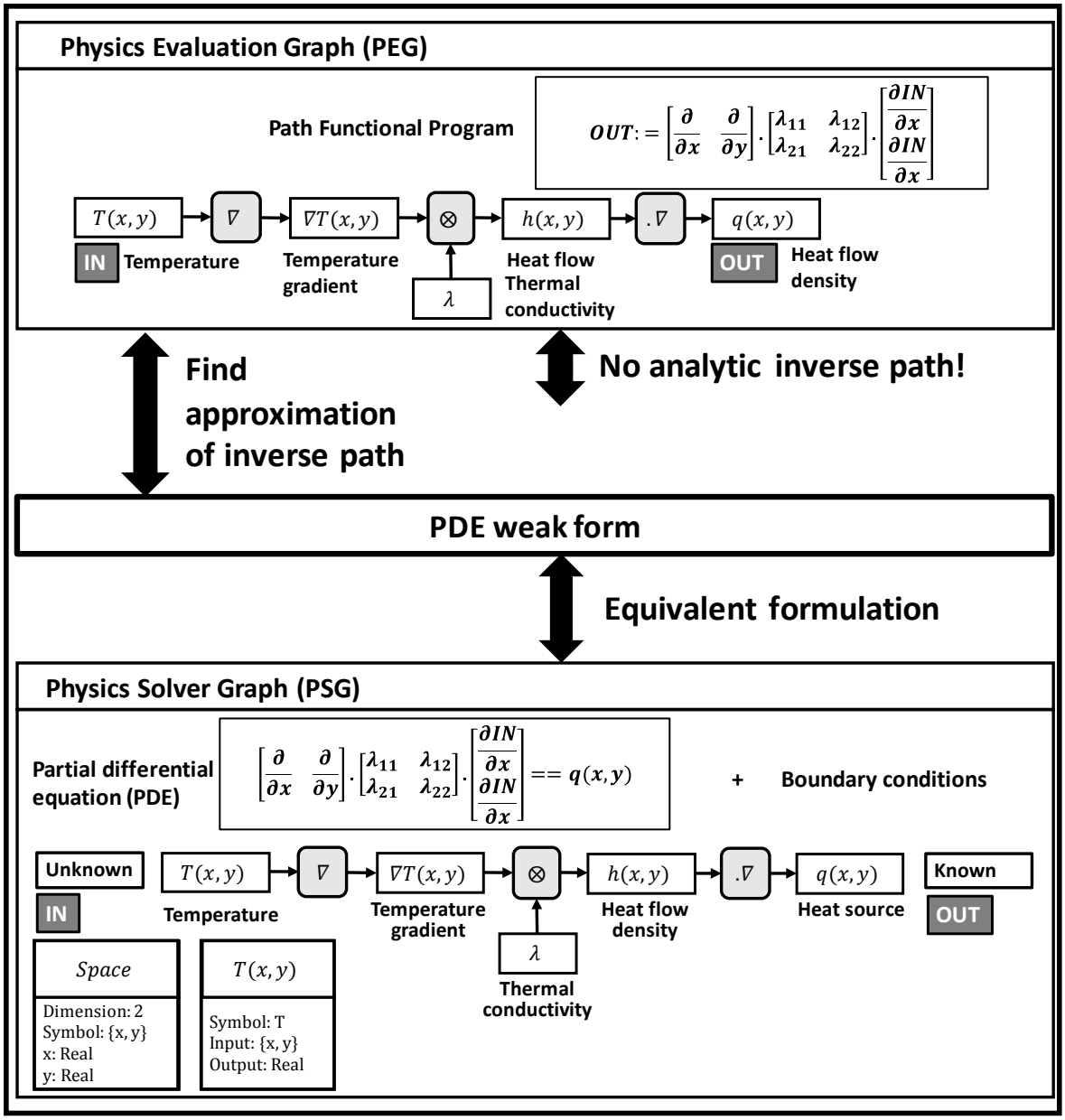


Figure 8: Inverse path and its equivalent formulation for PDEs

Characterizing or finding PDEs is useful to identify a problem, because problems are classified by PDEs, but they are difficult to understand without their derivation from physical laws. Furthermore, their boundary conditions are expressed in relation to the unknown, the start node of FPP. PDEs only represent the mathematical relationship between start and end objects in a FPP. Physical quantities and operators along the path are not in the PDE. In contrast, MSGs or PSGs give the derivation of PDEs. The graphs are more than equations, they capture modelling decisions leading to equations.

c. Boundary conditions

Finding a unique numerical PDE solution requires boundary conditions (BCs), along with other parameters (material, geometry). Specifying BCs is the task of mathematicians. Some BCs are well-known such as Dirichlet, Neumann, Robin, and Cauchy BCs. Mathematicians prove existence of solutions to PDEs under specified constraints. For solver input completeness, it would be very useful to automatically associate BCs during construction of PDEs or ODEs, but this would require solution existence proofs to be automated, which is not currently possible.

However, characterizing the physical quantity associated to BCs is still useful to engineering. We call this quantity, the *BC value type* (e.g., electric potential). PSGs give the derivation of PDEs and ODEs helping determine which physical quantities can be post-processed but requires BCs. Only adding their BC value types is possible. BC value types are associated directly to differential operators. Following Stoke's theorem,

$$\int_{\partial\Omega} w = \int_{\Omega} dw$$

each differential operator applied to a math object and integrated over a region has a corresponding boundary value type for the result of the integral of the math object along the boundary of the region. If we consider the triple (input math object, operator, output math object), then the BC type is linked to the input by the boundary integral, as shown in the bottom row of Figure 9. For curl and divergence, this introduces a third physical quantity. For gradient or differential, no additional physical quantity is introduced, the BC type math object is the input to the operator (integral of a single point value is equal to itself).

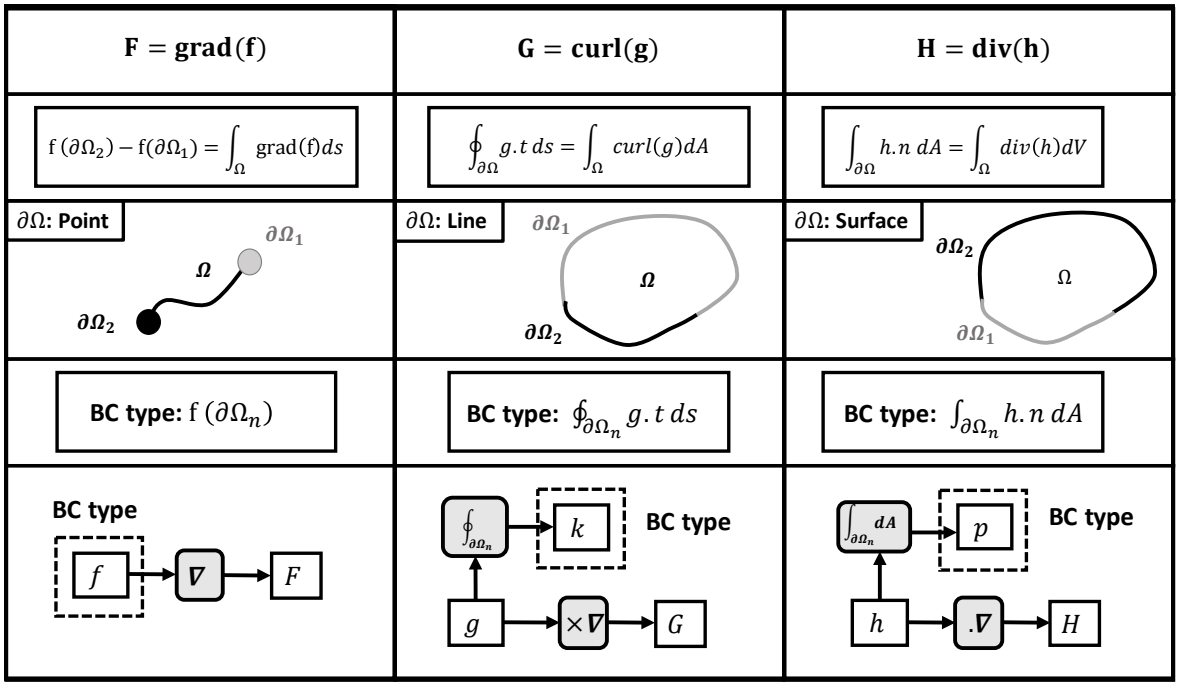


Figure 9: Stoke's law and corresponding BC types attached to differential operators

These BC types will be automatically added in the MG or PG whenever a differential operator is defined. They are important because they also correspond to physical quantities. For example, Figure 10 shows a thermal conduction PG, where heat flow is the BC type of the divergence operation applied to heat flow, and temperature is the BC type of the gradient operator. If we consider the PSG with temperature unknown and heat source as known, temperature and heat flow are the collected BC types which correspond to the Dirichlet BCs type and Neumann BCs type respectively. By automatically adding BC types, whenever a differential operator is used, we have a PG complete model.

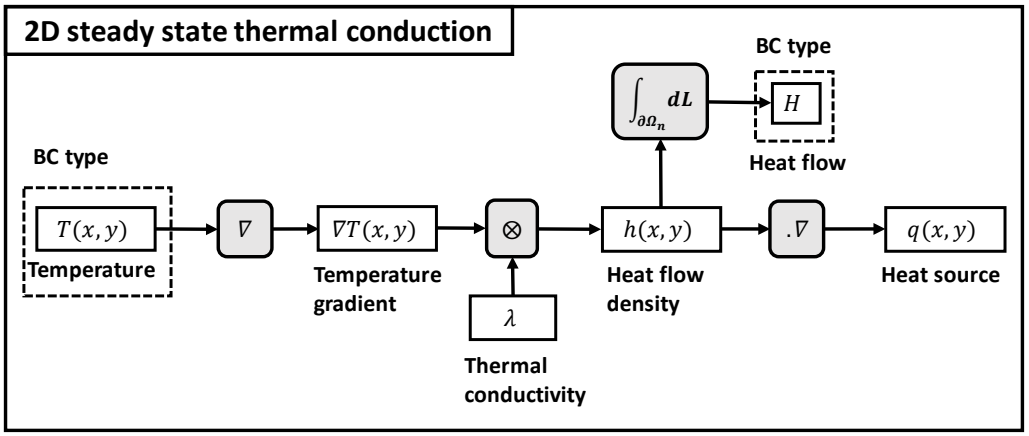


Figure 10: 2D steady state thermal conduction PG

d. Common mathematical structure

A *common mathematical structure* (CMS) is an MG that is shared by many PGs. Identifying CMSs is useful for reuse (e.g., assessing solver compatibility) and model understanding (e.g., from a mathematical perspective solving thermal conduction problems is the same as solving electrostatics problems). For example, FEA solves classical field theory problems. The CMS in Figure 11 is the basis of many domain-specific problems (e.g. 2D steady thermal conduction in Figure 10). Benefits of CMS will be illustrated in subsection 4.h.

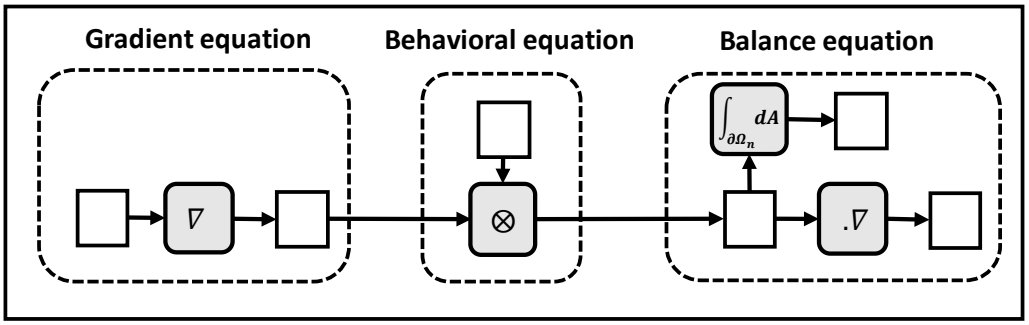


Figure 11: An example of a CMS of classical physics

e. Finite element specifications

As described in section 4.b, problems defined by an MSG don't have an inverse but an equivalent description the weak form. The discretization process consists in defining the problem in the integral form in order to find an approximation of the inverse path of the FPP. Finite elements (FE) are functions or tensor fields representing physical quantities on discrete topological elements of space. Degrees of freedom (DOFs) of finite elements are free or fixed variables that represent evaluations of physical quantities done on topological elements (point, line, surface, volume) . Functions or tensor fields of FEs, usually represented by polynomial (or tensor) functions, are rearranged as shape functions or shape fields, each fully defined, and each one multiplied by one DOF coefficient. There are as many shape functions as DOFs. For example, in electrostatics, on the left in Figure 12, a potential can be interpolated by a FE with 3 DOFs, specifically 3 electric potentials at discrete points in space as free variables. Another example, from thermal conduction, would be heat flow density interpolation, on the right in Figure 12. This time, normal components of heat flow density are integrated along each edge, leading to 4 DOFs or 4 heat flow as free variables.

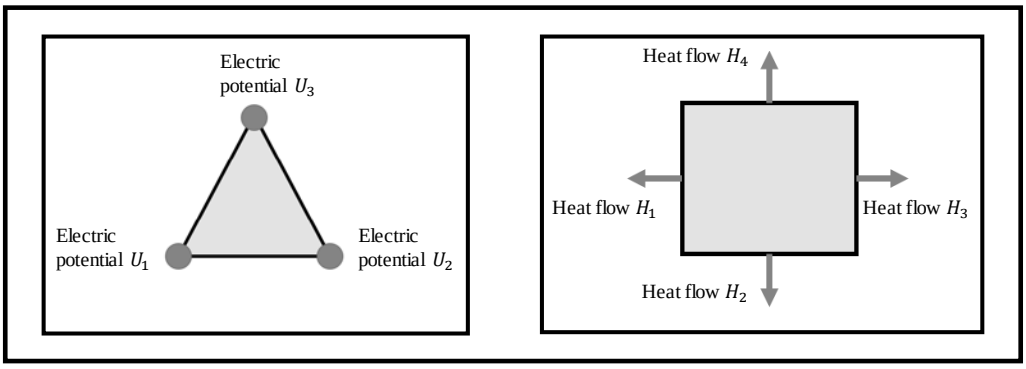


Figure 12: DOFs for electric potential and heat flow density interpolation using finite elements

Finding a formal FE description is complicated by the same finite elements being referred to as many different names in the literature. For example, a linear line element is also called a Lagrange line element. Engineering names, such as beam or bar element, are also ambiguous. A beam, for example, can have 4 degrees of freedom or 2 degrees of freedom., A finite element periodic table in [46] classifies these, but requires advanced mathematical knowledge to understand.

We developed a formal finite element specification (FES) [47] based on the generic finite element definition of Ciarlet [48] and the DOF type description as found in [49]. However, the triplet (geometry, DOF and basis space) is described with topology to provide compact description. A central aspect of

FES is associating DOFs to topological elements (point, line, surface, volume). A geometry Ω, like a triangle for example, can be partioned into a set of three lines $\mathbf{C}_1(\Omega) = \{\mathbf{L}_1\ \mathbf{L}_2, \mathbf{L}_3\}$ corresponding to disjoint surface boundaries; a set of three points $\mathbf{C}_0(\Omega) = \{\mathbf{P}_1\ \mathbf{P}_2, \mathbf{P}_3\}$, the edge boundaries. Subtracting the $\{\mathbf{C}_0(\Omega), \mathbf{C}_1(\Omega)\}$ union from Ω gives $\mathbf{C}_2(\Omega)$, the interior surface. Each member of $\mathbf{C}_2(\Omega)$, $\mathbf{C}_1(\Omega)$, $\mathbf{C}_0(\Omega)$ are sets of points; surface, edge, singleton (one member) point sets respectively. DOFs are defined by associating DOF types, e.g., point evaluation (PE), first derivative (FD), along with a natural number to $\mathbf{C}_0(\Omega)$, $\mathbf{C}_1(\Omega)$ and $\mathbf{C}_2(\Omega)$. For example $\{PE: 1\}$ associated to $\mathbf{C}_0(\Omega)$, written $\mathrm{D}_{\mathbf{C}_0(\Omega)} = \{PE: 1\}$, means point evaluation for each member of $\mathbf{C}_0(\Omega)$, therefore 3 and 4 PEs (DOFs) for triangles and squares respectively. $\mathrm{D}_{\mathbf{C}_1(\Omega)} = \{PE: 1\}$ means one PE at midpoint for each line point set of $\mathbf{C}_1(\Omega)$. In the triangle case, on the right in Figure 13, the evaluations are at 3 midpoints, one for each line. In general, for PE on lines, points divide lines into regular partitions, e.g., a midpoint divides a line into two equal parts. Multiple DOF types and number can be assigned to each $\mathrm{D}_{\mathbf{C}_n(\Omega)}$. A FE is explicity specified by composing all necessary $\mathrm{D}_{\mathbf{C}_n(\Omega)}$.

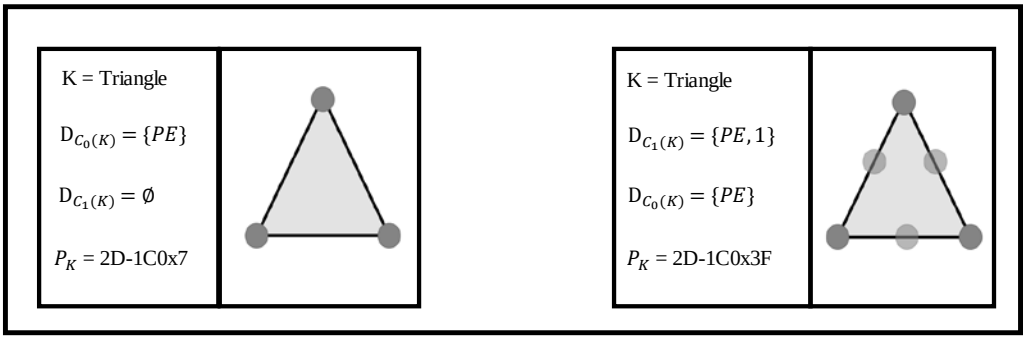


Figure 13: Example of FE definition using the FES

With geometry and DOF type specification of an FE, it is possible to calculate the number of DOFs and find an appropriate function to represent all DOFs. The FES can also encode polynomial functions. The set of all possible monomials can be lexicographically ordered, so the set of monomials appearing in a given polynomial can be mapped to a bit sequence, where each bit corresponds to the presence of a particular monomial in the polynomial. The resulting sequence is encoded as a hexadecimal number. The decoding of the hexadecimal number only requires writing the monomials for the space dimension in a graded lexicographic order, then assigning to each monomial position a bit that clarifies whether the monomial is used or not. For example, Figure 14 shows the encoding of a two-dimensional scalar polynomial (e.g. 2D-1C), more specifically a quadratic serenpedity functional space, functional space defined on a square, that uses all monomial terms except x^3, y^3 for the set of monomial $\{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3\}$. The main idea of the encoding is to provide an implementation description that is independent of

domain-specific ontologies (e.g., serenpedity). The current encoding method is on possibility as an example how domain-independent descriptions could work.

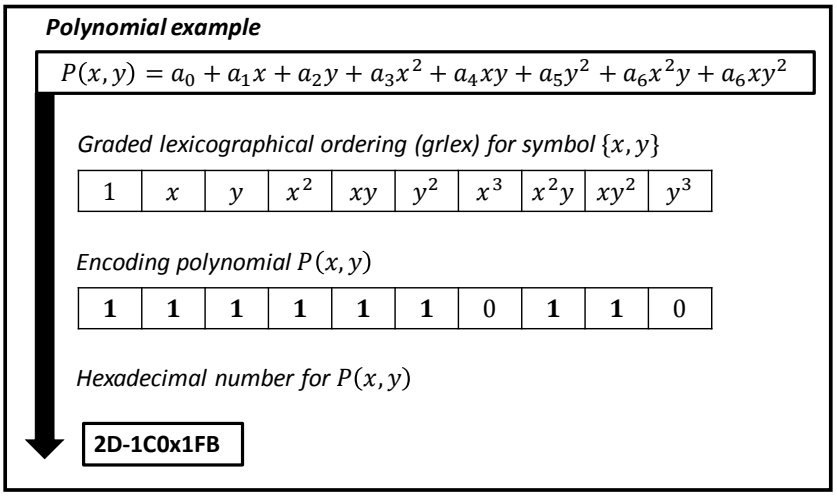


Figure 14: Example of functional space encoding

f. Library element

After creating PGs and PSGs, FEs are specified and associated to physics objects. FEs represent unknowns and space-dependent parameters (material or geometric parameters) if necessary. To illustrate the process, we consider the PG of a bar element shown in Figure 15, created using the kinematic equation that links strain to displacement; Hooke's law, which links stress to strain; and the balance equation that links body load to stress. Force is added automatically as a result of the divergence operator used in the balance equation. Then a problem is specified with a PSG, e.g., body load-displacement as known-unknown pair and displacements and forces as BC types. This is one of 12 evaluation and solver problems that can be defined using the same PG of Figure 15.

Next, the association of FEs to physics objects of the PSG, defines numerical graphs (NGs), by first declaring the math type of the physics object and then the FE. Many NGs can be defined from one PSG. For the body load/ displacement pair example, displacements could either be linear or quadratic FEs. Constant, linear, quadratic FEs to describe space-dependent sections are also possible, which in total, with displacement options, would produce 6 NGs. At this point, math objects consistency can be checked, verifying whether FEs choices (e.g., linear or quadratic) are consistent, helping understand numerical choices. In Figure 15, if distributed load is constant, choosing a linear displacement FE would be inconsistent, because balance equations would be broken and we would have only an approximation.

The principle of virtual displacement, which only requires kinematic admissible displacements, allows such approximations. However, inconsistent

physical equations are frequent when mixed principles introduce additional unknows in a PSG. In Figure 15, choosing quadratic displacement with constant body load yields an exact solution. These examples show that many numerical models can be defined using the same PSG, which itself is one of many options derived from a PG. This is reflected in the large number of library elements in software packages.

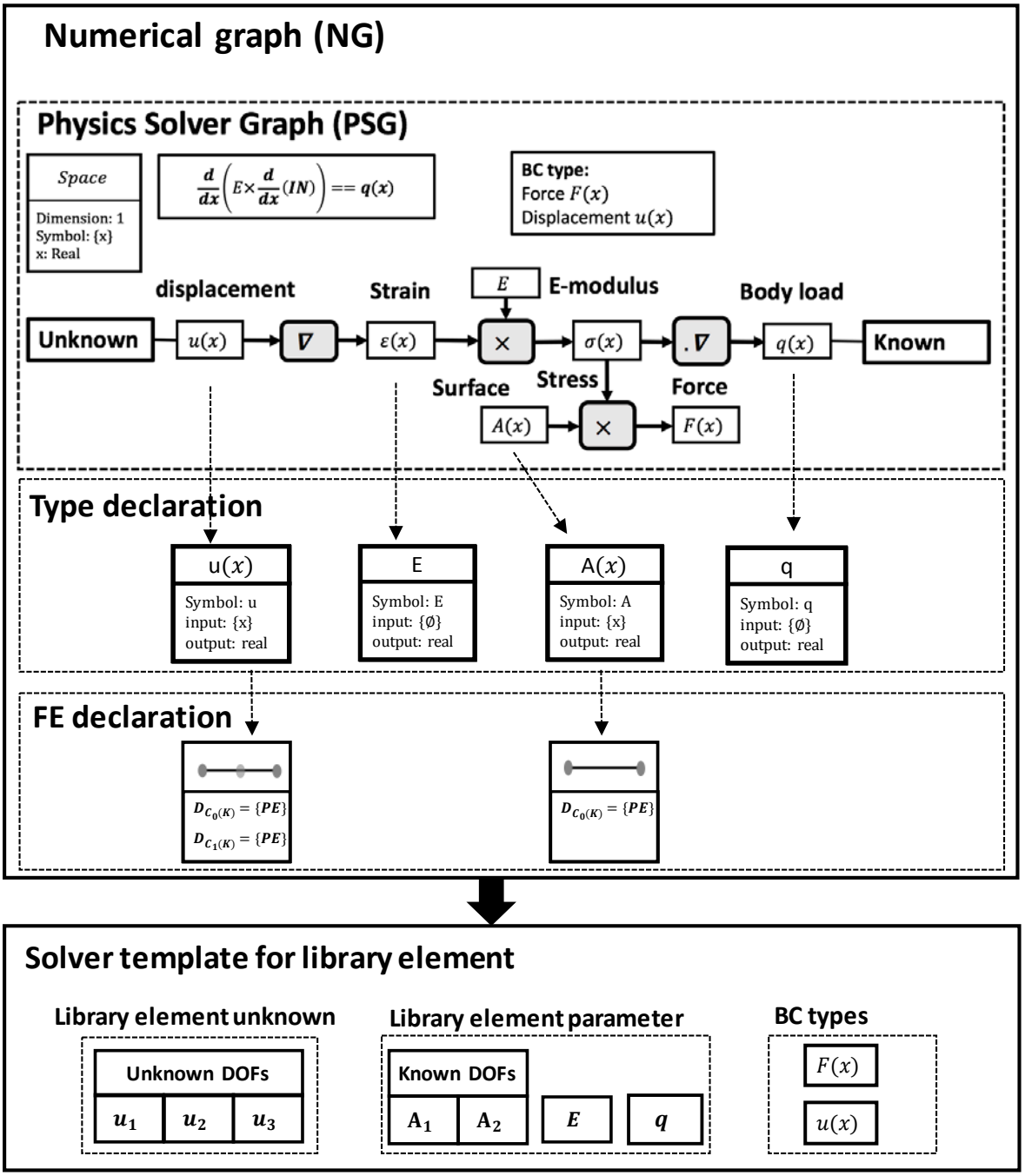


Figure 15: Example of a Numerical Graph for a bar

Specification of library elements leads to at most four kinds of digital artefacts: mathematical models as MGs (math graphs), physics models as PGs (physics graphs), physics problems as PSGs (physics solver graphs), and a subset of PGs and numerical models as NGs (numerical graphs).

NGs identify solvers (PDEs), collect all solver inputs (BC types, FEs, physical quantities with units and relations), information to construct the PDE weak forms, an integral formulation of the PDE, required for assembling FEA simulations. In addition, NGs, being specialization of PSGs, enable post-processing once the solution is found.

g. Simulation

In simulation driven design, traceability between design and simulation is essential, yet often obscured conventional practice. As an example of description granularity and information traceability, rather than optimize simulation, consider a part that is attached to another plate with two screws, as illustrated in Figure 16. A force is applied at the unattached end. The part is created using CAD with plate length as parametric variables. If meshing would natively occur in CAD environments instead of exporting CAD files to pre-processors or CAE tools, if the resulting mesh could be exchanged using a mesh standard that is serialized with popular file formats (e.g., JSON) and defined with schemas capable of representing BREP (boundary representation) mesh topology that associate metadata to topological elements, as described earlier, then CAD meta information would be preserved and reusable by any solver capable of reading such mesh standard. The mesh would store topological elements (node, edge, faces, volumes), the topology (how elements connect) and regions of interest (virtual topologies). CAD uses BREP and can associate design metadata to bounded shapes (e.g. material, assembly information), which can be translated with a native mesh process to virtual mesh topologies that keep metadata associations. The mesh could be exported with two separate information sets, one that describes the mesh topology and region topologies with keys, the other that associates keys with meta-data (e.g., material). In our example, the mesh has 4 virtual topologies; the plate (with material information and basis geometry for the mesh), the two holes with assembly information (use of screws) and a section of the boundary region (force). A new mesh with consistent metadata can be created when length or other parameters are updated in CAD.

The next step is selecting a library element, which requires FEA expertise and library knowledge. Sometimes a package for the required simulation is not available (e.g., license or absence of the library element, probably not the case for this problem). Searching for library package solutions requires reading documentation. Comparing these models is difficult because documentation standards vary and packages are not open enough. A standard based on the graph data structure previously presented would simplify the library search process. In the example above, search would be, e.g., for NGs with triangle

shapes containing physical objects such as force and displacement. In this example, starting with a PG of 2D elastostatics, the problem can be identified (here we know that force and displacement are the BC types), therefore a PSG connecting displacement to body load is required to have these two BC types. Then, a NG is created by associating a finite element to physical quantities or space-dependent parameters if necessary (e.g. space-dependent plate width).

The next step is to create a simulation model. In the example above, the NGs are associated to the plate domain and NG BC types (e.g., displacement and force) are associated to mesh regions (e.g., displacement to boundary holes). Once the simulation model is complete, solvers that follow this standard could propose their services. Another option is a new ecosystem of software that could support library creation and automatic generation of library code or links to existing code.

The final step is simulation. Each simulation run is defined by simulation parameters (material data, e.g., shear and bulk modulus, derived, e.g., from steel information, thickness value).

As a result of this approach we have 5 artefacts, design metadata, mesh data, NG (library model), simulation model metadata (attaching BC type to mesh data), and simulation data (referencing simulation value to NG parameter). In addition, the neutral mesh links design metadata to simulation model meta. A change in the CAD geometry (e.g., side length) does not impact the simulation model data. Furthermore, if an FEA engineer changes the FE geometry specification, for example changing triangular to square mesh, the CAD software could read this information and update the mesh accordingly .

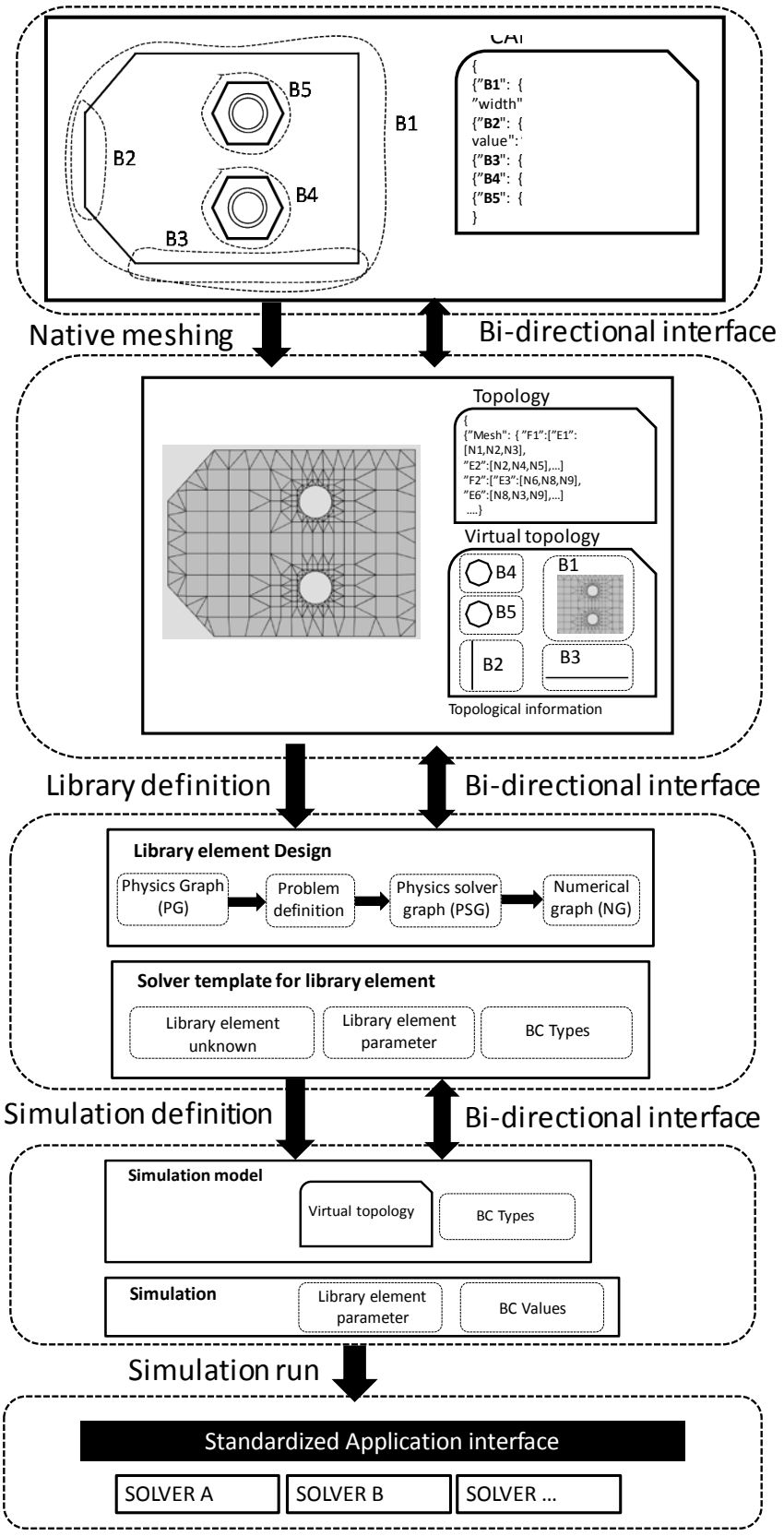
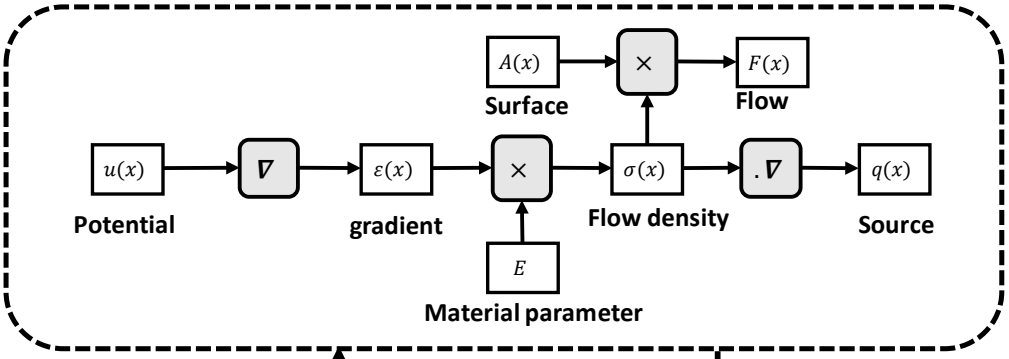


Figure 16: Standardized integration of solvers with assumption of a mesh and FEA library standard

h. Rapid extension of libraries

Identifying CMSs helps with reuse because of the specialization process that produces PGs from MGs, PSGs from PGs and finally NGs from PSGs. For example, a single MG handles most one-dimensional space-dependent PGs used for FEA. Such core models are essential to rapidly extend libraries. From this one MG, 10 PGs can be derived using correspondence tables. MG node objects are specialized to PG node objects by adding symbols and units. Physics objects and math objects with same mathematical relations in a graph are linked by specialization. This enables existing libraries to be rapidly extended to other physical domains when models share the same MS. Figure 17 shows three specialization examples (axial stress, themal conduction, electrostatics, see rows of table) from the 1D space-dependent core model.



Model name	Potential			Gradient			Flow density			Source			Flow			Material parameter		
	name	symbol	unit	name	symbol	unit	name	symbol	unit	name	symbol	unit	name	symbol	unit	name	symbol	unit
Axial Stress	Displacement	$\boldsymbol{u}$	[m]	Strain	$\boldsymbol{\epsilon}$	[-]	Normal stress	$\boldsymbol{\sigma}$	[N/m2]	Body load	$\boldsymbol{q}$	[N/m3]	Force	$\boldsymbol{F}$	[N]	Elasticity modulus	$\boldsymbol{E}$	[N/m2]
Thermal conduction	Temperature	$\boldsymbol{T}$	[K]	Temperature gradient	$\boldsymbol{\nabla T}$	[K/m]	Heat flow density	$\boldsymbol{h}$	[W/m2]	Heat source	$\boldsymbol{\sigma_t}$	[W/m3]	Heat flow	$\boldsymbol{H}$	[W]	Thermal conductivity	$\boldsymbol{\lambda}$	[W/(m.K)]
Electrostatics	Electric potential	$\boldsymbol{U}$	[V]	Electric field	$\boldsymbol{E}$	[V/m]	Electric displacement	$\boldsymbol{D}$	[C/m2]	Charge density	$\boldsymbol{\rho}$	[C/m3]	[-]	$\boldsymbol{\psi}$	[C]	Permittivity	$\boldsymbol{\epsilon}$	[F/m]

Figure 17: Rapid creation of libraries by CMS specialization

Many strategies are possible for library extension. A domain-independent strategy would be to create any possible NGs derived from MGs and MPGs, then specialize them to the corresponding physics, generating many PGs. Another would be to progressively create PGs, PSGs, NGs and identify PGs that share the same MG, then create models that can be reused for other domains.

In addition of being reusable, PGs can be composed with coupling equations, which can either connect PGs of different physics domains, leading in this case to multi-physics PGs; or connect PGs of the same domain, leading

to PGs describing more complex behaviors. For example, a thermal conduction PG and an axial loading PG, both sharing the same MG, can be connected by the thermal expansion equation. This coupling creates so-called thermo-mechanical models.

5. Conclusion

Virtual product development and simulation driven design decreases development costs by reducing expensive physical prototype testing. Early virtual design testing using CAE simulation, such as FEA, helps verify whether designs satisfy stakeholder requirements. Tracing digital models to requirements and system architecture is efficiently managed with model-based engineering, a methodology that facilitates integration and exchange of digital models between engineering disciplines. Standards are enablers of such model exchange.

Unfortunately, current trends show CAD and FEA packages are increasingly natively integrated in the same proprietary environment to leverage benefits of parametric CAD design [22]. This introduces two standardization challenges, one is related to translation errors between CAD formats (meta-data losses and geometric tolerance errors), the other due to informal FEA model description by standards. The first introduces substantial work when transitioning between CAD environments and between CAD and CAE, because of metadata losses. As an intermediate solution before CAx standards solve these issues, the development of a mesh standard that neutrally references topological regions could link PDM/CAD data to FEA data. The second is mainly due to the FEA software ecosystem, which delivers element libraries as compiled code packages. As a result, current standards only identify models with PLM data, such as software name, proprietary references to the compiled code and informal ontologies to characterize models, supporting only point to point integration.

An ecosystem shift is needed to solve the second challenge above, one that gives the same flexibility as time-only simulations and more transparency. Instead of setting inputs of compiled code templates, the starting point to model or generate FEA libraries should be an abstraction level understandable by all engineers, i.e., physics equations. At the numerical abstraction level, FEA engineers detail their FE choices. If both abstraction levels are connected, then FEA engineers and other engineering disciplines share a common understanding of their models. We show how these two abstraction levels connect by progressively creating and storing information as graph datastructures (MGs, PGs, PSGs, NGs). With numerical graphs (NGs), input template for solvers can be generated (PDE, BC types, material and geometry, FE choices). Solvers could be integrated into platforms though standardized

interfaces by providing a formal library element specification, or in the future new kinds of software solutions that model library elements and generate code.

Identifying common mathematical structure, core models, is essential for rapid extension and reuse of libraries between physical domains. In addition, physics graphs can be composed. We see two progressive usage scenarios, one that formalizes the description of FEA libraries (integration), the second that directly links physics models and numerical decisions to solvers with a standardized interface to run simulation with any solvers or even generate code (interoperability and transparency).

6. Acknowledgements

The authors thank Benjamin Urick, Stephen Langer, and Joseph Draper for their detailed comments and helpful discussion.

This work was performed under grant awards 70NANB16H174 and 70NANB18H192 from the U.S. National Institute of Standards and Technology. Identification of any commercial equipment and materials is only to adequately specify certain procedures. It is not intended to imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment are necessarily the best available for the purpose.

7. References

[1] BKCASE Editorial Board. (2017). The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.9.1 R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed 2019. www.sebokwiki.org.

[2] Object Management Group (September 2015). OMG Systems Modeling Language^{TM}, version 1.4: http://www.omg.org/spec/SysML/1.4.

[3] Object Management Group (March 2015). OMG Unified Modeling LanguageTM, version 2.5: http://www.omg.org/spec/UML/2.5.

[4] Modelica Association (2017) Modelica Language Specification, version 3.4

[5] Dadfarnia, M., Bock, C., Barbau, R. (2016). An Improved Method of Physical Interaction and Signal Flow Modeling for Systems Engineering: Conference on Systems Engineering Research.

[6] Hardwick, M, Clay, R., Boggs, P., Walsh, E., Larzelere, A., Altshuler, A. (2005), “DART System Analysis,” Sandia National Laboratory Report SAND2005-4647, Aug.

[7] International Organization for standardization (2016) ISO 10303-21:2016 Industrial automation systems and integration -- Product data representation and exchange -- Part 1: Overview and fundamental principles

[8] Ho-Le, K (1988). Finite element mesh generation methods: a review and classification, Butterworth & Co
[9] Geuzaine, Christophe & Remacle Jean-Francois (1997-2019). Gmsh – A three dimensional finite element mesh generator with build-in pre- and post-processing facilities – open source www.gmsh.info
[10] Beal, M. W. and Shephard, M. S. (1997), A general topology-based data structure. Int. J. Numer. Meth. Eng., 40: 1573-1596.
[11] Bruaset, Are Magnus & Langtangen, Hans Petter (1998). Diffpack: A software Environment for Rapid Prototyping of PDE Solvers
[12] Alnaes, M.S. & Blechta, J. & Hake, J. & Johansson, A. & Kehlet, B. & Logg, C., Richardson J.& Ring, J.& Rognes, M. E.& Well, G.N. (2015) The FEniCS Project Version 1.5: Archive of Numerical Software. vol. 3.
[13] Hecht, F. (2012). New development in FreeFem++. Journal of numerical mathematics, 20(3-4), 251-266.
[14] Van Rossum, G. (2007). Python programming language: http://www.python.org.
[15] A. Henderson, ParaView Guide, A Parallel Visualization Application. Kitware Inc., 2007.
[16] Schroeder, Will & Martin, Ken & Lorensen, Bill (2006), The Visualization Toolkit (4th ed.), Kitware, ISBN 978-1-930934-19-1
[17] W3C (September 2006) Extensible Markup Language (XML) 1.1 (Second Edition)
[18] Lee S.H., (2005). A CAD-CAE integration approach using feature-based multi-resolution and multi-abstraction modelling techniques: Computer-aided design, Elsevier.
[19] Beall, Mark & Walsh, Joe & Shephard, Mark. (2003). Accessing CAD Geometry for Mesh Generation: Proceedings of 12th International Meshing Roundtable. 33-42.
[20] Wassermann, Benjamin & Kollmannsberger, Stefan & Yin, Shuohui & Kudela, László & Rank, Ernst. (2018). Integrating CAD and Numerical Analysis: 'Dirty Geometry' handling using the Finite Cell Method.
[21] Junwahn, Kim & Pratt, Michael J. & Iyer, Raj & Sriram, Ram (2007). Data Exchange of Parametric CAD Models using ISO 10303-108, NISTIR 7433, NIST
[22] Hirz, Mario & Rossbacher, Patrick & Gulanova, Jana. (2017). Future trends in CAD – from the perspective of automotive industry. Computer-Aided Design and Applications. 14. 1-8. 10.1080/16864360.2017.1287675.

[23] Tierney, C. M., Sun, L., Robinson, T. T., & Armstrong, C. G. (2015). Generating analysis topology using virtual topology operators. Procedia Engineering, 124, 226-238
[24] Hirz, M. & Dietrch, W. & Gfrerrer, A. & Lang, J. (2013), Integrated Computer-Aided Design in Automotive development, Development processes, Geometric Fundamentals, Methods of CAD, Knowledge-Based Engineering Data Management. Springer
[25] Beal, M. W. and Shephard, M. S. (1997), A general topology-based data structure. Int. J. Numer. Meth. Eng., 40: 1573-1596.
[26] W3C (December 2017) The JSON Data Interchange Syntax: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
[27] International Organization for Standardization (2014) 10303-209:2014: Industrial automation systems and integration -- Product data representation and exchange -- Part 209: Application protocol: Multidisciplinary analysis and design.
[28] Long Archiving and Retrieval Engineering Analysis and Simulation Workgroup (2019) http://www.lotar-international.org/lotar-workgroups/engineering-analysis-simulation/
[29] MOSSEC (2019) - Modelling and Simulation information in a collaborative Systems Engineering Context - http://www.mossec.org
[30] Lipman, Robert (2018) – STEP File Analyzer User's Guide (Version 5) NIST Advanced Manufacturing Series 200-6. https://doi.org/10.6028/NIST.AMS.200-6
[31] L. Akers, Robert & Baffes, Paul & Kant, Elaine & Randall, Curtis & Steinberg, Stanly & L. Young, Robert. (1998). Automatic synthesis of numerical codes for solving partial differential equations. Mathematics and Computers in Simulation. 45. 3-22. 10.1016/S0378-4754(97)00082-7.
[32] Korelc, Joze. (1997). Automatic Generation of Finite-Element Code by Simultaneous Optimization of Expressions. Theoretical Computer Science. 187. 231-248.
[33] Alnæs, Martin & Logg, Anders & Ølgaard, Kristian & Rognes, Marie E. & Wells, Garth N. (2014) Unified Form Language: A domain-specific language for weak formulation of partial differential equations. Volume 40 issue 2, February 2014, Article No. .ACM Transactions on Mathematical Software (TOMS)
[34] Alnæs, Martin & Logg, Anders & Mardal, K-. A. & Skavhaug, O. & Langtangen, H.P. (2012). Unified Framework for Finite Element Assembly

[35] Balay, Satish & Abhyankar, Shrirang, & F. Adams, Mark & Brown, Jed & Brune, Peter & Buschelman, Kris & Dalcin, Lisandro & Eijkhout, Victor & Gropp , William~D. & Kaushik, Dinesh & Knepley, Matthew~G.& McInnes, Lois Curfman and Rupp, Karl and Smith, Barry~F. & Zampini, Stefano and Zhang, Hong, (2015). PETS, Users Manual, Argonne National Laboratory

[36] Van de Geijn, Robert & Goto Kazushige (2011). BLAS (Basic Linear Algebra Subprograms). Encyclopedia of Parallel Computing.

[37] Scilab (2019) XCOS, open source, https://www.scilab.org/about/scilab-open-source-software

[38] The MathWorks (2019), Simulink/Simscape Documentation, https://www.mathworks.com/help/simulink/, https://www.mathworks.com/help/physmod/simscape/.

[39] Tonti, Enzo (2013) The mathematical structure of classical and relatistic physics: Birkhauser

[40] Felippa, Carlos. (1994). A survey of parametrized variational principles and applications to computational mechanics. Computer Methods in Applied Mechanics and Engineering.

[41] Franklin H. Branin. The algebraic-topological basis for network analogies and the vector calculus. In Proceedings of the Symposium on Generalized Networks, volume 16, pages 453 – 491, Brooklyn, New York, 1966. Polytechnic Institute of Brooklyn

[42] ISO 80000-1:2009 (2009), Quantities and units, ISO

[43] W3C (2014) Resource Description Framework (RDF) 1.1 https://www.w3.org/RDF/

[44] LaTex project (2019) LaTex – a document preparation system - https://www.latex-project.org/about/

[45] W3C (2014) Mathematical Markup Language (MathML) version 3.0 2nd edition https://www.w3.org/TR/MathML3/

[46] Logg, A. & Arnold, D. (2014). Periodic table of finite elements: Siam News.

[47] Szarazi, Jerome & Bock, Conrad (2017) Integrating Finite Element Analysis with Systems Engineering Models, NAFEMS world conference, NAFEMS.

[48] Ciarlet, P. (2002). The finite element method for elliptic problems: SIAM.

[49] Logg A. & Al. (2012). Automated solution of differential equation by the finite element method. Springer.