Integrating Finite Element Analysis with Systems Engineering Models

Jerome Szarazi
(Koneksys, United Kingdom);

Axel Reichwein
(Koneksys, United States);

Conrad Bock
(National Institute of Standards and Technology, United States);

Abstract

In order to promote traceability, consistency, interoperability and better collaboration between systems engineering and Finite Element Analysis (FEA)-based simulation activities, we propose a tool-independent description of FEA models that integrates with the Systems Modeling Language (SysML), for future standardization. As technical systems become more complex, it is important to support traceability between systems engineering artifacts, such as requirements, and test cases, and corresponding FEA artifacts, such as FEA models, simulation conditions, and results.

While there is a standard for model-based systems engineering in the form of SysML, there is no standard description of FEA models. Existing FEA model descriptions are incomplete, tool-specific, informal or a combination of these. As a result, interoperability between FEA software applications is compromised, and communication between engineers is inefficient. A standard for the description of FEA models is difficult to develop, as the geometry, mathematics and physics of finite elements can vary greatly.

We propose a finite element mathematics specification based on recent works [1] and on the topological characteristics of finite elements that is formal, precise, and understandable to engineers. Mathematical expertise is still required to consistently set attributes of the specification but it can potentially capture new kinds of elements. The specification removes dependence on finite element names, which are sometimes inconsistent. We think that such a description is suitable for broad adoption among both FEA and systems engineers. We validate the new description of finite element mathematics by solving FEA problems using Python code we developed and demonstrate that these elements can be described in SysML.

1. Introduction and motivation

The motivation of this work is to better integrate FEA expert activities into the development lifecycle by integrating them with systems engineering modeling. We believe that it will benefit cross-disciplinary communication, traceability, model exchange, interoperability, model reusability, process knowledge and eventually process automation.

Efficient cross-disciplinary communication is essential for collaboration. System engineers have an overview of development activities by continually taking input from many stakeholders, making system-level decisions for improving the design, and communicating the results to cross-disciplinary teams for further design and validation. Based on system-level decisions, discipline-specific engineers create or reconfigure models for evaluating and validating new and modified designs. FEA models are used for validation across many engineering disciplines to simulate thermal, fluid, mechanical, electrical, magnetic problems or a combination of these. It is essential that FEA specialists, other engineering disciplines and system engineers share a common understanding of FEA models across disciplines to leverage a cross-disciplinary communication.

Sharing a common model representation across FEA tools will facilitate traceability from system requirements to FEA models. Product development is an iterative process where requirements change over time. The product specification may undergo many changes that are often decided at system level, but affect FEA models. To analyse the impact of a change, it is important that system requirements can be traced to FEA model artefacts. For example, a system cost reduction process may require a material replacement that should be translated into a material parameter change of the FEA model used to validate a safety requirement. By tracing links from requirements to models, system engineers can directly identify the impact of cost reduction on safety requirements and collaborate with FEA specialists to make better decisions. In addition, FEA engineers can better understand the context and value of their simulations in relation to customer needs.

Interoperability between applications is a major concern when selecting a tool, as it improves the data and information sharing within the same or with external organizations. It also avoids binding users to a single tool environment, making migration to another tool framework difficult. A standardized, tool independent FEA model representation will help to store and load a model from one application to the other. With standardized model interfaces, relationships and interdependencies between models of different abstraction can be defined. For example, this has been applied to integration of lumped parameter and systems engineering models [2].

Interoperability has two main levels, syntactic and semantic. The first facilitates the data exchange between tools by defining a common serialization format. The second helps to share context and define concept equivalences, enabling models to give the same simulation results across tools, and facilitate transformation between different kinds of tools. A tool independent FEA representation integrated with systems models will help to support syntactic and semantic interoperability between FEA tools, as well as model transformation between FEA tools and systems engineering tools.

Standards should help define unambiguous models that can be identified, understood and become reusable knowledge in engineering processes. For example, it is common that multiple simulations are required to validate a design. Validation procedures and other FEA tasks can be mapped into a workflow representing a chain of simulation actions performed on models, where model input and output can be connected. When the models used in workflows are interoperable, processes can be efficiently rearranged, process knowledge can be shared, and processes continuously improved.

This process knowledge can be exchanged with other disciplines. An example is the simulation of manufacturing processes using FEA. FEA is not only used for product design verification but also to simulate manufacturing processes. Typically, FEA calculates residual stress generated by forming, subtracting, or joining material in the assembly process. These simulations help to evaluate manufacturing process impact on design integrity by feeding-back residual stress results to the design. Another benefit is the improvement of simulation results and robustness by comparing simulation results to physical tests in a consistent manner. These comparisons can increase the quality of simulations.

Integrating FEA into system engineering requires an abstraction of FEA knowledge, leading to standards that will help overcome the heterogeneity of FEA model syntax and semantics. We start by evaluating existing standards in Section 2, then review FEA and some challenges in characterizing it abstractly in Sections 3 and 4, respectively. Section 5 proposes a new tool-independent specification for the mathematical portion of finite elements and provides a SysML model for it. Section 6 summarizes the paper and outlines future work.

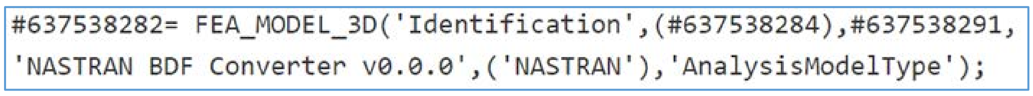
2. Status of standardization

SysML is an open graphical modeling specification that extends a subset of the Unified Modeling Language (UML) [3,4]. UML has been successfully used for architecting software applications, while SysML provides simple and efficient constructs to model a large variety of systems engineering problems. It is tool and methodology independent and supports model and data interchange via the eXtensible Markup Language [5].

SysML defines diagrams that illustrate system component relationships under different views, as well as semantics for its notation. SysML provides diagrams for requirements, structure, behavior and parametric relationships. These diagrams support specification, analysis, design, verification of systems. SysML supports syntactic and semantic interoperability for system engineering models.

For FEA activities, International Standards Organization 10303 (STEP) Application Protocol 209 edition 2 (AP209ed2) is the reference standard for multidisciplinary analysis and design product data with respect to interoperability, legacy data archiving and information reusability [6]. It integrates simulation data management, computer aided design, computer aided engineering and product data management. One of the main benefits of using a STEP standard is that FEA uses Computer Aided Design (CAD) geometries as input and STEP AP242 is a successful standard for CAD 3D geometry, assembly and Product Manufacturing Information interoperability [7]. The use of these two application protocols could support a seamless data exchange between design and analysis. Considering that the transformation of CAD data to FEA information is a major bottleneck in analysis activities, supporting interaction between design and analysis in a bi-directional manner would improve productivity.

Unfortunately, when reviewing STEP AP209, we noticed it mainly captures Product Lifecycle Management (organisation, date, etc) and geometric information. For the simulation part, AP209 defines a FEA model with a collection of non-constrained string entities (analysis type, creating software, finite element name, material…) leading to an informal FEA description, as in Figure 1. STEP AP209 provides only syntactic interoperability, not semantic interoperability. It cannot ensure the same simulation results across FEA tools or support integration with other kinds of tools. A more formal FEA description for data exchange is needed.



```
#637538282= FEA_MODEL_3D('Identification',(#637538284),#637538291,
'NASTRAN BDF Converter v0.0.0',('NASTRAN'),'AnalysisModelType');
```

Figure 1: Example of a FEA model definition in a P21 file following AP209

Interoperability between software applications and the communication between engineers, including system engineers and FEA engineers, is therefore compromised. In order to integrate finite element analysis with system engineering we need to characterize FEA and define a tool-independent abstraction that can then be modelled in SysML.

3. Overview of FEA

Finite element analysis is a numerical method for finding approximate solutions to partial differential equation or energy minimizing problems by solving linear algebraic equations for steady state or ordinary differential equations for dynamic problems [8]. FEA models require derivatives of functions of space (geometry) and time, as opposed to lump parameter systems, which only involve derivatives of functions of time.

FEA is a very powerful numerical method with attractive properties:

- *Modularity:* A domain specific physics problem can be modelled algebraically for a simple polygonal entity using interpolation functions. The resulting algebraic model can be stored as a software module or function, called a library element.
- *Reusability:* A library element can be reused in analyses of multiple systems.
- *Scalability:* Library elements are mapped to mesh cells partitioning system geometry. A complex system composed of multiple finite elements can be assembled and described by a global stiffness matrix.

FEA is typically used to solve multidimensional physics problems describing a system under the action of body and surface loads and kinematically constrained. The algebraic element relating loads to unknowns is the stiffness matrix K in case of static problems, and the mass matrix M and damping matrix D in case of dynamic problems.

The FEA procedure is simple and reproducible. It can be described by a workflow consisting of three steps, the pre-processing phase (mesh generation, material assignment and boundary labelling), analysis definition phase (finite element selection, model parameter setting, solver definition and output settings), and post-processing phase (visualization and data export) [9].

FEA applications have software architecture that follows the workflow. At the file level, a completed analysis will be composed of pre-processing files (CAD and mesh files), an analysis definition file or input file, and results files. Modern FEA software provides graphical user interfaces to support workflow. This is very helpful considering that many parameter settings are geometrical, such as boundary conditions definition. Another useful feature is the bi-directional associative interface providing a seamless integration of CAD and FEA data.

4. Challenges in characterizing finite elements analysis

Finding a unified description of FEA is difficult in part because a large number of possible FEA analyses result from the many options involved:

- Modern software provides a vast collection of FEA library elements to simulate mechanical, thermal, electrostatics, magneto-statics, fluid and electromagnetism. Multiphysics problems, which are systems consisting of more than one component governed by their own physics principles, are also supported.
- System studies can be time independent and will be evaluated by running a steady state analysis. For time-dependent systems transient analysis will be performed.
- System stability can be assessed by performing a modal analysis to find resonance frequencies or modes. Buckling analysis helps to avoid failure modes for solid system components under compression.
- The analyses above require material model definitions, which can be linear or nonlinear. Material models can also involve coupling in another physics, as for example in thermo-mechanical problems. For solid material under some large deformation, geometric nonlinearities will be considered, resulting in a re-meshing operation at each simulation step.
- From a numerical perspective, discretizing an analysis by choosing one or more discretization variables can affect simulation performance.
- Depending on the problem abstraction, a problem can be described in 1D, 2D or 3D. For symmetric systems or systems under rotation, cylindric or polar coordinate system for example can be chosen to simplify the analysis.
- To run an analysis on the discrete geometric system model (mesh), the finite element type, which is the core numerical element of the analysis, needs to be selected.

The total combination of these choices would provide a rough evaluation of the number of possible FEA models. If we consider a line element for example, we could apply 4 different physics and support simulation in 3 different dimensions, then we would have 12 combinations and therefore 12 FEA library elements. As a result, modern FEA applications involve a vast number of choices and results that are difficult to track.

The FEA process generates many artefacts that are difficult to organize. Multiphysics problems can be solved by custom code or tool vendor solutions having their own artefacts. Software vendors define their own FEA model referencing schemas, some of which are nonexplanatory while others try to incorporate some of the model parameter information. Expert know-how is required to find equivalences between models. As AP 209 does not provide any schema to classify finite elements, tools use their proprietary names for their APIs and for referencing their FEA models.

5. New proposed FE mathematics specification

In order to ensure efficient communication between system engineering and FEA, we need a standard convention to describe finite elements. It should include a compact notation that contains all information necessary to solve an FEA problems in a platform independent manner, by automatically generating shape functions, element stiffness matrices based on a problem specification. When characterizing an FE simulation, we had previously concluded that many kinds of FEA library element models are possible. A classification schema for these models is needed.

In this paper, we only discuss specification of FE mathematics, which we define as the mathematical objects representing polynomials with one or many variables over a geometric domain. These are the core objects of FEA simulation. This section outlines the challenges in finding such a description and propose a compact notation and SysML model that is unambiguous and understandable by most engineers.

a. Previous work on FE mathematics classification

As we started our search for a finite element mathematics description, we soon realised that most descriptions are ambiguous. The same FE-mathematics can be referred to as many different names in the literature. For example, a linear line element is also described as a Lagrange line element. Many FE mathematics are named after their discoverer(s) but this is sometimes not even clear as to which name to use. A Lagrange triangle can for example also be called a Courant element. Furthermore, the use of mathematician names does not provide any information for non-specialists.

Describing FE mathematics with engineering names, such as beam or bar element, is also ambiguous. A beam, for example, can have 4 degrees of freedom or 2 degrees of freedom. In the first case, the representation of 2 transversal displacements and 2 rotation angles requires a cubic polynomial or Hermitte element. In the second case, it requires only a Lagrange element to interpolate the moment. To find a description of FE mathematics, it is important to remove name dependencies, separate the physics from the mathematics, and classify or characterize FE mathematics by their properties.

FE mathematics classification work seems to have been first undertaken by Ciarlet. Modern finite element literature refers also quite often to his definition. In [10] an FE mathematics is defined as a triplet $(\boldsymbol{K}, \boldsymbol{P}, \boldsymbol{\Sigma})$ consisting of a geometric figure $\boldsymbol{K}$, a set of basis functions or basis space $\boldsymbol{P}$ and degrees of freedom $\boldsymbol{\Sigma}$ (DOFs). Although the original definition is simple, it seems that Ciarlet's objective was to describe families of FE mathematics. To further categorize them, Ciarlet introduced the typing of the geometry K. By doing so, he could establish relationships between a typed geometry K and the degree of the basis functions that are valid for any dimension. Although this additional categorization provided great insight on mathematical properties, it made the description of FE mathematics more complex.

Ciarlet's work was extended in [11], leading to a periodic table of finite elements covering differential forms of all possible degrees for each function space families. The table helps to understand the relationship between FE mathematics families and the application scope of the captured FE mathematics. The limitation is it does not classify some common elements.

b. New proposed FE mathematics specification

The new FE mathematics description is based on the generic finite element definition of Ciarlet and the DOF type description as found in [1]. However, the triplet (geometry, DOF and basis space) is described using topology. Furthermore, the new FE mathematics description is coordinate-independent, in contrast to Ciarlet's more complex coordinate-dependant description. Also, the description of DOFs is uncoupled from the finite element geometry. In contrast, Ciarlet's description is based on a coupling between DOFs, finite element geometry and basis space, leading to a less compact and more complex classification of finite elements.

To address the problem, we propose a compact, simple and unambiguous description of the mathematics of finite elements. This description reflects the mathematical properties of finite elements and provides all necessary information for implementation. The proposed FE mathematics specification can describe any finite element as it can be applied to any discrete geometry and can specify any basis space. The description is composed of a triple as defined in [10], but the information entities (geometry, DOFs, basis space) can be defined independently of each other and are thereby reusable. Mathematical expertise is required to consistently set attributes of the three entities in a finite element, but the specification can potentially capture new kinds of elements. The specification removes dependence on the finite element names which are sometimes inconsistent.

The new proposed FE mathematics specification is defined as follows:

- Geometry K (line, polygon, polyhedron, prism…)

- DOF sets on geometry K:

$$\mathbf{\Sigma}(K) := \{\mathbf{\Sigma}_{Cn}(K), \mathbf{\Sigma}_{Cn-1}(K), \dots, \mathbf{\Sigma}_{C0}(K)\}$$

  where $\mathbf{\Sigma}_{\mathrm{Cn}}(K)$ is the set of DOFs applied to all n-faces of the geometry (0-faces $\mathbf{C}_0$: vertices, 1-faces $\mathbf{C}_1$: edges, 2-faces $\mathbf{C}_2$: faces, 3-face $\mathbf{C}_3$: volumes).

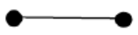
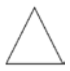
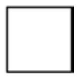
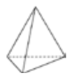
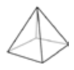
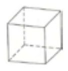
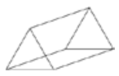
  A DOF is defined by a type and a count which specifies the number of evaluated requirements on one n-face of K.

- Basis functions set or basis space $\boldsymbol{P}$, defined by a string

Geometry type

The table below show a non-exhaustive list of geometries used in FE-mathematics and their respective string encodings.

Table 1: Listing of common geometric figures used in FEA and their corresponding encoding string

Geometry	Geometry name	Encoding string
	Line	L
	Triangle	T2
	Square	S
	Tetrahedron	T3
	Square pyramid	SP
	Cube	C
	Triangular prism	TP

Degrees of Freedom type (DOF type)

The set of all possible DOFs $\boldsymbol{\Sigma_f}$ types is given by:

$$\Sigma_f = \{p(\boldsymbol{x}): \boldsymbol{PE};\ \nabla p(\boldsymbol{x}): \boldsymbol{FD}\ ;\ \frac{\partial^2 p(\boldsymbol{x})}{\partial^2 \boldsymbol{x}}: \boldsymbol{SD}\ ;\ \ldots.\}$$

where $p(\boldsymbol{x})$ is a scalar or $\boldsymbol{p}(\boldsymbol{x})$ a vector-valued polynomial describing physical quantities. To facilitate reading, acronyms are used for literal operator definition as in [1], such as PE for point evaluation, see the table below.

Table 2: Definition of DOF types and corresponding encodings according to [1]

DOF type	**Definition [1]**	**Symbol**	**Mathematical expression of the linear form and shape function definition**
Point evaluation	Point evaluation of a function P at a vertex of the geometry	PE	$p(\boldsymbol{x})$
First derivative	Point evaluation of all first derivatives of the function P at a vertex of the geometry	FD	$\nabla p(\boldsymbol{x})$
Second derivative	Point evaluation of all second derivatives of the function P at a vertex of the geometry	SD	$\frac{\partial^2 p(\boldsymbol{x})}{\partial^2 \boldsymbol{x}}$
Directional derivative	Evaluation of the directional derivative of the scalar function v in the direction n at the point x.	DD	$\nabla p(\boldsymbol{x}).\boldsymbol{n}$
Tangential component	Evaluation of the vector-valued function v in the tangential direction along line of the geometry at the point x	VT	$\int_{\boldsymbol{0}}^{\boldsymbol{l}} \boldsymbol{p}(\boldsymbol{x}).\boldsymbol{t}\,\boldsymbol{ds}$
Normal component	Evaluation of the vector-valued function v in the normal direction along a face of the geometry at the point x	VN	$\int_{\boldsymbol{0}}^{\boldsymbol{l}} \boldsymbol{p}(\boldsymbol{x}).\boldsymbol{n}\,\boldsymbol{ds}$
Interior moment	Interior moment degrees of freedom; that is, degrees of freedom defined by integration against a weight function over the interior of the domain K	IM	$\int_K \boldsymbol{p}.q\,dS$

Basis space P

The FE mathematics specification also defines basis space $\boldsymbol{P}$**.** The geometry K, the DOFs $\boldsymbol{\Sigma}(K)$ and the basis space $\boldsymbol{P}$ are interdependent attributes. In [10] and [11], FE mathematics is further categorized by defining a family of basis space P. Its definition is abstract and difficult for engineers to read.

To facilitate implementation of FE mathematics, the specification provides a mechanism to capture the set of basis functions $\boldsymbol{P}$ using lexicographic ordering of monomials, following many symbolic computing programs that generate monomials this way. The method consists of creating a dictionary of all possible monomials combinations or words. These generated words are then ordered and indexed. We chose graded lexicographic ordering, which first sorts by total degree, then lexicographically. As an example, the set $S_2 = \{1, x, y, xy, x^2, y^2, \ldots\}$ of generated monomials can be ordered by $grlex$: $\ldots y^2 \succ xy \succ x^2 \succ y \succ x \succ 1$, following graded lexicographic ordering. In general, the representation of the monomials of the basis space $\boldsymbol{P}$ would only require using a subset of this ordered list. To represent this subset, we set 0 or 1 depending on whether a monomial word is used or not. A bit string to capture the monomials of the basis space P is then created following the graded lexicographic ordering. For example, the basis space $P_1 = \{1,\ x,\ y,\ xy\}$ used for a square can be represented as 11101. This bit string can be further compacted into the hexadecimal number 0x1D.

To complete the FE mathematics specification, we need to define the dimension. To create monomial words, we use predefined alphabet letters. This monomial alphabet is composed of a set of letters or symbols where one of the symbols represents a dimension. The set of generated monomials is therefore dimension dependant. In our examples, the letters $x,\ y,\ z$ are used and represent space dimension. The set $S_1 = \{1, x, x^2, x^3, \ldots\}$ represents one dimensional monomials and S_2 would be the set of two-dimensional monomials. The space dimension is prefixed with ND where N is the space dimension. Depending on the field type, a function space can have one or many components. (scalar polynomial field, vector polynomial field, …) The component index is specified by NC where N is the component index. The basis space $P_1 = \{1,\ x,\ y,\ xy\}$ would be encoded as in the table below.

Table 3: Example of scalar polynomial encoding

Space dimension	Field component	Basis space code
2D	1C	0x1D

The encoding depicted in Table 3 (2D-1C-0x1D) can be translated into a symbolic math program capable of translating it to a human readable form. To illustrate we decode 2D-1C0x1FB. Again, the “2D” gives the dimension, which means the basis functions use two variables (x and y, see below). A set of monomials can be generated and then ordered following graded lexicographic ordering. The “1C” indicates that we have only one component therefore representing a scalar polynomial. The “0X1FB” is a hexadecimal number which can be translated into a bit word declaring what monomial to use in the ordered list. The resulting basis space is $P_2 = \{1, x, y, x^2, xy, y^2, x^2y, xy^2\}$, which is used for an FE mathematics of type “serendipity element” as described in [12].

Table 4: Reading basis functions from basis space code 2D-1C0x1FB

monomials set S_2	x^5	xy^2	1	x^2	x^3y	....				
Graded lexicographic ordering	1	x	y	x^2	xy	y^2	x^3	x^2y	xy^2	y^3
0x1FB	1	1	1	1	1	1	0	1	1	0
Basis functions	1	x	y	x^2	xy	y^2	0	x^2y	xy^2	0

The main benefit of this representation of basis space $\boldsymbol{P}$ is to provide all necessary information for a straightforward implementation, without requiring abstract mathematical knowledge. The basis space $P_1$for example is referred to as a tensor space and is constructed by computing the tensor product of the basis space $B_1 = \{1,\ x\}$ and the basis space $B_2 = \{1,\ y\}$, which requires special expertise [13]. Using the representation above, the implementation of FE mathematics in custom code does not require knowledge about constructing spaces of basis functions.

Application examples

Figure 2 shows an example of a mathematical finite element used for beam calculation. This would read: “$\boldsymbol{\Sigma}_0(K)$ defines a point evaluation and a first derivative requirement applied to each 0-face (vertex) of the line geometry K”. This is a compact definition of a 1D-Hermitte element. Aggregation of physical information (transversal displacement, rotation) or geometric information (dimension, Cartesian…) can extend the specification.

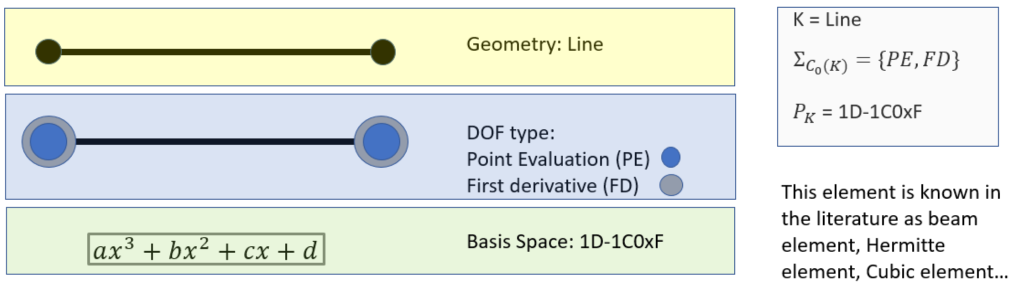


Figure 2: Application example for a Hermitte element

Figure 3 shows another FE mathematics example used in thermal simulation. FE mathematics specification describes it as a triangle geometry K composed of two DOFs sets $\mathbf{\Sigma}_0(K)$ and $\mathbf{\Sigma}_1(K)$ where $\mathbf{\Sigma}_0(K)$ defines point evaluation applied to each 0-face of K and $\mathbf{\Sigma}_1(K)$ defines one point evaluation applied to each 1-face (edge) of K.

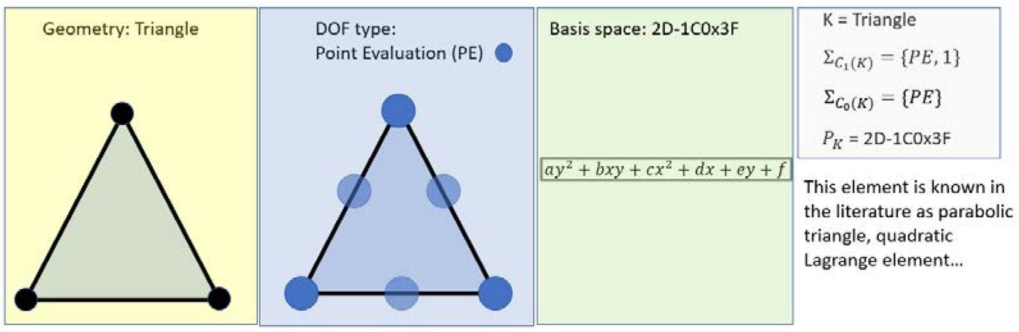

Geometry: Triangle
DOF type:
Point Evaluation (PE) ●
Basis space: 2D-1C0x3F
$ay^2 + bxy + cx^2 + dx + ey + f$
K = Triangle
$\Sigma_{C_1(K)} = \{PE, 1\}$
$\Sigma_{C_0(K)} = \{PE\}$
P_K = 2D-1C0x3F
This element is known in the literature as parabolic triangle, quadratic Lagrange element…

Figure 3: Application example for "a parabolic triangle"

The specification can potentially capture new elements. For example, the requirements previously defined for the triangle of Fig 3 could be reused and applied to any geometry, such as the hexagonal prism in Fig 4.

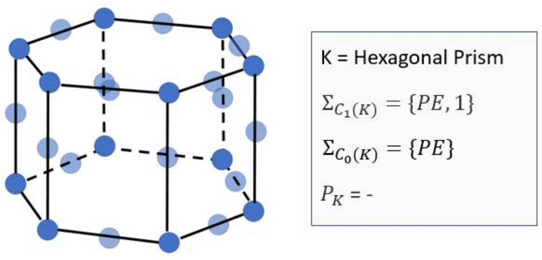
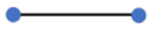
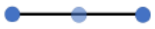
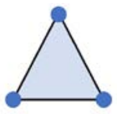
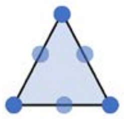
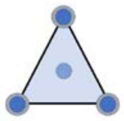
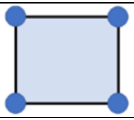
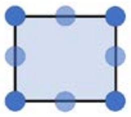
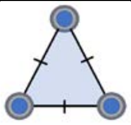
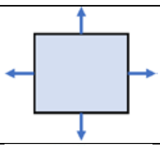
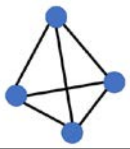


Figure 4: Application example for "a parabolic triangle"

Additional examples and their respective encodings are shown in the table below.

Table 4: Application examples of the FE specification

Figure	Common name	Compact description	Code
	Linear element, Lagrange element, bar element….	K = Line $\Sigma_{C_0(K)} = \{PE\}$ P_K = 1D-1C0x3	L C0PE 1D-1C0x3
	Quadratic element, Lagrange quadratic	K = Line $\Sigma_{C_1(K)} = \{PE, 1\}$ $\Sigma_{C_0(K)} = \{PE\}$ P_K = 1D-1C0x7	L C0PEC1PE1 1D-1C0x7
	Linear triangle, Lagrange linear triangle, courant element, T3…	K = Triangle $\Sigma_{C_0(K)} = \{PE\}$ P_K = 2D-1C0x7	T2 C0PE 2D-1C0x7
	Quadratic triangle, T6…	K = Triangle $\Sigma_{C_1(K)} = \{PE, 1\}$ $\Sigma_{C_0(K)} = \{PE\}$ P_K = 2D-1C0x3F	T2 C0PEC1PE1 2D-1C0x3F
	Hermitte triangle	K = Triangle $\Sigma_{C_2(K)} = \{PE, 1\}$ $\Sigma_{C_0(K)} = \{PE, FD\}$ P_K =2D-1C0x3FF	T2 C0PEFDC2PE1 2D-1C0x3FF
	Linear square element, Lagrange square element	K = Square $\Sigma_{C_0(K)} = \{PE\}$ P_K = 2D-1C0x1D	S C0PE 2D-1C0x1D
	Serendipity quadratic element	K = Square $\Sigma_{C_1(K)} = \{PE, 1\}$ $\Sigma_{C_0(K)} = \{PE\}$ P_K = 2D-1C0xFB	S COPEC1PE1 2D-1C0x1FB
	Argyris element	K =Triangle $\Sigma_{C_1(K)} = \{DD, 1\}$ $\Sigma_{C_0(K)} = \{PE, FD, SD\}$ P_K = 2D-1C0xFFFFF	T2 C0PEFDSDC1DD1 2D-1C0x1FFFFF
	Raviart-Thomas	K = Square $\Sigma_{C_1(K)} = \{\mathrm{VN}, 1\}$ P_K = 2D-1C0x3-2C0x5	S C1VN1 2D-1C0x3-2C0x5
	Linear Tetrahedron, 3-simplex linear Lagrange element…	K = Tetrahedron $\Sigma_{C_0(K)} = \{PE\}$ P_K = 3D-1C0xF	T3 C0PE 3D-1C0xF

Representation in SysML

The new specification provides a compact and precise description of FE mathematics that can be modeled in a SysML block definition diagram. SysML block definition diagrams are based on UML class diagrams, where blocks are based on UML classes. A FE mathematics block composes a geometry block, one to many DOF set block(s) and a basis space block. Blocks are detailed by properties that specify types for their values. The model uses integers for space dimension and DOF count on a face. The basis space definition is of type string. The DOF type, the face type and the geometry are defined by enumeration literals, which are an ordered set of names. The enumeration for geometry, for naming of the faces and DOF type are shown in the diagram. Property values can be produced (derived) from other information, indicated by a forward slash ('/'). Such properties could be the number of faces composing the geometry or the total count of DOFs.

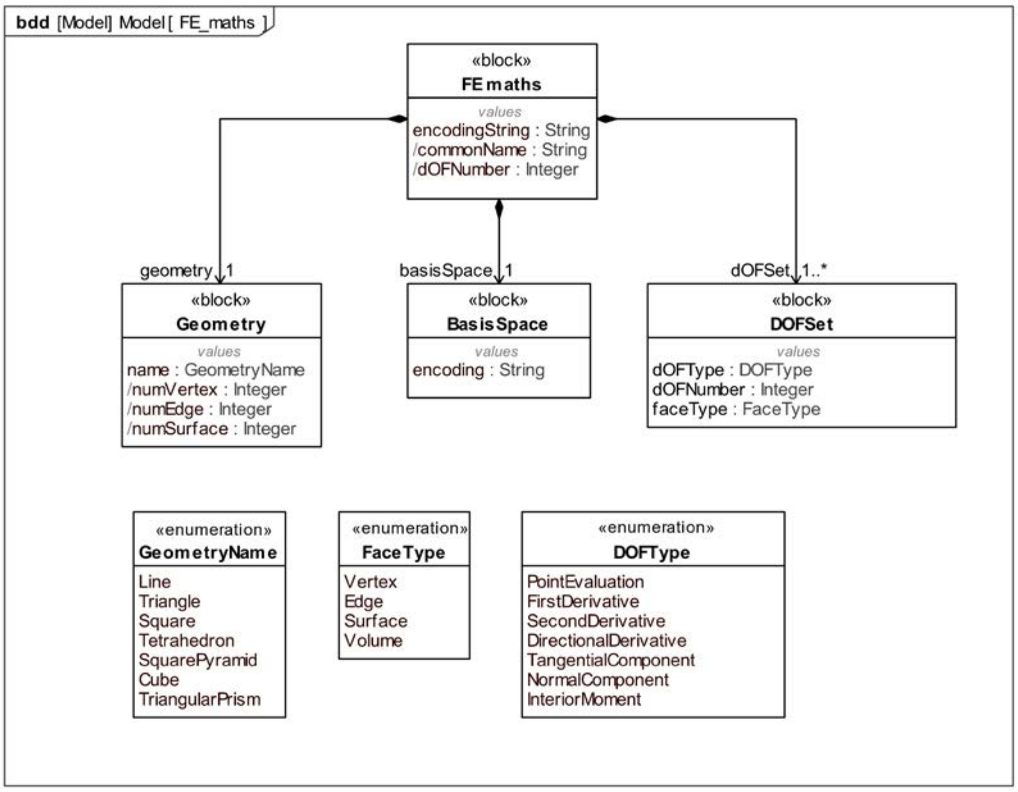


Figure 5: SysML block definition diagram of the FE mathematics specification

The FE mathematics block definition diagram is an abstract model reflecting the general description of FE mathematics. An instance diagrams is derived from it and shows objects specified by property values.

The serendipity element of table 5 is captured in the instance diagram of figure 6 which specifies the geometry type as Square, basis space with the encoding “2D-1C0x1FB”, the two DOFs set “C1PE1” and “C0PE” with their respectives DOFs type, DOF number and the facetype to which they are applied. FE mathematics objects can therefore be captured and documented in instance diagrams.

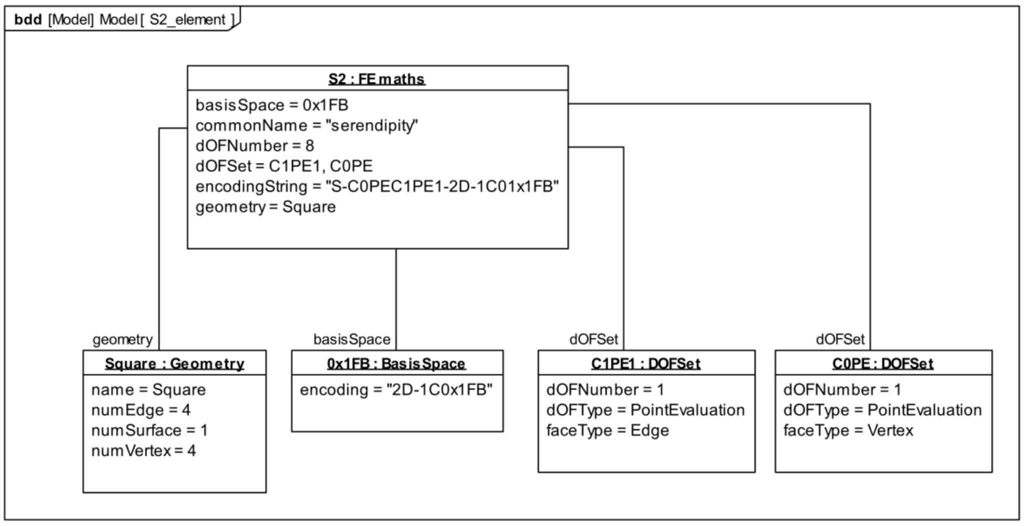


Figure 6: SysML diagram of an FE Mathematics instance

Validation and testing

The FE mathematics specification is used to calculate shape functions space, which form the basis of the dual space of linear forms represented by the DOFs. To find shape functions, the procedure transforms the basis space P into the shape function space, which is another basis, where one DOF, or by extension a discrete physics value (temperature, displacement, etc) located at a point or distributed on a point set of the geometry is represented by a shape function.

This procedure requires a definition for the reference coordinates of the geometry, which can be arbitrarily chosen. Depending on the coordinate choice, the shape functions will be different. The coordinate information is therefore implementation dependent and set by the FE code developer. This is one of the reasons why the FE mathematics specification has been defined in a coordinate independent manner. If shape function information is shared, then the reference coordinates of the geometry need to be communicated.

To validate the specification, we developed our own symbolic code using the Sympy library, a python based library for symbolic mathematics [14,15]. Another implementation of symbolic FE mathematics can be also found in [16]. Symbolic code promotes interoperability by supporting many serialization formats. In Sympy, a mathematical expression is stored as an object which can be “printed” into many formats such as a string, LaTex expression, MathML, some image formats such JPEG and help with the documentation process, as illustrated in the figures below. Other methods can transform the symbolic mathematical expression into code such C, Fortran, Javascript, Theano or python. We implemented our python code to calculate shape functions based upon the specification information. The figure below shows the information model for a linear triangle.

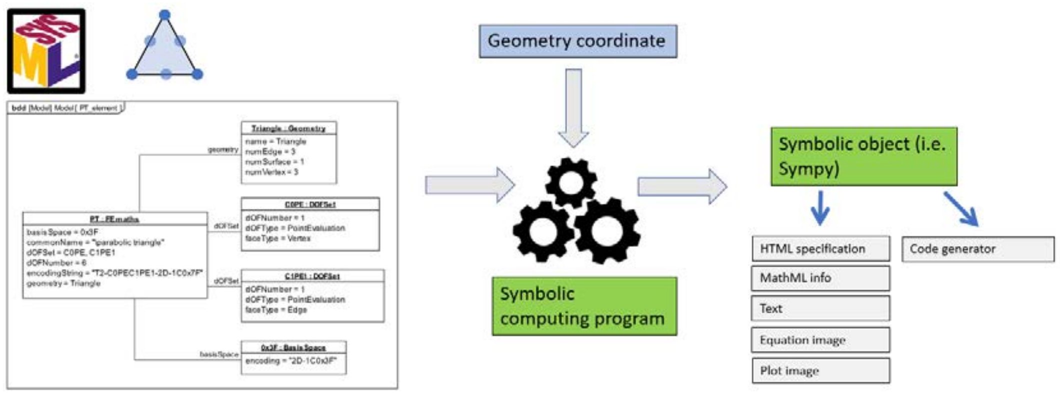
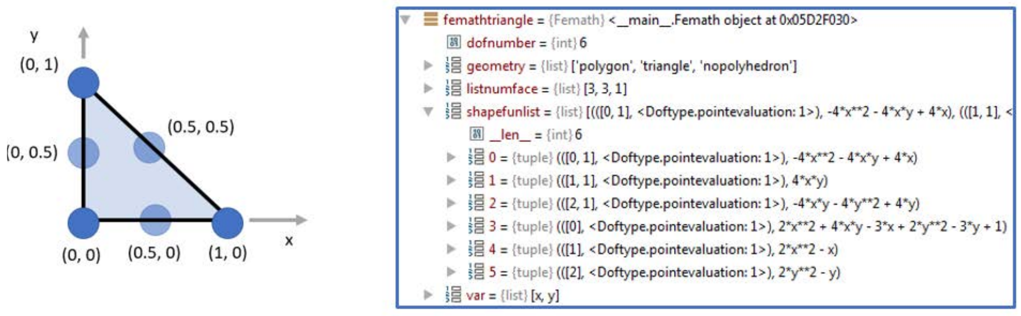


Figure 7: Computation of shape function’s symbolic expression



```
femathtriangle = {Femath} <__main__.Femath object at 0x05D2F030>
    dofnumber = {int} 6
    geometry = {list} ['polygon', 'triangle', 'nopolyhedron']
    listnumface = {list} [3, 3, 1]
    shapefunlist = {list} [(([0, 1], <Doftype.pointevaluation: 1>), -4*x**2 - 4*x*y + 4*x), (([1, 1], <
        _len_ = {int} 6
        0 = {tuple} (([0, 1], <Doftype.pointevaluation: 1>), -4*x**2 - 4*x*y + 4*x)
        1 = {tuple} (([1, 1], <Doftype.pointevaluation: 1>), 4*x*y)
        2 = {tuple} (([2, 1], <Doftype.pointevaluation: 1>), -4*x*y - 4*y**2 + 4*y)
        3 = {tuple} (([0], <Doftype.pointevaluation: 1>), 2*x**2 + 4*x*y - 3*x + 2*y**2 - 3*y + 1)
        4 = {tuple} (([1], <Doftype.pointevaluation: 1>), 2*x**2 - x)
        5 = {tuple} (([2], <Doftype.pointevaluation: 1>), 2*y**2 - y)
    var = {list} [x, y]
```

Figure 8: Python object of a “parabolic triangle” for specific reference geometry coordinates

Shape functions are the basis of the discretization process and therefore an essential step in the FEA procedure. Physics variables are expressed with shape functions and DOFs. This expression is then implemented into the weak form, an equation, minimizing the approximation error or the energy of the physics problem, depending on the approach.

The resulting expression leads to an algebraic system relating DOFs to loads. In the case of steady state problems, stiffness matrices relate DOFs to loads. The figure below shows a stiffness matrix symbolically computed for a finite element describing a line element under the influence of axial loads in a 2D space. A Sympy module can transform symbolic expressions of stiffness matrices into lambda functions, by stating coordinates as inputs and stiffness matrix components as outputs. Lambda functions can calculate numerical values very fast. Using a lambda function of the stiffness matrix, a problem composed by many line elements can be assembled for a solver.

$$\begin{bmatrix}
\frac{(x_0-x_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(x_0-x_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} \\
\frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(y_0-y_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(y_0-y_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} \\
-\frac{(x_0-x_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(x_0-x_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} \\
-\frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & -\frac{(y_0-y_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(x_0-x_1)(y_0-y_1)}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}} & \frac{(y_0-y_1)^2}{(x_0^2-2x_0x_1+x_1^2+y_0^2-2y_0y_1+y_1^2)^{\frac{3}{2}}}
\end{bmatrix}$$

Figure 9: Symbolic stiffness matrix

6. Conclusion and future work

Model interoperability is essential to improving collaboration between FEA and other engineering disciplines. This is facilitated by systems engineering models acting as information hubs, which need to integrate FEA information. A successful integration of systems engineering models and FEA improves communication, traceability, model reusability and process knowledge in the development process. A key enabler for successful integration is standards. While there is a standard for systems engineering models (SysML), we found that standards for FEA are imprecise and ambiguous. A unified description of FEA is difficult in part because a large number of possible FEA analyses result from the many options involved.

To overcome this difficulty, we first decomposed FEA information by describing FE mathematics in a compact way that does not require abstract mathematical knowledge. The representation assigns degrees of freedom to topological entity types and provides a systematic method to capture basis functions. We tested the compact description on most FE mathematics and then created a SysML model to represent it. We tested this model by implementing symbolic code to generate shape functions. Some of these shape functions were also used to generated symbolic stiffness matrices.

We believe that the new FE mathematics representation can be consistently exchanged, providing all necessary information for implementation and FE mathematics model reusability. Implementation using symbolic computing further supports model reusability and the serialization of symbolic expressions into many formats promotes interoperability. As a result, we have an unambiguous and compact specification of FE mathematics that is readable by most engineers and can potentially capture new elements.

The code is currently limited to using the string encoding of the FE specification, which will be addressed by a translator between SysML and the code, still under development. The specification of FE mathematics is a first step to consistently represent FEA problems. Extension of the new finite element specification to cover FEA physics is on-going work. The ultimate objective is to describe FEA models using SysML by extending the FE mathematics specification to capture relationships between systems engineering and FEA-related simulation information. This FEA information model should provide all information necessary to assemble a problem for a solver.

Acknowledgements

We would like to thank Peter Denno and Li Ma for input to this paper.

This work was performed under grant awards 70NANB14H251 and 70NANB16H174 from the U.S. National Institute of Standards and Technology.

Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

References

[1] Logg A. et Al. (2012). *Automated solution of differential equation by the finite element method*: Springer.

[2] 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.

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

[4] Object Management Group (March 2015). *OMG Unified Modeling Language^{TM}, version 2.5*: http://www.omg.org/spec/UML/2.5.

[5] W3C (September 2006) Extensible Markup Language (XML) 1.1 (Second Edition)

[6] ISO 10303 STEP AP 209 ed2 (2013). *Composite and metallic structural analysis and related design*: ISO – International Organization for Standardization.

[7] ISO 10303-242 STEP AP 242 "Managed model based 3D engineering: ISO – International Organization for Standardization

[8] Hulbert, G. (1992). *Time finite element methods for structural dynamics*: International journal for numerical methods in engineering. vol. 33, 307-331.

[9] Bathe, K-J. (1996). *Finite element procedures*: Prentice hall.

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

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

[12] Arnold, D., Awanou, G. (2011). *The Serendipity Family of Finite Elements*: The journal for the Society for the foundations of Computational Mathematics.

[13] McRac, A., Bercea, G-T., Mitchell, L., Ham, D., Cotter, C. (2014). *Automated generation and symbolic manipulation of tensor product finite elements*: SIAM Journal on Scientific Computing.

[14] Van Rossum, G. (2007). *Python programming language*: http://www.python.org.

[15] Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. (2017). *SymPy: symbolic computing in Python*: PeerJ Computer Science.

[16] 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.