Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 688–696

Procedia Computer Science

Supporting the Perpetuation and Reproducibility of Numerical Method Publications

Antônio T. A. Gomes[a], Diego Paredes[a,*], Frédéric Valentin[a]

[a] *National Laboratory for Scientific Computing (LNCC)*
Av. Getúlio Vargas, 333 – Quitandinha – Petrópolis, RJ – 25651-075 – Brazil

Abstract

The development of new numerical methods is of great importance in computational science. Due to their many appealing properties, Finite Element (FEMs), Finite Volume (FVMs) and Finite Difference (DFMs) methods are of particular interest, with a very large number of journal articles devoted to them. Unfortunately, these methods can be time consuming to implement when one would like to explore certain numerical properties or make comparisons with other methods. In this paper, we propose a new computational environment in which the specification and implementation of such methods introduced in journal articles can be perpetuated and the experimentation with them can be made reproducible and easily compared with related work.

Keywords: High-performance computing, Web-based review process, Numerical methods, Rapid prototyping

1. Introduction

Mathematical models based on differential equations constitute a large part of the modeling of natural phenomena. Being intrinsically complex, analytical solutions to these equations are only available in very specific cases. As such, approximate solutions must be obtained computationally using some type of numerical method. Researchers and practitioners seek a method bearing in mind a certain set of desired characteristics, including:

- **Flexibility.** The method should be easily adapted to new computational situations, such as the resolution of a model on varied and possibly complex domains.
- **Accuracy.** The solution to the method should be as near to the true solution as necessary.
- **Performance.** The use of parallelization is highly desirable when solving complex problems in reasonable time, and numerical methods amenable to parallelization are of particular interest.
- **Mathematical theory.** This can aid in the theoretical understanding of the properties of the method.

*Corresponding author
Email addresses: atagomes@lncc.br (Antônio T. A. Gomes), dparedes@lncc.br (Diego Paredes), valentin@lncc.br (Frédéric Valentin)

Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.072

Among the various families of numerical methods available, the most important methods fulfilling either completely or partially the cited requirements for the solution of differential equations are Finite Element (FEMs), Finite Difference (FDMs), and Finite Volumes (FVMs) Methods. Originally conceived as tools to approximate solutions to problems in solid mechanics, they are now widely used in diverse scientific fields, including fluid flow and transport problems, simulation of heat transfer, electromagnetic, environmental and biological phenomena. Consequently, articles concerning the aforementioned numerical methods are ubiquitous in the scientific literature.

From a mathematical perspective, such numerical methods create a discrete formulation to search for an approximation to the true solution. This discretization is equivalent to a system of linear equations $\mathbf{A}\,\mathbf{u} = \mathbf{b}$, which is readily solvable using, *e.g.* LU decomposition. Unfortunately, the system matrix $\mathbf{A}$ resulting from more classical methods may at times be nearly singular, resulting in stability issues. Alternatively, it is interesting to seek more efficient approaches to finding approximate solutions while maintaining accuracy in the method. This is why the definition of new methods represents an important line of active research, resulting in a very large volume of papers. In this context of a large volume of papers devoted to—or related to—numerical methods, the overall process of conception, validation, review and publication of such papers is increasingly challenging. Before developing a new numerical method, for instance, an author needs to be familiar with methods that already exist in order to understand the problems that must be addressed. The process of working through the existing bibliography can be very time-consuming, particularly when the numerical results presented for a given method are not sufficient to fully understand its behavior in interesting cases. In this situation, an author must implement the method to perform the desired tests, often with significant time-cost. A similar issue arises when a reviewer is assessing a manuscript that describes a proposal for a new method. Figure 1 shows one example [1], among many others [2, 3, 4], of a published, mathematically elegant FEM requiring a non-trivial implementation.

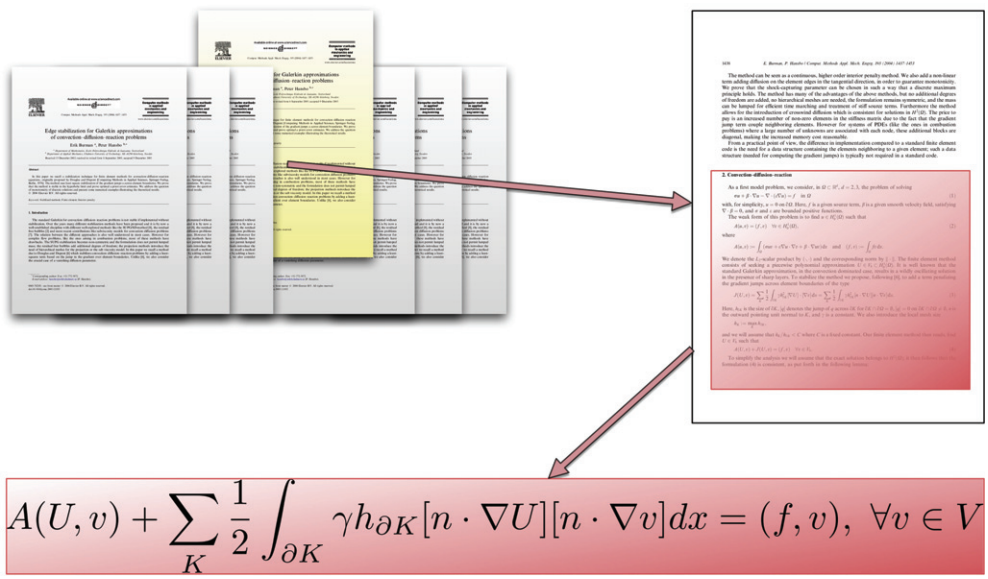

Figure 1: A mathematically elegant FEM requiring a non-trivial implementation.

The above discussion raises a fundamental question:

> *How can the publication process (from the point of view of both authors, reviewers, and editors) be streamlined so that the numerical properties of a method may be understood without the time-sink of (re)implementation ?*

In an attempt to cover the expectations generated by the above question, we envision a very simple approach to adapting the typical process for publication of papers in computational sciences or scientific computing journals. The rationale behind our approach is that the specification *and* implementation of numerical methods for solving

differential equations should be perpetuated and the experimentation with such methods should be made reproducible and easily compared with related work.

Our approach is based on a high-performance computing environment called SPiNMe (Scientific Productivity in Numerical Methods). In the SPiNMe environment, numerical methods can be implemented and run over a set of well-defined input data, including meshes and initial/boundary conditions—optionally, additional/diverse input data may be provided—, and have its output data, including accuracy and performance information, compared with other implementations. Researchers can test their implementation within the SPiNMe environment, compare their results with others previously published in the environment, and when considering such results worth publication, submit the method implementation, the input data used and output data obtained, and the corresponding manuscript in the environment for peer reviewing. The reviewers have access to all such information from a submission—hereafter *executable submission*—and can provide their own input data, running the submitted method implementation against such data and analyzing the corresponding output data if they find it necessary. The executable submission being accepted for publication, it can be made available to journal readers as an *executable paper*.

Up to our knowledge, the present work proposes an original way to approach the issue of streamlining the publication process in computational sciences or scientific computing journals. More specifically, the main contributions of the work may be outlined as follows:

- **Flexibility without lack of comparability.** The SPiNMe environment allows the various features underlying numerical method definitions—domain discretizations, variational formulations, basis functions, numerical integration processes, time-marching schemes, linear solvers, and so on—to be explored by method implementations. At the same time, the comparison of different methods does not imply the creation of whole new software packages for each new executable submission.
- **Productivity.** The SPiNMe environment provides a way for scientists to rapidly prototype their methods. Rapid prototyping techniques enables a short time interval between the conception of a proposal and the actual deployment of a functional product to serve as a proof-of-concept. This approach is of current practice in many different industries and finds place in the development of computing systems [5] as well. In particular, rapid prototyping has been recently explored for parallel [6] and scientific programming [7].
- **Long term compatibility.** The SPiNMe environment only requires from researchers the use of a local web browser (beside the typical tools used for typesetting manuscripts). Therefore, the publication process is much less prone to problems related to changes in the users' operating system and hardware architecture.
- **Security.** The SPiNMe environment offers *sandboxes* for the controlled execution of method implementations. Therefore, security threats such as defective or malicious code can only affect its own execution.

To attain these contributions, we adopted a SaaS (Software-as-a-Service) service model [8] for the proposed SPiNMe environment. We abide by this model through three key design decisions for the SPiNMe environment. First, a single set of numerical library implementations is provided to the researcher for the implementation of his/her numerical method. As such, the differences between two method implementations are arguably due to differences between the methods themselves only. Second, the provided numerical libraries can be parameterized either via input data or via "plug-in" code representing the particularities of a specific numerical method—whether it be implementation of new basis functions, variational formulations, or integration processes, etc. Finally, plug-in code is implemented in a high-level language: therefore, researchers can focus more on prototyping their methods than on coding typical idioms of lower-level languages such as C++ and Fortran (*e.g.* for memory management). Moreover, such plug-in code is amenable to sandboxing by means of application-level virtualization, in which such code is only offered a prescribed set of mathematically-related capabilities.

The remainder of the paper is organized as follows. Section 2 reviews related work. The architecture of the SPiNMe environment is presented in Section 3 and a prototype in Section 4. Section 5 is devoted to conclusions and future work.

2. Related work

To the best of our knowledge there is no project with the same intended contributions as the SPiNMe environment, although a number of projects have inspired its design.

ModelDB [9] is a repository of computational neuroscience models. This repository provides a comprehensive neuroscience dataset submitted by authors, including papers, model types and concepts, and simulation software. Nevertheless, ModelDB does not provide an execution platform and lacks a well-defined comparability basis for related work.

PyCC [7] is a Python-driven numerical method library for solving differential equations with FEM. Flexibility and efficiency are the two main design goals of PyCC, and are attained in a very similar way as the pieces of plug-in code in the SPiNMe environment. Nonetheless, PyCC allows the use of symbolic expressions for typical geometric elements, polynomial basis functions, and variational formulations, making it fairly easy for using FEM but not necessarily defining new methods based on FEM. Besides that, in contrast with our approach, PyCC provides a software package, not a service environment.

SHARE [10] is the project that bears most resemblance with SPiNMe. Crucially, both projects employ virtualization for running academic software, albeit at different virtualization levels. SHARE is a cloud computing environment that enables academic researchers to demonstrate and evaluate research software. Management of research software in the SHARE environment is attained by cloning and customizing "standard" system-level virtual machine images to host such type of software; this characteristic contrasts with the application-level virtualization employed in the SPiNMe environment. A customized image in the SHARE environment can be associated with related documentation (*e.g.* papers) and published to a group of users in the environment, which can then run the software hosted in an instantiation of such an image. Overall, the SHARE environment arguably has a broader applicability than SPiNMe, but with the cost of a poorer basis for comparability with related work.

3. SPiNMe architecture

In the following, we present the design of the SPiNMe environment. We use two different architectural views for such presentation: *use cases view* and *deployment view*.

3.1. Use Cases View

Researchers may interact with the SPiNMe environment through a web interface at three different privilege levels (see Figure 2). The first level is the **Reader** (a common user of the environment), who can *run* the various implemented methods. One example of a Reader would be a journal reviewer when evaluating an executable submission. The second level is the **Author-Developer**, who will often be a researcher interested in publishing the implementation of a method. The Author-Developer may *test* his/her implementation in the SPiNMe environment and later *include* it in his/her personal account in the environment. When the Author-Developer believes his/her method is ready for evaluation, he/she may *submit* the method to the **Moderator** (the third privilege level in the SPiNMe environment), who must *approve* or reject it. One example of a Moderator would be the editor-in-chief of a journal. Such approval/rejection can be done either by the Moderator alone or with the help of other Readers acting as reviewers. Upon acceptance, the method is made publicly available in the environment as an executable paper.

3.2. Deployment view

The SPiNMe environment comprises of three main component sets, as shown in Figure 3. The **Web Server** hosts the software components with which the users interact via web browsers. A **Cluster** (or set thereof) handles computational aspects of numerical method implementations, as well as auxiliary tools such as visualizers. A **Database Server** hosts a database management system, where the executable submissions are stored. The fact that the numerical method implementations run on clusters and all the test and submission process are handled through web-based interfaces and stored in a database makes the executable submissions very likely be perpetuated.

The portion of the SPiNMe environment concerning the Web Server component set is straightforward to implement.

The portion of the SPiNMe environment concerning the Cluster component set makes use of well-known libraries such as BLAS [11] for numerical linear algebra and VTK [12] for visualization. As for the numerical method implementations, we need a library sufficiently flexible to allow for the experimentation on the various features of method definitions—basis functions, variational formulations, and so on. A preliminary analysis of some publicly available

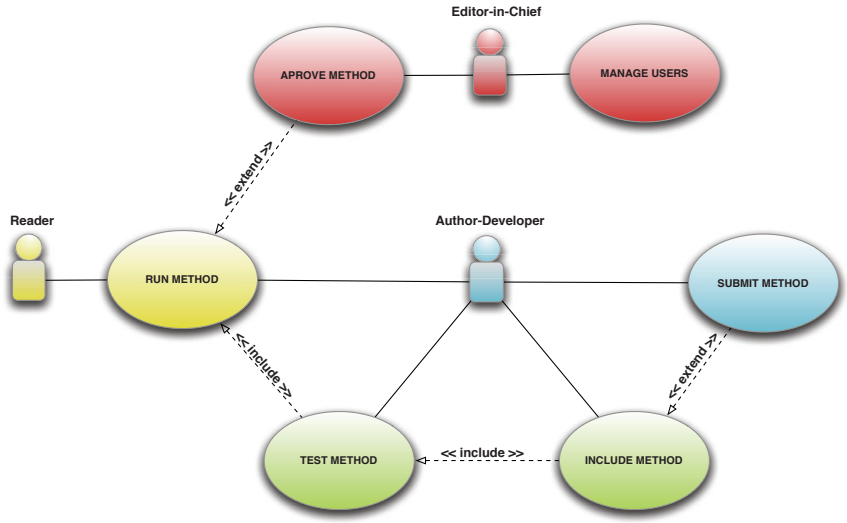

Figure 2: Use cases diagram.

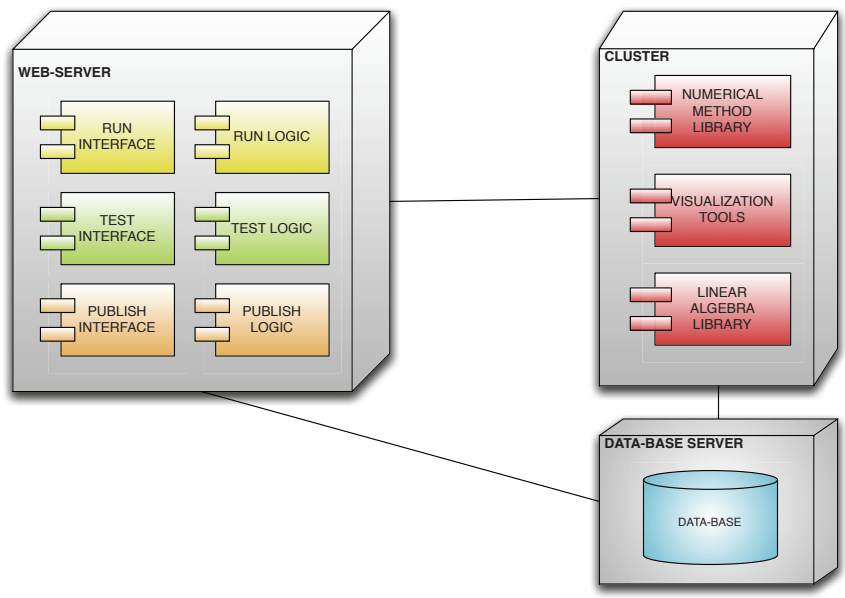

Figure 3: Deployment diagram.

numerical method libraries points out that there is no existing library to date with the flexibility we demand. Therefore, we are working on the adaptation of an existing library—the NeoPZ library[1]—to the needs of the SPiNMe environment. Our first efforts on the adaptation of the NeoPZ library have focused on FEM. Such adaptation relies upon embedding a high-level language interpreter into the NeoPZ library, which researchers may use to more rapidly prototype FEM definitions through the parametrization of NeoPZ by means of plug-in code.

[1] http://code.google.com/p/neopz/

4. Method prototyping

Method prototyping in the SPiNMe environment is based on the Lua scripting language [13]. We have chosen Lua to prototype method implementations due to its simple and extensible syntax, high abstraction level, and good performance in general. The Lua interpreter gives to the pieces of plug-in code access to *method building blocks* through a high-level API. To date, we have implemented method building blocks that allow an Author-Developer to express the variational formulation of several types of differential equations in the SPiNMe environment. The other features of numerical methods will be the subject of forthcoming work (see Section 5).

4.1. The FEM case

To better illustrate how the method building blocks for expressing variational formulations can be used in the SPiNMe environment, we provide herein a quick overview of FEM.

We start by defining the typical problem FEMs try to solve: find a function $u(\mathbf{x})$ that satisfies the following partial differential equation

$$\begin{aligned} Au(\mathbf{x}) &= f(\mathbf{x}),\ \forall \mathbf{x} \in \Omega\,, \\ u(\mathbf{x}) &= g(\mathbf{x}),\ \forall \mathbf{x} \in \partial\Omega\,, \end{aligned} \tag{1}$$

where A stands for a differential operator, $f(\mathbf{x})$ and $g(\mathbf{x})$ are given data, and Ω is a domain with boundary $\partial\Omega$. The exact solution to Equation (1) is not available in general, so we must approximate. This is the role played by FEM.

Roughly speaking, methods of this type need a discretization of Ω called *mesh*. Such a mesh is built on geometric elements such as triangles, quadrilaterals, tetrahedra, or hexahedra, for example. Figure 4 depicts an example of a typical mesh.

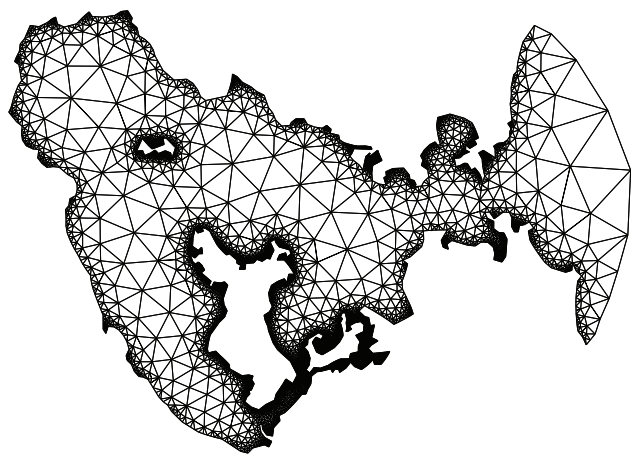

Figure 4: Example of mesh discretization of a complex domain (Guanabara Bay, Rio de Janeiro, Brazil) [14].

The first step toward the approximate solution consists of writing Equation (1) in the so-called *variational formulation* (or *weak form*), which is defined over an infinite dimensional function space V:

$$\int_\Omega Au(\mathbf{x})\, v(\mathbf{x})\, d\Omega = \int_\Omega f(\mathbf{x})\, v(\mathbf{x})\, d\Omega \quad \forall v(\mathbf{x}) \in V.$$

Next, the variational formulation is restricted to an N-dimensional function space V_h. The goal is to produce a discrete function $u_h(\mathbf{x}) \in V_h$, which is "close enough" to exact solution $u(\mathbf{x})$ (see [15] for details), and solves

$$\int_\Omega Au_h(\mathbf{x})\, v_h(\mathbf{x})\, d\Omega = \int_\Omega f(\mathbf{x})\, v_h(\mathbf{x})\, d\Omega \quad \forall v_h(\mathbf{x}) \in V_h. \tag{2}$$

Letting $\{\phi_1, \ldots, \phi_N\}$ be a basis of V_h, the solution to Equation (2) may be expressed as a linear combination of ϕ_j

$$u_h(\mathbf{x}) = \sum_{j=1}^{N} u_j \, \phi_j(\mathbf{x}) \,, \tag{3}$$

where $u_i \in \mathbb{R}$, $i = 1, \ldots, N$. Substituting (3) in (2) and then selecting $v_h = \phi_i$, for $i = 1, \ldots, N$, it holds that $\mathbf{u} = \{u_i\}_{i=1,\ldots,N}$ can be computed from the algebraic linear system $\mathbf{A}\,\mathbf{u} = \mathbf{b}$. Here matrix $\mathbf{A}$ and vector $\mathbf{b}$ are associated to the left and right hand-side terms in (2), respectively.

As an example of the procedure previously described, let us select $A \equiv -\Delta$ and $g(\mathbf{x}) \equiv 0$ in Equation (1). This corresponds to the heat conduction problem: *Find* $u(\mathbf{x})$ *such that*

$$\begin{aligned} -\Delta u(\mathbf{x}) &= f(\mathbf{x}) \text{ in } \Omega \\ u(\mathbf{x}) &= 0 \text{ on } \partial\Omega \,, \end{aligned} \tag{4}$$

where $\Delta u(\mathbf{x}) = \frac{\partial^2 u}{\partial x_1^2}(\mathbf{x}) + \frac{\partial^2 u}{\partial x_2^2}(\mathbf{x})$. For this particular case, the discrete variational formulation (2) reads

$$\sum_{j=1}^{N} u_j \int_\Omega \nabla\phi_j(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x}) \, d\Omega = \int_\Omega f(\mathbf{x}) \, \phi_i(\mathbf{x}) \, d\Omega \,, \forall i = 1, \ldots, N \,, \tag{5}$$

where we recall $\nabla\phi_i(\mathbf{x}) := (\frac{\partial\phi_i(\mathbf{x})}{\partial x_1}, \frac{\partial\phi_i(\mathbf{x})}{\partial x_2})$ (for a bidimensional domain). We notice the equation above is nothing but the linear system $\mathbf{A}\,\mathbf{u} = \mathbf{b}$ with the entries now given by

$$\mathbf{A} := \left\{A_{ij}\right\} = \left\{ \int_\Omega \nabla\phi_i(\mathbf{x}) \cdot \nabla\phi_j(\mathbf{x}) \, d\Omega \right\}, \quad \mathbf{b} := \{b_i\} = \left\{ \int_\Omega f(\mathbf{x}) \, \phi_i(\mathbf{x}) \, d\Omega \right\} , \tag{6}$$

where $i, \; j = 1, \ldots, N$.

4.2. Expressing variational formlations in SPiNMe

As can be seen in Section 4.1, a fundamental component in the implementation of any FEM is the computation of matrix $\mathbf{A}$ and vector $\mathbf{b}$ resulting from the variational formulation. This computation is usually accomplished by iterating over the elements of the mesh and assembling the contribution from each element to $\mathbf{A}$ and $\mathbf{b}$. In the SPiNMe environment, the method building blocks for expressing variational formulations offer the Author-Developer the necessary information to compute each element's contribution to $\mathbf{A}$ and $\mathbf{b}$. These building blocks are accessed through two different pieces of plug-in code, which must be implemented as Lua functions. The first Lua function is an *initialization function*. It allows the initialization of variables related to variational formulations, such as number of state variables, physical coefficients, source terms, etc.

The second Lua function is a *contribution function*. It is responsible for the actual computation of the local contributions of an element in a mesh to $\mathbf{A}$ and $\mathbf{b}$. The contribution function must receive 4 arguments: (i) a `data` structure containing information about the element over which the local contributions to $\mathbf{A}$ and $\mathbf{b}$ are to be computed—*e.g.* the values of ϕ_i and $\frac{\partial\phi_i}{\partial x_d}$, where $d = 1, \ldots, D$ and D depends on the dimension of the mesh, (ii) a `weight` number used for computing the numerical integrations in A_{ij} and b_i related with the element (which depends on the numerical integration process being employed), (iii) a `ek` matrix containing the current values of A_{ij} related with the element, and (iv) a `ef` vector containing the current values of b_i related with the element. The post-conditions of this function are the updated values of `ek` and `ef`. It is worth emphasizing that `ek` and `ef` are *updated*, since neighboring elements in the mesh will contribute to the same entries A_{ij} and b_i for some i and j.

As an example, Listing 1 shows the implementation of a Lua contribution function for the variational formulation in (5) applied to a bidimensional (*i.e.* $D = 2$) mesh. The example assumes that $f(\mathbf{x})$ in Equation (4) has been defined by a corresponding Lua initialization function.

```
function (data, weight, ek, ef)
local x = data.x -- coordinates x1 and x2 (as we are dealing with a bidimensional mesh)
local phi = data.phi -- basis functions
local dphix = data.dphix -- basis functions derivates over x1 and x2
local phr = phi.Rows()
for in = 1, phr do
 ef[in] = ef[in] + weight * f(x) * phi(in)
 for jn = 1, phr do
   ek[in][jn] = ek[in][jn] + weight * ( dphi(1,in) * dphi(1,jn) + dphi(2,in) * dphi(2,jn) )
 end
end
end
```

Listing 1: Example of a Lua contribution function in the SPiNMe environment.

5. Summary and Outlook

With the design presented herein, we aim at creating an environment for the easy implementation, comparison, and sharing of classical and new numerical methods. As the SPiNMe environment will be accessible for use through any computer with access to the Internet, a wide audience of users may be reached. Researchers will be able to make use of a simple and easy tool with which to validate new methods.

Currently, the SPiNMe environment only allows for the implementation of variational formulations as plug-in code. Our current efforts are focused on offering of an Author-Developer the option of implementing different basis functions and numerical integration processes as plug-in code. Currently, the SPiNMe environment offers to the Author-Developer piecewise polynomials as the only available basis functions and the Gaussian quadrature as the only available numerical integration process.

In addition, we have identified two main directions for future work. First, we will extend the implementation of the SPiNMe environment to families of numerical methods other than FEM. The design of the environment is similar, with adaptations being necessary to the numerical method library (as new DFMs or FVMs for instance) and to the method building blocks for the rapid prototyping of new methods as plug-in code.

Second, since data provenance has not been considered in our design, registering and tracking of actions taken on a published numerical method is not explicitly supported. This is a highly desirable feature, as the number and scope of numerical methods have been growing steadily over time. To this end, we will introduce the concept of *phenomena* in the SPiNMe environment, which will be responsible for encapsulating all information regarding a specific phenomena arising in science (*e.g*. heat transfer, turbulent flows). In addition to the description of such phenomena, this concept will contain the partial differential equations and the numerical methods used to solve them, as well as a repository filled with results of previous numerical simulations (and all information used for running them), experimental data, and bibliography related with the phenomena. Thereby, all the information and tools for understanding some phenomenon will be linked to *executable papers* helping to meet the requirement of a reproducible simulation.

Acknowledgments

This work is supported by the Brazilian National Council for Science and Technology Development (CNPq), the Chilean National Commission for Scientific and Technological Research (CONICYT), and the Brazilian Ministry of Science and Technology (MCT). The authors want to thank Christopher Harder for helpful discussions and comments.

References

[1] E. Burman, P. Hansbo, Edge stabilization for galerkin approximations of convection-diffusion-reaction problems, Computer Methods in Applied Mechanics and Engineering 193 (15-16) (2004) 1437 – 1453, recent Advances in Stabilized and Multiscale Finite Element Methods.

[2] V. John, S. Kaya, W. Layton, A two-level variational multiscale method for convection-dominated convection-diffusion equations, Computer Methods in Applied Mechanics and Engineering 195 (33-36) (2006) 4594 – 4603.

[3] L. P. Franca, A. L. Madureira, F. Valentin, Towards multiscale functions: enriching finite element spaces with local but not bubble-like functions, Computer Methods in Applied Mechanics and Engineering 194 (27-29) (2005) 3006 – 3021.

[4] J. Sudirham, J. van der Vegt, R. van Damme, Space-time discontinuous galerkin method for advection-diffusion problems on time-dependent domains, Applied Numerical Mathematics 56 (12) (2006) 1491 – 1518.

[5] S. F. Midkiff, Rapid prototyping of computing systems, IEEE Pervasive Computing 1 (1) (2002) 14–18.
[6] Y.-C. Chou, S. Nestinger, H. Cheng, Ch MPI: Interpretive parallel computing in C, Computing in Science Engineering 12 (2) (2010) 54–67.
[7] A. Logg, K. A. Mardal, M. S. Alnaes, H. P. Langtangen, O. Skavhaug, A hybrid approach to efficient finite element code development, in: D. A. Bader (Ed.), Petascale Computing: Algorithms and Applications, Chapman and Hall, 2007, Ch. 19.
[8] P. Mell, T. Grance, The NIST Definition of Cloud Computing, Tech. rep., National Institute of Standards and Technology, Information Technology Laboratory (2009).
URL `http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc`
[9] M. L. Hines, T. Morse, M. Migliore, N. T. Carnevale, G. M. Shepherd, Modeldb: A database to support computational neuroscience, Journal of Computational Neuroscience 17 (2004) 7–11.
[10] P. Van Gorp, P. Grefen, Supporting the internet-based evaluation of research software with cloud infrastructure, Software and Systems Modeling (2010) 1–18.
[11] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, An updated set of basic linear algebra subprograms (BLAS), ACM Trans. Math. Softw. 28 (2002) 135–151.
[12] W. Schroeder, K. Martin, B. Lorensen, Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition, Kitware, 2006.
[13] R. Ierusalimschy, L. Henrique, F. Waldemar, C. Filho, Lua – an extensible extension language, Software: Practice and Experience 26 (1996) 635–652.
[14] R. Araya, F. Valentin, A multiscale a posteriori error estimate, Comput. Methods Appl. Mech. Engrg. 194 (2005) 2077–2094.
[15] A. Ern, J.-L. Guermond, Theory and practice of finite elements, Springer-Verlag, 2004.