Rationales for the selection of software components for building a FEM simulation system for multi-physics problems

Roman Putanowicz[*]
Institute for Computational Civil Engineering, Cracow University of Technology
ul. Warszawska 24, 31-155 Cracow, Poland
e-mail: r.putanowicz@L5.pk.edu.pl

Abstract

This paper discusses rationales for selection of software components for building scientific simulation tools. Nowadays no single research team has the resources or knowledge to build non-trivial simulation software form scratch. Sharing experience about the motives behind the choice of software components and the consequences of particular decisions seems a valuable knowledge as it can help to avoid some potential traps. In the paper we discuss software selection decisions for our problem solving environment for numerical modelling of degradation of engineering materials. For selected tools we discuss pros and cons of their use and mention potential alternatives. The detailed discussion concerns selection of FEM library, components for handling mesh data structures, visualization and GUI libraries, provision of scripting language interfaces, software engineering tools.

Keywords: software, coupled fields, numerical analysis, finite element methods

1. Introduction

Building a new scientific simulation environment, for instance based on finite element method, is a daunting, complex task [1]. Besides knowledge and skills required to solve technical problems it also raises the issues of long time commitment, resources allocation and team management. Despite these difficulties, building own simulation tools gives us an opportunity to shape them according to the new ideas that appear in computational science such as meshless methods, XFEM, discrete exterior calculus, to name a few. It also gives us an unparalleled insight into the nature of those simulation systems.

In order to manage the complexity and the development costs of a simulation system a common approach is to use ready components, either COTS (Commercial Of-The-Shelf) or OSS (Open Source Software). While the code sharing is nothing new, we are observing steady shift towards component programming, even if not all use of software libraries can be branded as such. This trend is fuelled by the appearance of many comprehensive, high quality software packages and improvement and spread of standard interfaces between components, both on middleware and application level.

While removing the burden of low level implementation, the component approach faces the developers with the question which components the system should be composed of? Because of the early stages of the components programming, the support of component's interoperability and exchageability is still unsatisfactory. Thus selecting a component is often "one-shot decision" – after the component is tied to the rest of the system it is too costly to exchange it with another. The other issue with components is that while being functionally equivalent they can differ significantly in the costs of getting started and of maintenance.

2. Software selection drivers

The problem with selecting software components or actually any software technology is such that when it turns out that a particular choice causes problems it is usually to late to modify it. Is there any systematic way to arrive at good decisions? To the author's experience there is none – of course one can make a list of features that the components must support and investigate potential candidates, or even make a detailed study on the impact of particular choice. Such study is however expensive and time consuming. In the end what really counts is the experience. This is probably especially true for academic software project, where the software choice is done by scientists who would rather focus on their own field than investigate software itself. In most cases, it probably looks this way that the person responsible for the software choice selects some most promising candidates and the first one which installs cleanly and just "feels right" and seems easy to learn is selected. Thus the process is often driven by the first impression which can be misleading. In the light of the above it is mostly valuable to gather experience from other researches about why they selected particular component or software technology. It would be even more valuable to know how they see their decisions from the time perspective. Unfortunately such kind of wisdom is hard to find, especially comments about wrong decisions and their consequences. This is the main motivation behind this paper, to share some of our experience related to the selection of software components. This is the experience of the early stage of the project, so the outcome of some decisions is not yet fully defined.

3. FEMDK project

The material for this paper is based on our work on FEMDK project. The scope of this project is solving multi-field problems that appear during the analysis of degradation phenomena of engineering materials with special attention paid to concrete. One of the main goals is building a problem solving environment which would facilitate fast creation of tools for solving coupled problems. The main stress is on the ease of defining multi-field problems, a possibility of experimenting with new numerical algorithms, a potential to accommodate various requirements regarding data formats, geometric models, element types, FEM interpolation, solvers, etc.

[*]The work sponsored by European Union and Polish Ministry of Science and Higher Education Grant No. POIG.01.01.02-10-106/09-00 (2010-2013).

4. Software selection decisions

In this section we present rationales for our choice of software components for FEMDK project. We comment on the tools we have chosen but also give some remarks on potential alternatives.

4.1. FEM library

The heart of FEMDK project is so-called FEM Kernel – a library which supports implementation of Finite Element Method. The main criteria for selecting FEM kernel library were: the support for automatic compilation of variational forms, the ability to handle simplicial and non-simplicial meshes, support for multiple fields, support for XFEM methods. While there are at least a couple of libraries with these features we have chosen `GetFEM++` [2]. Other candidates considered were `Deal.II, libMesh, Dolfin`.

4.2. Visualisation library

The decision concerning selection of components for visualisation services is twofold. Firstly, one has to decide whether some very high level libraries should be used or one stays at relatively low level of OpenGL library and some simple wrappers for it. A high level visualisation toolkits like `VTK` provide almost all imaginable support but are somehow "fat" – they generate overhead in terms of memory consumption and processing speed. To gain maximum efficiency they must be properly handled. For some types of applications it may make more sense to stick to relatively low level OpenGL library and some drivers for it, for instance `QwtPlot3D` or `ifv++`. For FEMDK however, we have decided to use `VTK` based solutions.

`VTK` library can be an excellent tool for data visualisation task but it might not be as handy for general 3D graphics, for instance for building a graphical preprocessor. For FEMDK we have decided to use `HOOPS 3D` library by TechSoft 3D [3] which is the state-of-the-art 3D graphics system, proprietary, but available free of charge for research purposes.

4.3. Mesh handling library

For an environment such as FEMDK the crucial issue is to ensure support to exchange data between various mesh data structured in memory as well as between various data file formats. To handle such tasks the `MOAB` (Mesh Oriented dAta Base) library was selected [4]. `MOAB` provides comprehensive, scalable, efficient solution for handling structured and unstructured meshes and for associating any data with mesh elements. What is more, it is an implementation of the iMesh interface developed by The Interoperable Technologies for Advanced Petascale Simulations (ITAPS) center, and can be used with other interfaces for geometry and filed information. At one point as a viable alternative to `MOAB` the `GrAL` library was considered, however `MOAB` was chosen because of more vivid developers and users community.

4.4. GUI library

Although FEMDK is not exclusively graphical user interface oriented environment, GUI library plays important role in it. The choice of a GUI library is the hard one, as such library must be compatible with several other graphical components. It also determines to large extent the potential for portability of the whole environment. For FEMDK three candidates were seriously considered `Qt`, `wx` and `FLTK`. We have decided to use `Qt` because besides GUI it supports development of non-GUI applications and provides many general purpose programming tools.

4.5. Scripting extension language

The tasks envisioned for FEMDK environment mandate the provision of a scripting interface. The choice of a scripting language is an important decision as the scripting language will contribute at large to how the users perceive the whole environment. In case of FEMDK three languages were considered: Tcl, Guile and Python. From the technical point of view there is not much difference in how interpreters for these languages are embedded in an application nor how the extension for them can be written. For some time we have considered Guile as the main candidate. Guile is interpreter of Scheme which in turn is a functional language, and as such it has some attractive features different from the two other candidates. In the end, however, we have decided to use Python, because of already existing Python interfaces to components we would like to use in FEMDK. Python was also indicated as the preferred language by some FEMDK users.

4.6. Configuration tools

As FEMDK is more targeted at developers than at end-users the choice of software configuration and building tools is a decision that will affect potential users. Because of the use of Qt library its `qmake` tool was our first choice, but it has turned out that it is not flexible enough to our needs. Autotools (i.e. Automake, Autoconf, Libtool, etc) seem rather to be rooted in UNIX-like systems, and they demand somehow more experience from the users. In the end, we have decided to use `CMake` which is portable and able to support our various needs.

4.7. Other components used by FEMDK project

Because of constrained resources we could not select other components with the same amount of attention. The ones mentioned below were just selected simply because we already have some experience in using them.

- Handling of scientific data – for persistent data storage we use `HDF5`. It is a data model, library and file format flexible enough to accommodate all our requirements. `HDF5` is also supported by some our other components.
- Automation of multi language programming – for this task we use `SWIG` (Simplified Wrapper and Interface Generator).
- Solver libraries – there are several candidates for this kind of components and we have decided to investigate the `Trilinos` framework [5].
- Mesh generation tools – here we are restricted by the available tools. We plan to closely integrate in our environment the following generators: `gmsh, geompack++, triangle`.
- Symbolic computing – the use of symbolic computing is in our "wish list", nevertheless we have started to investigate pros and cons of using `Maxima` computer algebra system.

References

[1] Rod Oldehoeft. Taming complexity in high performance computing. In *Computational science, mathematics and software*, Ronald F. Boisvert and Elias N. Houstis (Eds.). Purdue University Press, West Lafayette, IN, USA, 57-77, 2002.

[2] GetFEM++ Homepage, `http://download.gna.org/getfem/html/homepage/index.html`, 2010.

[3] HOOPS 3D Framework, `http://developer.techsoft3d.com/hoops/index.html`, 2010.

[4] MOAB: A Mesh-Oriented datABase, `http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB`, 2010.

[5] The Trilinos Project, `http://trilinos.sandia.gov`, 2010.