Software Design Patterns in Research Software with Examples from OpenFOAM
Date: March 9, 2022
Presented by: Tomislav Maric (TU Darmstadt)
(The slides are available under “Materials from the Webinar” in the above link.)

Q. Do you have a location/url with the HPC software that has been approved/compatible to work with Exascale now? Do you have a direct URL? I don’t really have time to search through a deep site.

A. In the US, I suggest you take a look at https://www.exascaleproject.org (https://www.exascaleproject.org/the-2021-application-development-milestone-report-is-released and https://www.exascaleproject.org/wp-content/uploads/2020/02/ECP-ST-CAR-V20-1.pdf). In Japan, https://www.r-ccs.riken.jp/en/fugaku.

Q. Also, we have several Exascale systems coming soon (Intel & Nvidia). Will they all be compatible with each other? I was on top of this last year but had to work on other projects so I’m having to relearn that status of Exascale.

A. You might want to take a look at last month’s webinar, https://www.exascaleproject.org/event/wrongway (and https://www.hpcwire.com/2022/03/02/15-slides-on-right-and-wrong-ways-to-program-gpu-based-us-leadership-supercomputers). BTW, next month’s webinar will be on Evaluating Performance Portability of HPC Applications and Benchmarks Across Diverse HPC Architectures.

Q. In slide 9, isn’t the “foo.bar()” style (without passing any objects into the functions) encouraging using global variables?

A. The Computational Fluid Dynamics solution algorithm is a procedural algorithm that manipulates global variables (e.g., pressure and velocity fields) so using global variables makes sense in this context. There are many instances, in OpenFOAM, where arguments are passed on to functions to perform the update.

Q. I have read that there is a move to more composition than inheritance. What is your preference?

A. I don’t have a preference, it all depends on the problem being modeled. Keeping things simple is the primary goal, whichever mechanism allows this, should be used.

Q. Are the design patterns shown here today applicable in both .org and .com versions of OpenFOAM?

A. Yes, the patterns are used for GeometricField, lduMatrix, FunctionObjects, Mesh, Time, all relatively “old” classes.

Q. How large is the runtime "hit" caused by the polymorphism.

A. There is no “hit”: the dynamic polymorphism is being used to select Computational Fluid Dynamics algorithms and those algorithms iterate over hundreds of thousands of cells/faces/points in the mesh in long loops and perform complex calculations. The computational time is incomparably longer than the time required for the dynamic polymorphism. Still, it is a very important question: if the dynamic polymorphism is used on the level of each cell/face/point and different algorithms really are selected at runtime for different cells/faces/points, then the dynamic polymorphism “hit” becomes very visible.

Q. What is the best design pattern book for learning, as an example, the design patterns?

A. Difficult to say, I have learned this some ten years ago, i the meantime there is a lot of video material and online resources, I personally like https://refactoring.guru/design-patterns and I’m linking the slides to this site - I am not affiliated with it, I just like the concise descriptions there.

Q. From your experience, what would be good practices for designing class interfaces that survive for several years, without needing to update them often to cover more cases? OpenFOAM, for example, refactors class interfaces sometimes. Which interfaces have survived the longest? When are methods too specific or too generic, requiring updates over the years?

A. I wanted to achieve this myself as well, and it only led me to design-freeze, and too much generalization in the code, artificial increase of the abstraction level. Codes grow, they change with time, this is very difficult to avoid. I am a fan of Test Driven Development, but not in the classical sense of small unit tests; in the sense of research tests. I write my research test applications first, and in those applications, I write calls to the library I would like to have as a “user” of my own API. Then I implement the numerics, make sure it works, works fast, then I refactor (red, green, refactor). This is something that happens in cycles, from one research milestone, to the next one, and some parts of the code change (evolve I hope), some are clean from the start. There is a nice quote from Scott Meyers, “Make your interfaces easy to use correctly and difficult to use incorrectly” or something along that line, that I keep in mind.