International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

On Patterns of Multi-domain Interaction for Scientific Software Development focused on Separation of Concerns

Ileana Ober and Iulian Ober

IRIT - University of Toulouse, France
{Ileana.Ober, Iulian.Ober}@irit.fr

Abstract
This year's ICCS conference theme promotes the use of computational science as a means to foster multidisciplinarity and synergies with other fields. Our thesis is that this trend towards multidisciplinarity should be accompanied by the use of best practices issued from the software engineering community in order to avoid obtaining overly complex and tangled code, difficult to validate, to maintain and to port. In this paper we argue for the need of applying separation of concerns principles when the development involves scientists from various application fields. We overview several strategies that may be used to achieve this separation, focusing mainly on two approaches drawn from our previous experiences with multidisciplinary projects, addressing two distinct patterns of multi-domain interaction that may occur in scientific software development.

Keywords: separation of concerns, multidisciplinary development, modeling, domain specific languages

1 Introduction

A thought provoking article published by *Nature* [13] makes a critical analysis of the way software is developed and used in the context of scientific research and gives some examples of situations where this affected the quality of the scientific research results by leading to wrong conclusions. As computers and programming environments have grown more complex, the development of software by scientists with practically no training in computer programming, according to ad-hoc or self-taught methods, incurs at best a waste of time and energy. The previously cited analysis does not hesitate to use unflattering adjectives such as "awful programming", "mangled coding", that leads to "monster code".

As the research teams and topics are covering larger research areas, the interactions between various scientific disciplines increase and are often materialized by the computational science, as highlighted by this year's ICCS theme. While this is a perfectly legitimate aim and it corresponds to a real trend encountered in practice, we are concerned about the potential to worsen the quality of software currently used in scientific research.

The aim of the current paper is firstly to warn against the fact that increased multidisciplinarity poses a new risk on the quality of the produced software. Moreover, we see the current

trend towards multidisciplinarity as a huge opportunity to correct some of the deviations seen in the scientific code crafting and introduce rigorous software engineering best practices. This shift is both needed and feasible. Scientific research is not the only field where several domain experts need to work together and software plays the role of intermediary, as we will show through examples in the paper.

The issue on which we focus in this paper is separation of concerns, which is of paramount importance in a multidisciplinary computational science project. After stressing out its importance, we overview several strategies that may be used to achieve this separation. We focus mainly on two approaches drawn from our previous experiences with multidisciplinary projects, addressing two distinct patterns of multi-domain interaction.

The rest of this paper is structured as follows: in Section 2 gives a brief historical perspective on the notion of separation of concerns in software development and its importance. In Section 3 we describe two strategies for introducing and managing separation of concerns, issued from our previous experience, before concluding the paper.

2 Separation of concerns in software engineering

It is widely recognized that the growing complexity of scientific software is a major engineering challenge (see for example [6, 9]), which can only be magnified in a multidisciplinary context. The path towards a solution for this problem necessarily involves engineering techniques that have proved efficient in taming the complexity in other application fields. Software complexity has been an subject of study in search for solutions for a long time, and separation of concerns was advocated very early on, as the following excerpt from E.W. Dijkstra's 1976 *A Discipline of Programming* [4] reminds us: "*The purpose [...] is to reduce the detailed reasoning needed to a doable amount, and a separation of concerns is the way we hope to achieve this reduction. The crucial choice is, of course, what aspects to study in isolation*".

Yet the finding of the right separation lines that allow to cut through the complexity of a large software system remains a difficult task. The years since Dijkstra's remark have brought a profusion of possible solutions in the form of concepts, language constructs and methodologies for structuring software and its development. Modularity percepts [18] and standard paradigms such as object-oriented design [14] go a long way towards this goal. However, the design of complex systems having multiple stakeholders and integrating multiple heterogeneous components often requires an intent focus on separation of concerns and more specific weapons geared towards it. This has given rise to approaches such as subject-oriented design and multidimensional separation of concerns [20], aspect-oriented programming and design [10, 7], role-based design [11], to name only a few.

The existence of these approaches proves that simply relying on modularity and component based software design is not always enough to tackle the complexity of nowadays systems. If long-term sustainability of a software development project is aimed, the separation and tracking of concerns is of the essence, as it has been advocated in [19].

Another perspective on the problem of multidisciplinary development and on separation of concerns is provided by systems engineering, and in particular the recently developing field of Model-Based Systems Engineering (MBSE) [12]. This approach has emerged in the context of the development of complex cyber-physical systems (CPS) in automotive, aerospace or industrial supervisory control applications. As noted in [3], the development of these systems is by nature multidisciplinary and the effective means to manage its complexity nowadays is through the use of models. However, since CPS integrate computation, networking and physical dynamics, modeling techniques have to deal with this heterogeneity by allowing the different aspects

to be modeled separately, while providing a semantic basis for these models to inter-operate. We arrive thus at multi-modeling approaches [2] and at modeling languages such as SysML [17] that support multiple integrated views of the different aspects of a system. Model-driven engineering, a paradigm that has matured over the last twenty years [1], provides the technical support allowing to create, analyze, transform and integrate these multi-models.

3 Strategies for introducing separation of concerns.

In this section we discuss two strategies for managing and fostering separation of concerns, inspired from our previous work on large multidisciplinary projects. Both approaches, which may be qualified as vertical and horizontal, leverage the idea of domain-specific models addressing the concerns of experts in a particular discipline. The vertical approach is organized into abstraction layers, each layer corresponding to a particular expertise and adding detail to the layers above. The horizontal approach, useful when several expert teams work on different models at the same level of abstraction in related yet domain-specific languages, aims at the mechanized integration of the resulting models.

A layered approach to model-based HPC application development In previous work [16], we were involved in a four-year project that aimed to address the complexity resulting from the frequent evolutions of supercomputers and the tight coupling between software and hardware aspects, in the context of the development of scientific applications relying on high-performance computing. The technique used in this approach relied on the use of models.

Our approach, called MDE4HPC, follows a classical model based strategy: model all artifacts - including those that are non-software dependent - produced by the range of experts who typically take part in the development (physicists, applied mathematicians, computer scientists, hardware and software architects) and combine them to generate the application code. To that end, MDE4HPC offers a multi-layered architecture where each abstraction level corresponds to specific profiles of experts. Each expert will handle his dedicated model in a hardware-independent way, at the right level of abstraction, in a user-friendly modeling environment. The different concerns are separated thanks to the use of several models, as well as several modeling viewpoints on these models.

MDE4HPC uses: models (one type per domain) and $HPCML$ (High Performal Computing Modeling Language) which is defined on three layers: one dedicated to physics, one dedicated to the numerical model and another one dedicated to execution related matters (software and hardware). The layered definition of the $HPCML$ enforces the separation of concerns, by allowing each domain expert to focus on artifacts relevant to its work. Obviously, the resulting layers are not disjoint. The parts that are common to various domains help with ensuring the consistency of the transformations required when going back-and-forth between two domains. At the surface syntax level, the domain-specific language constraints proposed to various domain experts may even vary (in practice this was seldom the case, yet it occurred), however as the underlying metamodel (for the concerned parts) is the same, the resulted transformation is accurate.

Our approach for the development of scientific applications permits a selective view of the system and the separation of concerns, through the layered architecture of MDE4HPC. It is obviously not the only technique permitting a clean separation of concerns, but it proved quite effective.

A constructive approach for interoperability In the context of computational science applications that involve scientists from several domains, the software development may use different programming languages, either due to existence of programming languages tailored for the needs of a specific field (e.g. Tonto [8] - the Fortran based language for computational chemistry), or due to programming language preferences issued from background constraints (in terms of legacy code or team training). In order to avoid imposing unnecessary constraints on each of the different teams involved in the development, it is sometimes interesting to allow each of them to do the software development according to its experience and devise an automated approach for the interoperability between the artifacts produced by each of them.

In previous work [15] we have addressed the problem of allowing domain experts to cooperate using programming languages that are closely related, yet not identical. Here, we discuss these results as an example of constructive approach for interoperability of related languages. The context of our work was similar to what is encountered in certain computational science software development projects in a multidisciplinary environment. For a more detailed presentation of the approach presented in this section, the reader is referred to [15]. The problem addressed by this approach was issued from a case study that we developed with colleagues from the French National Space Agency (CNES) in the context of the DOMINO French National project [1]. The case study revealed the need to deal with a set of software components specified using related – yet different – domain specific languages (DSLs).

One critical issue in this case study is how to handle the differences existing between languages dedicated to the definition of operational procedures for space system testing and operations. Indeed, different space agencies or equipment builders use different languages to remotely control the satellites: Pluto [5] for the European Space Agency, Stol[2] by NASA, and Elisa by EADS/Astrium. These languages contain primitives to deal with algorithmic constructs, task scheduling, remote commands and measures in an imperative style. In spite of the need to occasionally use them jointly, several attempts to impose a unique language have failed, for both economical and political reasons. The various languages of the family differ at the syntactic and semantic level; nevertheless their compatibility is needed in order to achieve interoperability between agencies and user organizations and it is necessary for the development of a space mission infrastructure.

Although issued from a particular case study, this is a more general problem, as the need to use several closely related languages in the same development project occurs quite frequently in practice, in particular when the project brings together experts from various domains.

4 Conclusion and discussions

In this paper we argue for the need to apply proven state of the art software engineering techniques in the context of computational science. The evolution towards multidisciplinarity in scientific software development is bound to create a complexity problem that will provide the incentive to move away from artisanal software crafting towards a more professional approach to code development. Thus the problem could be turned into an opportunity and the use of best practices could significantly increase the quality and the sustainability of software development.

After overviewing the notion of separation of concerns in engineering, the challenges it raise and the issues it may solve, as well as some classical approaches to address it, we overview two strategies to achieve it, corresponding to two patterns of multidisciplinary interaction. The first one, that we qualify as vertical, occurs when scientific software involves challenging

[1] http://www.domino-rntl.org/

[2] the specifications for most of these languages are not public

computational problems that cannot be addressed by the field scientists alone and need the collaboration of numerical analysts and computer platform specialists (including specialists in hardware, compilers, particular libraries, etc.). The second approach, qualified as horizontal, may occur when multiple scientific fields contribute contribute components written in different yet related languages to a multidisciplinary software project. The problem in this case is the integration of these components and our approach aims at automating the interoperability of these components.

Although the focus of this paper is separation of concerns, other software engineering techniques, such as rigorous requirement engineering, requirement traceability and rigorous validation processes, provide great potential for improving the quality of scientific software.

References

[1] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[2] C. Brooks, Chihhong P. Cheng, T.H. Feng, E. A. Lee, and R. von Hanxleden. Model engineering using multimodeling. Technical report, UC Berkeley, 2008.

[3] Patricia Derler, Edward A. Lee, and Alberto L. Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[4] Edsger W. Dijkstra. *A discipline of programming*. Prentice-Hall Englewood Cliffs, 1976.

[5] ESA. ECSS-E-70-31A. space engineering standard - ground systems and operations - monitoring and control data definition standard. Technical report, European Space Agency.

[6] National Science Foundation Advisory Committee for CyberInfrastructure Task Force on Software for Science and Engineering. Final report, 2011.

[7] I. Jacobson and P. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-W., 2004.

[8] D. Jayatilaka and D.J. Grimwood. Tonto: A fortran based object-oriented system for quantum chemistry and crystallography. In *Proceedings of ICCS*, pages 142–151. Springer, 2003.

[9] L.N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, May 2013.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP Proceedings*, LNCS 1241. Springer, 1997.

[11] Bent Bruun Kristensen. Object-oriented modelling with roles. In *OOIS'95*. Springer, 1996.

[12] INCOSE. Systems Engineering Handbook - A Guide for System Life Cycle Processes and Activities, version 3.2. International Council on Systems Engineering, 2010.

[13] Zeeya Merali. Computational science: ...error. *Nature*, 467(7317):775–777, October 2010.

[14] Bertrand Meyer. *Object-Oriented Software Construction (2nd Edition)*. Prentice Hall, 2000.

[15] I. Ober, L. Féraud, and C. Percebois. Dealing with variability within a family of domain-specific languages: comparative analysis of different techniques. *ISSE*, 6(1):21–28, 2010.

[16] I. Ober, M. Palyart, J-M. Bruel, and D. Lugato. On the use of models for high-performance scientific computing applications: an experience report. *SoSyM J.* To appear. http://dx.doi.org/10.1007/s10270-016-0518-0.

[17] OMG. Systems Modeling Language (SysML), v1.1. http://www.omg.org/spec/SysML/1.1/, 2008.

[18] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972.

[19] Martin P. Robillard and Gail C. Murphy. Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, 16(1), February 2007.

[20] Peri Tarr, Harold Ossher, William Harrison, and Stanley M Sutton Jr. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of ICSE*, pages 107–119. ACM, 1999.