MDE in Practice for Computational Science

Jean-Michel Bruel, Benoit Combemale, Ileana Ober, Hélène Raynal

MDE in Practice for Computational Science

Jean-Michel Bruel[1], Benoit Combemale[2], Ileana Ober[1], and Hélène Raynal[3]

[1] IRIT/University of Toulouse - {bruel,ileana.ober}@irit.fr
[2] INRIA and University of Rennes - benoit.combemale@irisa.fr
[3] INRA - helene.raynal@toulouse.inra.fr

Abstract

The complex problems that computational science addresses are more and more benefiting from the progress of computing facilities (simulators, librairies, accessible languages, ...). Nevertheless, the actual solutions call for several improvements. Among those, we address in this paper the needs for leveraging on knowledge and expertise by focusing on Domain-Specific Modeling Languages application. In this vision paper we illustrate, through concrete experiments, how the last DSML research help getting closer the problem and implementation spaces.

Keywords: Model-Driven Engineering, Domain-Specific Modeling Languages, Scientific Computing

1 Introduction

Computational Science tackles complex problems by definition. These problems concern people not only in large scale, but in their day-to-day life. With the development of computing facilities, novel application areas (social life, open data, ...) can legitimately benefit from the existing experience in the field [9]. Nevertheless, the lack of reusability, the growing complexity, and the "computing-oriented" nature of the actual solutions call for several improvements (reusability, scalability, ...). Among these, raising the level of abstraction is the one we address in this paper. Some key concerns have already been identified [13]. As an illustration we can mention the problem of the validity of the experimentation which depends on the validity of the defined programs (bugs not in the experiment and data but in the simulators/validators!). This raises the need for leveraging on knowledge and expertise of domain experts.

In the software and systems modeling community, research on domain-specific modeling languages (DSMLs) is focused, especially since the last decade, on providing technologies for developing languages and tools that allow domain experts to develop system solutions efficiently.

In this vision paper[1], based on concrete experiments, we claim that DSMLs can help closing the gap beween the (problem) space in which scientists work and the implementation space (solution) in which programming is involved most of the time. Incorporating domain-specific

[1]This work is partially supported by the ANR INS Project GEMOC (ANR-12-INSE-0011).

concepts and high-quality development experience into DSMLs can significantly improve scientist productivity and experimentation quality. Finally, we give some insights on perspectives that will illustrate the importance of the approach we advocate here.

2 Modeling in Science and Engineering

Building a model of some real world phenomenon seems to be an intrinsic human endeavor to understand and predict occurrences in the world. Many disciplines use models to help navigate through complex concepts in order to gain a better understanding of their target of study. For example, chemists use models to study atoms and molecules, mathematicians use models to study the abstract nature of numbers, functions and other concepts, mechanical and electrical engineers build models of the engines to predict how the engines will behave when built, etc. Modeling has been the essential mechanism to cope with the complexity of reality. While in science, models are used to describe existing phenomena of the real world, in engineering, models are used to describe a system that is to be developed in the future. Thus engineering models are typically *constructive* while scientific models are *descriptive*. In the following, we more closely examine the way models are used in science and engineering, we discuss how they are coming close together in current trends, and we open to possible cross fertilization.

2.1 Modeling in Science

Scientists handle the complexity of the phenomena they are studying through *modeling*. Of course, to be useful as communication means, models have to be made *explicit*, that is, assumptions on the world must be made explicit, and communicated in a language that can be understood by most of stakeholders. Stachowiak provides a detailed definition of modeling [18], where the three main characteristics of a *model* are described as follows: (i) There is an original. (ii) The model is an abstraction of the original. (iii) The model fulfills a purpose with respect to the original. Scientific models are typically used to understand the real world and predict some aspects of the real world: using Newton's laws of gravitation we can predict the time it will take for an apple to fall from a tree. That led the philosopher K. Popper to the characterization of scientific theories as *falsifiable models*, i.e. models that can be compared to some observable reality to assess whether and where they fit. Thus the *original* is part of this world. Scientists abstract away from complex details and typically the models they construct only hold within certain boundaries that need to be explicitly understood too. For instance, Newton's laws of gravitation only holds nearby the surface and for objects of certain size. Some models are known to be wrong, but still explain certain phenomena quite well, e.g., Kepler's geocentric model of the solar system. Abstraction always means that certain properties are lost while (hopefully) the relevant ones, with respect to the purpose of the model, are captured in enough detail to fulfill the model's purpose. Whether a model is helpful can therefore only be answered with the knowledge about its purpose.

2.2 Modeling in Engineering

Like scientists, engineers (including software engineers) use models to address complexity. However a key difference is that the phenomenon (e.g., engine, process, software, building) they model generally does not exist *at the time the model is built*, because the engineers' goal is to build the phenomenon from the model. That is, the model acts as a blueprint. In engineering, one wants to break down a complex system into as many models as needed in order to

address all the relevant concerns in such a way that they become understandable, analyzable and finally can be constructed. This *separation of concerns* engineering principle has stood the test of time. In software engineering, Model-Driven (software) Engineering (MDE) aims at reducing the accidental complexity associated with developing complex software-intensive systems [16]. A primary source of accidental complexity is the wide gap between the high-level concepts used by domain experts to express their specific needs and the low-level abstractions provided by general-purpose programming languages [6]. Manually bridging this gap is costly in terms of both time and effort. MDE approaches address this problem through the use of modeling techniques that support separation of concerns and automated generation of major system artifacts (e.g., test cases, implementations) from models. In MDE, a model describes an aspect of a system. Separation of concerns is supported through the use of different modeling languages, each providing constructs based on abstractions that are specific to an aspect of a system.

Incorporating domain-specific concepts and high-quality development experience into MDE technologies can significantly improve domain expert productivity and system quality. This realization has led to work, starting in the late nineties, on MDE language workbenches that support DSMLs and associated tools (e.g., model editors, simulators and code generators). A DSML provides a bridge between the (problem) space in which domain experts work and the implementation (programming) space. In this context, DSMLs can be used to support socio-technical coordination by providing the means for stakeholders to bridge the gap between how they perceive a problem and its solution, and the programming technologies used to implement a solution.

Domains in which DSMLs have been developed and used include automotive, avionics, and cyber-physical systems. Recently, Hutchinson et al. provided some indications about the industrial adoption of DSMLs [5].

Research on systematic development of DSMLs has produced a technology base that is robust enough to support the integration of DSML development processes into large-scale industrial system development environments. Current DSML workbenches support the development of DSMLs to create models that play pivotal roles in different development phases. Workbenches such as Microsofts DSL tools, MetaCases MetaEdit+, JetBrains MPS, Eclipse Modeling Framework (EMF) and Generic Modeling Environment (GME) support the specification of the abstract syntax, concrete syntax and the static and dynamic semantics of a DSML. These workbenches address the needs of DSML developers in a variety of application domains.

Using MDE technologies, a new DSML is typically first specified through a domain model, so-called *metamodel*, that define the abstract syntax of the DSML as a set of domain-specific concepts and relationships between them. Then, the MDE community has developed a rich ecosystem of interoperable, generative tools defined over standardized object-oriented metamodeling technologies such as EMOF [12]. These tools are capable of generating supporting DSML tools such as editors (either graphical or textual), code generators, simulators, and other Integrated Development Environment (IDE) services.

2.3 Convergence in current trends

While the previous discussion argue to a clear distinction between models in science and models in engineering, current trends however aim to fade this distinction.

Even if the primary goal of the models in engineering is to reason about the system to be built (e.g., for analysis or design). Yet, engineers have also to carry out legacy systems for maintenance, and running systems for, possibly dynamically, adaptations. Consequently, the

software engineering community has developed in the last decades tools and methods for building models from already implemented systems (e.g., retro-engineering and automatic repair), or from running systems (models at runtime). While in the former case models are used to understand an existing system for re-engineering purposes, in the latter case models act as a reflection layer to take decisions in terms of reconfiguration for adapting the system to an evolving environment.

In science, while models are primarily used to understand existing phenomena of the real word, computational methods enable the description of phenomena that doesn't not exist yet. The objective is either to make the scientific studies more efficient (e.g., emulate the growing of a plant or an animal until it is relevant to start the study), or to study the interest of a certain phenomenon that doesn't exist yet but could be implemented (e.g., evaluate the impact of the implementation of certain agricultural policies). Moreover, scientists more and more develop software that realize the experimentation, and whose validation and verification become crucial [13]. In such a case, models of the experimentation setup act as engineering models to build software systems able to implement the experimentation of existing or expected phenomena.

The convergence observed in the current trends of modeling in science and engineering is blurring the border and foster a possible cross-fertilization of the tools and methods that exist in both fields. In the rest of this paper, we present a vision where DSML workbenches used for software engineering models would be useful to get closer the (problem) space in which scientist work and the implementation (programming) space in which scientific models are manipulated to reason about them. We illustrate this vision with two experimentation where DSMLs are defined first in scientific computing applications and second in the domain of farming systems to help both scientists and farmers to reason about farming exploitation.

3 Why current MDE trends are mature enough

Some preliminary applications of model driven engineering in computational science have been observed in recent years, e.g., data modeling in scientific simulation workflows [1] and DSML in particular scientific domains such as the *Chemical Markup Language*[2] in chemistry.

We envision that this is the good moment to push forward such use of MDE in computational science. The problem was maybe too complex and too global to be thought as an application domain for the entire MDE in the previous status of the MDE tooling maturity. Our goal in this section is not to provide a detailed argumentation but we hope the following references and illustration will make sense as a convincing invitation to pursue the investigation. Recent efforts in the MDE community have lead to the demonstration of the maturity of MDE tools to tackle industrial problems. This is for example the intent of [5], where the authors discuss more specifically the impact of tools on MDE adoption, through a deep empirical study. One of their conclusions, which is a focus we would like to highlight in this paper (while not considering only tooling concerns) is: "Match tools to people, not the other way around". Some members of the MDE community are convinced that it is time now to switch the focus of our research effort from "doing better and faster" to "doing new thing". As an illustration of such an effort, we can mention the "Modeling Outside the Box" series of workshops where one of the last brainstorming edition has lead to a set of trends for the future of MDE [11].

We can check what has changed since the existing report on MDE adoption in industry, realized in 2008 [19] and where one of the main conclusions were that the most important obstacle is the complexity of developing an MDE environment tailored to the company needs.

[2] Cf. `http://www.xml-cml.org`

A great amount of effort has been done to face this problem. The best illustration is the fact that we talk less and less about MDE and more and more about DSMLs (see previous section). Tools like Xtext, Sirius, MPS, MetaEdit+ and OpenFlexo are focusing on user-friendliness and intuitive use of models. They are not only more used in industry but also taught in universities and schools. Even tools fully UML-prone like Papyrus is putting lots of effort in the tuning and customization capabilities, getting closer to the user experience. It is possible now to both benefit from a strong UML basis (using the profiling mechanism to extend the basic concepts) with the definition of domain-specific elements either at the syntactic level (visual representation) or the semantic level (extending the recently available execution semantic fUML or the tooling level (customized palettes and views). As mentioned in reference [5]: "Companies who successfully applied MDE largely did so by creating or using languages specifically developed for their domain". Indeed the list of industrially designed DSMLs is impressive.

In the following section we illustrate our position by providing two concrete experiences of the use of MDE in the context of two different computational domains, both being practical industrial ones.

4 Description of the Experiments

4.1 HPCML: a DSML for FORTRAN

In previous work, we had the chance of being involved in a four years long experiment of raising the abstraction level in the development of scientific applications using high performance computing though applying model-driven engineering (MDE) techniques. The work was mainly triggered by maintenance concerns (related to the frequent need of platform changes, with respect to a steady numerical part), as well as by the need to naturally (easily?) integrate the separation of concerns in the context of applications where several profiles of stakeholders would have to act on the same application.

Synopsis of the approach Our approach for adding abstraction in HPC applications, approach called MDE4HPC, is overviewed in [14]. MDE4HPC tackles the development complexity of scientific application, with intensive use of numerical simulation, by means of raising abstraction. This approach is based on the definition of a multi-layered domain specific language *HPCML*. This language has a modular and layered definition that allows to capture information specific to the modeling of the numerical aspects, of the physical model and of the computation information. For the definition of this language we defined its metamodel as well as a graphical syntax addressed to domain experts thus making the manipulation of models transparent to this class of users, which is critical as in general these domain experts have little or no a formal training in computer science. In order to make our approach effectively usable we made the needed tool developments to integrate the use of the domain specific language it in existing development tool chains. In this section we give some glimpses into this approach. For more details the reader is referred to [14]. The aim of this section is to report on the feasibility of using MDE in HPC applications, and to show how using this approach opens the way to the use of some features, that are not traditionally used by HPC applications, but are used on a larger scale in the development of real-time and embedded applications - fields that report success stories of adopting MDE since a few years ago.

The tool development of this experiment was carried out by colleagues from CEA and integrated into the internal CEA toolset ArchiMDE. ArchiMDE offers a GUI adapted to each

domain expert (e.g., applied mathematicians are able to take advantage of perspectives to implement a numerical schema as independently as possible from the underlying platform) and handles models that integrate information from various domain experts, by means of model transformations. It exploits the information provided by domain experts and generates an intermediate level application handled by the already existing *Arcane* framework [7], a software development framework for 2D and 3D numerical simulation codes, including mesh management, parallelism strategy, etc.

Evaluation Without going into the details of a real case study, in this section, we give an intuition of how a piece of computation described using our DSML would look like. Aware of the importance of using a graphical representation, we complemented the definition of the HPCML domain specific language with a graphical syntax, designed by applying the principles stated by the Moodys theory [10]. Our evaluation, included using our approach to develop the simplified Lagrangian hydrodynamics code introduced in [7]. More details on this evaluation are available in [15]. As an alternative to "classical" Fortran programming, the modeling using HPCML offers a more abstract and graphical view of the numerical application. For instance, the overall execution flow is summarized by the HPCFlowDescriptor, whose description is provided in Figure 1. This component features several constructions for expressing the parallelism: a parallel for and a couple fork/join. Having various means to express parallelism allows them to be combined to best exploit the characteristics of the underlying hardware platform. One can notice that although we have not introduced the syntax used in HPCML, it is possible to understand the main steps of this component. Note that the description available in Figure 1 describes only the execution flow and it is only part of a more complete specification. It is unlikely that the equivalent Fortran specification, that served as inspiration in our work, would be as accessible.

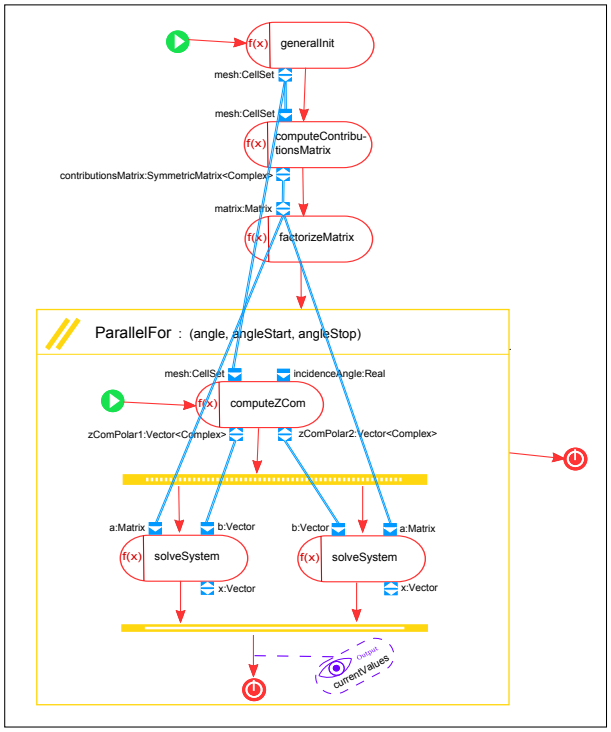

Figure 1: A MDE Component

Openings The current work opens the way to more exciting studies. One direction would be to enrich the current model with non functional information related for instance to uncertainty characterisations of some calculus, performance analysis of some algorithms, resource consumption information (in terms of time, energy or hardware charge), etc. This non functional information expressed at the model level would open the way to parameterisations following various criteria, possibly defined dynamically that would lead to more flexible computations. Depending on some resource availability parameters the execution would vary dynamically to

get either more trustworthy (thus minimising the uncertainty parameter), more rapid (minimising the execution times) of less energy consuming. Interesting openings exist in this context with work done recently in the context of Models@runtime [4].

4.2 The farming system model

Scientific simulation models have been used for decades in agricultural research. The rapid changes in the agricultural context driven by continuously arising challenges (climate change, environmental issues, food self sufficiency...) require more complex models that take into account different elements: crop, farm organization, environment, biodiversity, etc.

One of the key issues of agricultural research concerns the adaptation of the farming system to the new agricultural context. *In silico* approaches, based on simulation of a wide range of possible farming systems, offer the possibility of identifying more quickly new systems to tackle current social, political and environmental concerns [3]. However, building, testing, evaluating, using farming system models is far from being a straightforward task. Indeed, scientists must deal with four major challenges: i) the development of these models require the collaboration of scientists of several disciplines such as agronomy, animal science, soil science, bioclimatology, epidemiology, economics, science of management and computer science; ii) this multidisciplinary work conducts to integrating all the components coming from the different disciplines (with problem of model composition); iii) models are increasingly complex, either because they approach increasingly complexity of the real (multi-scales, a more accurate level of granularity) or because they incorporate more aspects than in the past; iv) to cover the whole modeling cycle from the conceptual models up to the results analyzing, modelers need tools with a wide range of functionalities (to design the conceptual model, for software development, for running simulation, for the visualization ...).

To overcome these difficulties, at INRA (French National Institute For Agricultural Research), the researcher community working on farming systems developed a modeling and simulation platform, namely RECORD [2]. This platform is based on the DEVS formalism [20] that enables robust coupling between heterogeneous components. The platform has a repository of models and components ready to use. It also provides integrated tools for initialization, running simulations and GUI (Graphical User Interfaces). This GUI help modelers without specific skills in programming, to develop and simulate their model, but these graphical interfaces do not allow total abstraction of software code. This is especially relevant when the conceptual model is designed following a participative approach with the stakeholders themselves. The aim of this experiment was to evaluate how MDE can be applied in a very operational context of computational science, and the genericity of some software tools used in MDE.

The application is a simplified version of a farming system. It has been proposed by the platform lead group of scientists and is commonly used in RECORD training sessions. It aims at modeling a farm with three production workshop: cows, ewes and crop. The farm has different fields and grasslands. The farmer decides the type of species on each field regards on agronomic and economic considerations. Some farmers activities are daily ones, while others occur only at some specific periods of the year. For these latter ones, decision rules have been defined, they will be used for launching effectively the activities required by the crop. We consider some constraints on the system, the total amount of irrigation water available for the farmer is limited, there are only two workers on the farm.

Synopsis of the approach In order to illustrate as much as possible the potential of MDE, we have chosen two different approaches for building the DSML. Among the various ways

to build a DSML, we have selected the two main approaches: (i) building the DSML *from scratch*, meaning starting from an empty metamodel and building it by construction, and (ii) building the DSML based on existing concepts, in our case as a UML profile. The effort as been conducted in parallel, by two different authors of this paper with the help of different teams. For the first approach we have used Sirius[3], following mainly the approach described in reference [8].For the second we have used Papyrus[4], following mainly the approach described in [17].

The average load in work was around 12 hours. The DSML development has started from a provided documentation detailing the Farming System described in the previous subsection. An additional Skype session has allowed to clarify some initial concerns.

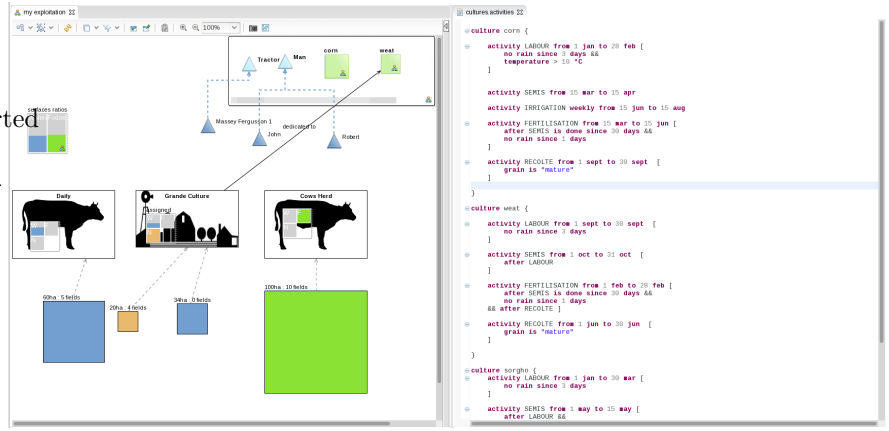

Figure 2: A Farming DSML based on an adhoc Metamodel

An illustration of the DSMLs obtained is given in Fig. 2. The complete details are available[5].

Evaluation Not surprisingly the DSMLs have much in common and define all the basic concepts (and their attributes) of the farming system. The main differences in the DSML definition reside in the process to define them and in the potential exploitation of the result. Despite the differences (that are not treated here due to length limitation)the overall benefits of a DSMLs were basically related to the two aspects of a DSML: its syntax and its semantic. The fact that the concrete syntax manipulated by the domain expert can be adapted to his/her needs was very important and hence very appreciated. For example, in the Sirius version is was possible to adjust values in a textual version of the model and see its corresponding graphical representation updating at run-time.The fact that the semantics is clearly defined allows "code" generation with a minimum of effort and with a maximum of evolution feature.

Openings The benefits mentioned above are somehow classical of the use of DSMLs. This was the minimal requirement: provide a tool-supported modeling environment that was easy to manipulate by the end-user. Some questions and perspectives remain. E.g., is this MDE approach really "user friendly" enough for people who are not modelers but who are needed in the design of the model; Is there a way to use the dedicated modeling environment without installing complicated tools (web interfaced tools); Is it possible to produce formal model in DEVS; Could an already defined ontology have been an help in the definition of the DSML; etc. As we can see there are a number of open issues. In the following section, we try to draw,

[3]See `https://www.eclipse.org/proposals/modeling.sirius/`.
[4]See `https://www.eclipse.org/papyrus/`.
[5]See `https://github.com/jmbruel/idm2014`

from the two specific case studies, some guidelines for a better integration of Model-Driven techniques in Computational Science.

5 Lessons Learned and Concluding Remarks

In the previous section, we overviewed two concrete projects, that benefit from using a model-driven approach in an attempt to adding abstraction and facilitate a project-level reasoning. These investigations show, on one hand, that raising the level of abstraction through the use of a model based approach, gives a genuine added-value that worth being considered as a technique to use more largely in scientific applications. Along with the positive feedback brought by these investigations, we had the opportunity to identify some points that deserve to be more closely studied in order to make the experience of adding abstraction through the use of modeling techniques even more beneficial, as well as easier to apply.

As a conclusion, we summarize the main directions on which we think further efforts should concentrate, in order to distill a model-driven computational science - a way of developing scientific applications that includes several abstraction layers, thus offering the possibility to reason at the right level of abstraction depending on the task one has to fulfill. During our experiments, we experienced the importance of user-friendliness for tools to support our approach. This was witnessed both by domain experts who at some points needed particular assistance and by people with solid experience in the use of modeling techniques. The need for mature, design-oriented, user-friendly tools is naturally not specific to the scientific computing and it is often cited as a hinder in a faster (wider) adoption of these techniques in the industry.Our experience is that in some application domains, such as the field of embedded software or real-time systems, MDE managed to overpass this weakness. The situation is slightly different in the area of scientific computing, where there is strong focus on code as a development artefact and a tendency to grasp each performance increase, deep into the code optimization. The adoption of modeling techniques in this field would probably be even more dependent on the existence of support closer to the domain user habits. It is obviously not clear what such a tool support actually means, but it is clear that it should at least include an easy install procedure, integrated versioning mechanisms and support for collaborative work (such as web based platforms).

One of the primary benefits brought by the use of abstraction through modeling is the possibility to better reason on the system under development, to symbolically simulate conceptual architectures. With a proper tool support, this alone could offer a noteworthy benefit. Applying model based development techniques could open the way to field specific analysis that today is not performed because it is difficult (or impossible to do it) while lacking abstraction mechanisms. In the fields of real-time and embedded systems the use of modeling techniques was quickly followed by the use of advanced verification mechanisms such as model checking and model based testing. Today none of these techniques are applied to scientific applications, nevertheless some applications could benefit from the use of model based validation and verification. Another challenge for the MDE community is to support design as an art, not just as a high-level programming technique. More efforts, both on methodology (for collaboratively building conceptual models) and on tools (more intuitive and less computer-oriented) would certainly help. The current trends in MDE are in this direction. The shift from classical engineering models to “what-if models” is a new but popular idea in MDE; the tendency to have a “Global problem” approach (zoom in/zoom out) versus a “System/Subsytem” approach (components) is also a direction more and more followed. All these trends are for us signs that we are at an interesting time for really benefiting from cross-disciplines fertilization between MDE and computational science, especially in the field of natural sciences.

References

[1] A. Bender, A. Poschlad, S. Bozic, and I. Kondov. A service-oriented framework for integration of domain-specific data models in scientific workflows. *Procedia Computer Science*, 18(0):1087 – 1096, 2013. 2013 International Conference on Computational Science.

[2] J.-E. Bergez, P. Chabrier, C. Gary, M. H. Jeuffroy, D. Makowski, G. Quesnel, E. Ramat, H. Raynal, N. Rousse, D. Wallach, P. Debaeke, P. Durand, M. Duru, J. Dury, P. Faverdin, C. Gascuel-Odoux, and F. Garcia. An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems. *Environmental modellig and Software*, 39:39–49, 2013.

[3] J.-E. Bergez, N. Colbach, O. Crespo, F. Garcia, M. Jeuffroy, E. Justes, C.Loyce, N. Munier-Jolain, and W. Sadok. Designing crop management systems by simulation. *European Journal of Agronomy*, 32:3–9, 2010.

[4] R. Breu, B. Agreiter, M. Farwick, M. Felderer, M. Hafner, and F. Innerhofer-Oberperfler. Living models - ten principles for change-driven software engineering. *Int. J. Software and Informatics*, 5(1-2):267–290, 2011.

[5] C. Bull and J. Whittle. Supporting reflective practice in software engineering education through a studio-based approach. *IEEE Software*, 31(4):44–50, 2014.

[6] R. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54. IEEE Computer Society, 2007.

[7] G. Grospellier and B. Lelandais. The Arcane development framework. In *POOSC'09*, pages 4:1–4:11, New York, NY, USA, 2009. ACM.

[8] J.-M. Jézéquel, B. Combemal, and D. Vojtisek. *Ingénierie Dirigée par les Modèles*. Ellipses, 2012.

[9] Z. Merali. Computational science:...error. *Nature*, pages 775–777, 2010.

[10] D. L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.

[11] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. Cheng, P. Collet, B. Combemale, R. France, R. Heldal, J. Hill, J. Kienzle, M. Schaettle, F. Steimann, D. Stikkolorum, and J. Whittle. The Relevance of Model-Driven Engineering Thirty Years from Now. In *ACM/IEEE Model-Driven Engineering Languages and Systems*, volume 8767 of *LNCS*, pages 183–200. Springer, 2014.

[12] Object Management Group. Meta object facility (MOF) core specification, 2014. Version 2.4.2.

[13] D. Orchard and A. Rice. A computational science agenda for programming language research. *Procedia Computer Science*, 29(0):713 – 727, 2014.

[14] M. Palyart, D. Lugato, I. Ober, and J. Bruel. Improving scalability and maintenance of software for high-performance scientific computing by combining MDE and frameworks. In *MODELS 2011*, LNCS, pages 213–227, 2011.

[15] M. Palyart, I. Ober, D. Lugato, and J. Bruel. HPCML: a modeling language dedicated to high-performance scientific computing. In *1st International Workshop on Model-Driven Engineering for High Performance and CLoud computing*, page 6, 2012.

[16] D. C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[17] B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *ISORC'07*, pages 2–9, May 2007.

[18] H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[19] R. Van Der Straeten, T. Mens, and S. Van Baelen. Challenges in Model-Driven Software Engineering. In M. Chaudron, editor, *Models in Software Engineering*, volume 5421 of *LNCS*, pages 35–47. Springer Berlin Heidelberg, 2009.

[20] B. Zeigler, H.Praehofer, and T. G. Kim. *Theory of Modelling and Simulation:Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, 2000.