Software Diversity: Challenges to handle the imposed, Opportunities to harness the chosen

Mathieu Acher, Olivier Barais, Benoit Baudry, Arnaud Blouin, Johann Bourcier, Benoit Combemale, Jean-Marc Jézéquel, Noël Plouzeau

Software Diversity: Challenges to handle the imposed, Opportunities to harness the chosen

Mathieu Acher, Olivier Barais, Benoit Baudry, Arnaud Blouin, Johann Bourcier, Benoit Combemale, Jean-Marc Jézéquel, and Noel Plouzeau

DiverSE team at INRIA / IRISA

1 Context

Diversity emerges as a critical concern that spans all activities in software engineering (from design to verification, from deployment to runtime resilience) and appears in all sorts of domains, which rely on software intensive systems, from systems of systems to pervasive combinations of Internet of Things and Internet of Services. If these domains are apparently radically different, we envision a strong convergence of the scientific principles underpinning their construction and validation towards **flexible and open yet dependable systems**.

In this paper, we discuss the software engineering challenges raised by these requirements for flexibility and openness, focusing on four dimensions of diversity: the **diversity of functionalities** required by the different customers (Section 2); the **diversity of languages** used by the stakeholders involved in the construction of these systems (Section 3); the **diversity of runtime environments** in which software has to run and adapt (Section 4); the **diversity of failures** against which the system must be able to react (Section 5). In particular, we want to emphasize the **challenges for handling imposed diversity**, as well as the **opportunities to leverage chosen diversity**. The main challenge is that software diversity imposes to integrate the fact that software must adapt to changes in the requirements and environment – in all development phases and in unpredictable ways. Yet, exploiting and increasing software diversity is a great opportunity to allow the spontaneous exploration of alternative software solutions and proactively prepare for unforeseen changes. Concretely, we want to provide software engineers with the ability:

- to characterize an 'envelope' of possible variations;
- to compose 'envelopes' (to discover new macro envelopes in an opportunistic manner);
- to dynamically synthesize software inside a given envelop.

The major scientific challenge we foresee for software engineering is elicited below

> Automatically **compose and synthesize software diversity** from design to runtime to **address unpredictable evolutions of software intensive systems**.

2 Diversity of functionalities

2.1 Imposed diversity: diversity of requirements and usages

The growing adoption of software in all sectors of our societies comes with a growing diversity of usages (from pure computation in its early days, to a variety ranging from transportation, energy, economy, communication, games and manufacturing today). This variety of usages and users puts pressure on software development companies, who aim at reusing as much code as possible from one customer to

another, yet who want to build the product that fits the user specific requirements. *Software Product Lines* (SPL) have emerged as a way to handle this challenge (reuse, yet be specific) [1]. Central to both processes is the management and modeling of variability across a product line of software systems. Variability is usually expressed in terms of *features*, originally defined by Kang et al. as: *"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"* [2].

A fundamental problem is that the number of variants can be exponential in the number of features: 300 boolean optional features lead to approximately 10^{90} configurations. Practitioners thus face the challenge of developing billions of variants. It is easy to forget a necessary constraint, leading to the synthesis of unsafe variants, or to under-approximate the capabilities of the software platform. Scalable modelling techniques are therefore crucial to specify and reason about a very large set of variants.

Challenge #1: scalable management of variability

2.2 Chosen diversity: adaptive systems in evolving environments

Software systems now need to dynamically evolve to fit changes in their requirements (e.g., change of environment, user, or platform) at runtime. The growing adoption and presence of software is a factor of chosen diversity that can answer this problem. Such a diversity is composed available software services developed and deployed by third parties that software systems can exploit at runtime to fit their current requirements. The challenge is to develop (self-)adaptive systems that can smoothly discover, select, and integrate available services at runtime.

Opportunity #1: exploiting ambient functionalities within adaptive systems

3 Diversity of languages

3.1 Imposed diversity: diversity of views and paradigms in systems engineering

Past research on modeling languages focused on technologies for developing languages and tools that allow domain experts to develop system solutions efficiently, i.e., domain-specific modeling languages (DSMLs) [3, 4]. A new generation of complex software-intensive systems, for example, smart health, smart grid, building energy management, and intelligent transportation systems, presents new opportunities for leveraging modeling languages. The development of these systems requires expertise in diverse domains.

Consequently, different types of stakeholders (e.g., scientists, engineers and end-users) must work in a coordinated manner on various aspects of the system across multiple development phases. DSMLs can be used to support the work of domain experts who focus on a specific system aspect, but they can also provide the means for coordinating work across teams specializing in different aspects and across development phases. The support and integration of DSMLs leads to what we call the globalization of modeling languages, i.e., the use of multiple languages for the coordinated development of diverse aspects of a system. One can make an analogy with world globalization in which relationships are established between sovereign countries to regulate interactions (e.g., travel and commerce related interactions) while preserving each countryś independent existence.

Challenge #2: globalization of domain-specific languages

3.2 Chosen diversity: proactive diversification of computation semantics

We see an opportunity for the automatic diversification of program's computation semantics, for example through the diversification of compilers or virtual machines. The main impact of this artificial diversity is to provide flexible computation and thus ease adaptation to different execution conditions. A combination

of static and dynamic analysis, could support the identification of what we call "plastic computation zones" in the code. We identify different categories of such zones: (i) areas in the code in which the order of computation can vary (e.g. the order in which a block of sequential statements is executed); (ii) areas that can be removed, keeping the essential functionality [5] (e.g., skip some loop iterations); (iii) areas that can replaced by alternative code (e.g., replace a try-catch by a return statement). Once we know which zones in the code can be randomized, it is necessary to modify the model of computation to leverage the computation plasticity. This consists in introducing variation points in the interpreter to reflect the diversity of models of computation. Then, the choice of a given variation is performed randomly at runtime.

Opportunity #2: flexible computation

4 Diversity of runtime environments

4.1 Imposed diversity: diversity of devices and execution environments

Flexible yet dependable systems have to cope with heterogeneous hardware execution platforms ranging from smart sensors to huge computation infrastructures and data centers. Evolutions range from a mere change in the system configuration to a major architectural redesign, for instance to support addition of new features or a change in the platform architecture (new hardware is made available, a running system switches to low bandwidth wireless communication, a computation node battery is running low, etc).

In this context, we need to devise formalisms to reason about the impact of an evolution and about the transition from one configuration to another [6, 7]. The main challenge is to provide new homogeneous architectural modelling languages and efficient techniques that enable continuous software reconfiguration to react to changes. The main challenge is to handle the diversity of runtime infrastructures, while managing the cooperation between different stakeholders. This requires abstractions (models) to (i) systematically define predictable configurations and variation points – see also the challenge of Section 2 – through which the system will evolve ; (ii) develop behaviors necessary to handle unpredicted evolutions.

Challenge #3: effective deployment and adaptation over heterogeneous platforms

4.2 Chosen diversity: diversity of distribution and deployment strategies

Diversity can also be an asset to optimize software architecture. Architecture models must integrate multiple concerns in order to properly manage the deployment of software components over a physical platform. However, these concerns can contradict each other (*e.g.*, accuracy and energy). This context, provides new opportunities to investigate solutions, which systematically explore the set of possible architecture models and establish valid trade-offs between all concerns in case of changes.

Opportunity #3: continuous exploration and improvement of software architecture

5 Diversity of failures

5.1 Imposed diversity: diversity of accidental and deliberate faults

One major challenge to build flexible and open yet dependable systems is that current software engineering techniques require architects to foresee all possible situations the system will have to face. However, openness and flexibility also mean unpredictability: unpredictable bugs, attacks, environmental evolutions, etc. Current fault-tolerance [8] and security [9] techniques provide software systems with the capacity of detecting accidental and deliberate faults. However, existing solutions assume that the set of bugs or vulnerabilities in a system do not evolve. This assumption does not hold for open systems, thus

it is essential to revisit fault-tolerance and security solutions to account for diverse and unpredictable faults.

Challenge #4: adaptive software resilience

5.2 Chosen diversity: diversity of and redundancy of software components

Current fault-tolerance and security are based on the introduction software diversity and redundancy in the system. There is an opportunity to enhance these techniques in order to cope with a wider diversity of faults, by multiplying the levels of diversity in the different software layers that are found in software intensive systems (system, libraries, frameworks, application). This increased diversity must be based on artificial program transformations and code synthesis, which increase the chances of exploring novel solutions, better fitted at one point in time. The biological analogy also indicates that diversity should emerge as a side-effect of evolution, to prevent over-specialization towards one kind of diversity.

Opportunity #4: synthetic, emergent software diversity

References

[1] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag, 2005.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," tech. rep., Carnegie-Mellon University Software Engineering Institute, November 1990.

[3] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proc. of Future of Software Engineering*, pp. 37–54, 2007.

[4] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Proc. of the Int. Conf. on Software Engineering (ICSE)*, pp. 471–480, 2011.

[5] Z. A. Zhu, S. Misailovic, J. A. Kelner, and M. C. Rinard, "Randomized accuracy-aware program transformations for efficient approximate computations," in *Proc. of the Symp. on Principles of Programming Languages (POPL)*, pp. 441–454, 2012.

[6] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*, pp. 1–26, 2009.

[7] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[8] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Engineering*, vol. 1, no. 2, 1975.

[9] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proc. of the Symp. on Security and Privacy (S&P)*, pp. 120–128, 1996.