Model Driven Architecture (MDA) and Component-Based Software Development (CBSD)

Prof. Dr. Uwe Aßmann, Technische Universität Dresden,
uwe.assmann@tu-dresden.de
http://st.inf.tu-dresden.de
http://www.rewerse.net/i3

In embedded software development, designers of product lines have to take both variations and extensions into account. Variations occur when modules are implemented differently or on different underlying architectures. Extensions are unplanned functional additions, resulting from product line evolution. This paper explores some universal concepts to combine both requirements.

Two major approaches to achieve variability and extensibility in a product line are model-driven architecture (MDA, by OMG) [MDA] and component-based software engineering (CBSE). Within MDA, the re-usable skeletons of applications are referred to as *Platform-Independent Models* (PIMs). A PIM captures the architecture and the algorithmic issues that are independent of all platforms. It is *translated* towards application models, specific for each execution platform and enriched by platform-specific information (Platform-Specific Models, PSM). These PSMs are then completed by hand towards the code of the products. The variability comes with the PSMs: the more PSMs are produced, the more products can be sold. Component-based software engineering (CBSE) serves the same goal. Here, *frameworks* and *components* play the role of PIM and PSM: a framework is instantiated towards an application by filling its *hooks* with components. However, although serving similar goals, both approaches differ in the way in which the application skeletons are instantiated: PIMs are translated towards applications; frameworks are linked, composed, or connected with components. Is there a way to combine both approaches? In other words, how to embed components into MDA, i.e., how to build, design and use MDA components?

Luckily, the way is not far, because MDA has a background in commonality/variability analysis. Taking a closer look, MDA is a design approach in which variability plays a major role: to build a product line, a PIM is extended to several PSMs, variants specific to a platform. Historically, the first approach to commonality/variability design has been Parnas' *information-hiding-based design* [Parnas]. In this approach, variabilities (design decisions that change) are separated into modules with fixed interfaces. When design decisions change, the implementations of these modules may change, without this having an impact on the interface. Clearly, this approach facilitates evolution, is robust against changes and well suited for product lines, since variants can be segregated into product-specific modules. However, Parnas' modular design method is based on *explicit composition interfaces,* which do not play any role in MDA.

This difference, however, can be explained, if *planned variability* in product lines is conceptually distinguished from *unforeseen extensibility* in software evolution. Clearly, a designer of a product line has knowledge where products vary, so that she can decide where variation points are inserted into a core framework, and which contracts guide their instantiation (*commonality/variability thinking*). On the other hand, software evolution is triggered by a customer who changes his requirements, and since such a change cannot be foreseen, the designer will not be able to plan how the software has to be *extended.* Hence, to prepare evolution, a designer also

needs to reflect about *stability/extension issues*. At least, this requires that a designer has to prepare for *implicit extension points* at which the skeletons *can potentially* be extended (also called *join points* [AOSD] or *implicit hooks* [ISC]). And this explains one difference between information-hiding based design and MDA: in Parnas' method, frameworks are varied at explicit variation points (interfaces), whereas in MDA, implicit extension points are employed.

These arguments lead to some interesting consequences. First of all, MDA is not only about platform issues, but rather about systematic variability. It is possible to base a PIM on templates, modules, and generic components, in short, all component models that use explicit variation points. With these techniques, a PIM can be varied towards products with systematic variations filling the explicit variation points – the degree of re-use depends only on the abstraction of the employed component model. Secondly, MDA can also be used for software evolution, if *grey-box* component models are employed that support unforeseen extension through implicit extension points. These new models, such as aspects [AOSD], hyperslices [HyperJ], role models [Roles], or fragment components [ISC] have been introduced to allow for merging and extension of components. With such a grey-box component model, a PIM can be extended by new components that are integrated at implicit extension points (join points). We also say that we *weave* an extension into a core model. With this grey-box technology, a PIM can be evolved in unforeseen ways, and MDA can be employed as an extension technology. Thus, in the future, there will be at least two major categories of MDA: the *parametric or generic MDA* for variability, based on black-box component models with explicit variation points, as well as the *extensible MDA* for evolution, based on grey-box component models with implicit extension points.

One problem remains: Who will build all the necessary tools, i.e. the template expanders and extension weavers for the multitude of specification and programming languages? Can we build template processors and weavers that work universally for all languages? Or, in other words, how can we build *universally generic* and *universally extensible languages*? Languages, that are suitable for universal templates and aspects? In the last years, our group has found a way to build grey-box component models for every language [REWERSE]. Given a metamodel of a language L, a fragment component model can be systematically generated for L, so that a re-use-oriented *add-on language Reuse-L* results, in which fragment components can be composed. This implies that a base language need not take precaution for genericity, extension, nor composition; instead, all necessary constructs are *derived* in the re-use language add-on and come for free. Since this principle is universal, grey-box component models for modeling and specification languages come for free, including attractive composition techniques, such as templates, semantic macros, views, role models, and aspects. And finally, using these principles, universal template expanders and aspect weavers can be built for all languages. Currently, our group works on such a generic toolset, *reuseware*, which can be downloaded from Sourceforge [Reuseware].

At the moment, UML is the main language for modeling in MDA. Thus, a grey-box UML component model seems to be indispensable for a fully generic and extensible MDA. Luckily, with add-on reuse languages, this component model should come for free, including UML template processors and weavers. Even, if in the future other languages are employed in the MDA stack, the universal technology will continue to work, so that on every stack level of the MDA re-use can be planned and unforeseen extensions can be provided for. This paves the way for true MDA components, both for commonality/variability and stability/extension scenarios.

References

[AOSD] The Aspect-Oriented System Development (AOSD) Community http://www.aosd.net.
[HyperJ] Ossher, H. et. al. Multi-Dimensional Separation of Concerns. The Hyperspace Approach. http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm
[ISC] U. Aßmann. Invasive Software Composition. Springer, 2003.
[MDA] Model-Driven Architecture. OMG http://www.omg.org/mda
[Parnas] D. L. Parnas, On the Criteria to Be Used in Decomposing Systems into Modules,
Communications of the ACM, 15 (12), pp 1053-1058, Dec. 1972
[Reuseware] J. Johannes, J. Henriksson. The Reuseware toolkit. http://reuseware.sourceforge.net.
[REWERSE] Reasoning on the Web (REWERSE). EU Network of Excellence. Working Group I3 „Composition and Typing“. http://www.rewerse.net/i3
[Roles] T. Reenskaug. Role Modeling. http://heim.ifi.uio.no/~trygver