Softw Syst Model (2007) 6:1–2
DOI 10.1007/s10270-006-0041-9

EDITORIAL

Does model driven engineering tame complexity?

Robert France · Bernhard Rumpe

Published online: 25 January 2007

Advocates of model-driven (software) engineering (MDE)[1] tout the need to raise the level of abstraction at which software is conceived, implemented, and evolved to better manage the inherent complexity of modern software-based systems. Examples of MDE-related languages, technologies, and techniques that have been proposed as "tamers" of inherent complexity are languages supporting multi-view modeling of systems (e.g., the UML), metamodeling approaches to specifying model transformations, metamodeling environments for creating and using domain specific languages, megamodeling environments for manipulating and managing models, and aspect-oriented modeling environments supporting multi-dimensional separation of concerns and composition. Experiences suggest that some forms of MDE technologies may introduce significant accidental complexities that can detract from their use as managers of inherent software complexity.

The degree of accidental complexity is not surprising given that we're still in the early phases of MDE. Few would argue that we fully understand how to effectively model complex software systems. It is unreasonable to expect early perfect solutions when tackling the wicked problems facing MDE researchers and technologists. If we are to use MDE to tame the complexity of developing dependable software systems, it is important to recognize accidental complexities and their root causes so that research can be focused on minimizing them. MDE research progress can be understood in terms of the extent that research products help to reduce the accidental complexities in existing techniques and technologies without introducing other significant accidental complexities.

So where do the accidental complexities come from? As an example, let us consider the case of specifying transformations on UML models using the Query, View, Transformation (QVT) standard specification language. A group of MDE research students at Colorado State University attempted to use the QVT to specify transformations that would take source UML model describing systems in middleware-independent terms and produce target middleware-specific models of systems. The students specified the source model patterns using metamodel elements and defined relationships between source and target elements. The students expected that they would spend most of their effort specifying the relationships between source and target model elements. Specifying the source patterns turned out to be arduous and error-prone. First the students had to navigate the UML 2.0 metamodel to find the relationships they needed to define in the source patterns for class and sequence model elements (they restricted their attention to transforming class and sequence models). Navigating the large UML 2.0 metamodel is a time-consuming and frustrating task. The source patterns that the students produced were large. In the case of one transformation, the students produced a three page specification of the source pattern. Reading the specifications requires significant effort. It was clear that the students spent significant effort producing the specifications, but the result was not easy to comprehend. It can be argued that per-

R. France
Colorado State University,
Fort Collins, Colorado, USA

B. Rumpe (✉)
Technische Universität Braunschweig,
Braunschweig, Germany
e-mail: Bernhard.Rumpe@sosym.org

[1] If you're really curious about our use of the term MDE rather than the other more popular labels jump to the last paragraph of part one.

sons with expert knowledge of the UML metamodel and with significant experience in specifying transformations would have produced a more concise model in a shorter time. However, there are currently relatively few experts and developing such expertise will take time and effort. Developing expertise in modeling is challenging, and thus one can expect that developing expertise in manipulating models to be just as or even more challenging.

Accidental complexities are inevitable when there is very little significant experience on which to base technologies. For this reason it should not be surprising that use of the UML metamodel and the QVT standard can give rise to accidental complexities. One can also expect accidental complexities in early technologies associated with emerging paradigms such as aspect-oriented modeling (complexities arising from separation and composition of overlapping views) and megamodeling.

Introduction of accidental complexity however might be unavoidable. This is true for other computer science technologies as well: The more functionality a device or program provides, the more technological issues we have to deal with. In early years of programming, roughly 90% of a program was concerned with providing the primary services, today it is only about 40% — the rest deals with platform-specific technologies, GUI, interoperability, security and other dependability concerns. A similar effect can be observed when using a cell-phone or the menus of a Microsoft program. However, the producers of these have understood to hide at least some of the complexities through well organized interfaces and have put much effort making their products usable. In the MDE domain we also need to work on making technologies more usable.

The research community would be better served if authors proposing new techniques and technologies discussed the known and anticipated accidental complexities that can arise as a result of their use. This can help them put their descriptions of future or planned work in some context and can help others proposing alternative approaches to contrast their approaches in terms of accidental complexities they avoid.

Contents in this issue

This issue contains one regular paper "Implementing Associations: UML 2.0 to Java 5", in which the authors *Dave Akehurst, Gareth Howells,* and *Klaus McDonald-Maier* discuss a variety of code patterns suitable for implementing the semantics of UML associations using the Java 5.0 programming language. Although this may seem to be a relatively easy task, the paper uncovers a variety of issues that need to be considered when implementing associations. The authors also describe how tool support for the implementation patterns can greatly speed up and improve the process of implementing models in a Model Driven Development project.

The second part of this issue contains the special section on "Software Engineering and Formal Methods" organized by *Jorge Cuellar* and *Zhiming Liu.* It consists of three papers and shows that formal models can play vital roles in model based software development, even when it requires practitioners to be directly exposed to the formal concepts.

We hope you enjoy reading the articles in this issue.

Robert France, Bernhard Rumpe
Editors in Chief