Softw Syst Model (2004) 3: 1–3 / Digital Object Identifier (DOI) 10.1007/s10270-004-0052-3

Editorial

In search of effective design abstractions

Robert France, Bernhard Rumpe

Published online: 24 February 2004 –

The perceived popularity of the modeling languages such as the UML may lead some to believe that there is wide-spread appreciation of the value of modeling in the software development industry. Informal polls in North American trade journals seem to indicate otherwise. Use of the UML seems to be limited to the use of use cases for requirements and class diagrams for graphically representing programs. The polls also seem to indicate that awareness of the Object Management Group's Model Driven Architecture (MDA) is not widespread. As MDA is currently the most widespread model-based development approach this strongly indicates that much needs to be done to convince practicing developers of the value of model-driven development approaches in general. But, we expect to see in 2004 a surge in modeling tools in particular those that claim to be "MDA-Compliant". Tool support is essential to realizing the vision of model-driven development, but use of tools without a solid understanding of the principles and methods they support can lead to failed projects and faulty perceptions of model-driven development. In this editorial we reflect on the value of the abstraction principle in software development and on how modeling approaches can support this principle.

The increasing complexity of evolving large software systems has generated a renewed interest in software modeling techniques. Inherent software complexity can arise from a number of sources, for example, the need to support diverse and complex data and computational resources, and diverse users with competing needs. Rapidly evolving software infrastructure can also contribute to the complexity of creating and evolving software systems. For example, IT departments are faced with the task of evolving large integrated systems to take advantage of new middleware, communication or other platform technologies in order to maintain or gain competitive advantage. The rapid rate of change in these technologies is a source of considerable concern: the complexity of the software makes changing the software an error-prone and arduous task that takes considerable time. Managing software complexity can be characterized as a search for the "right" abstractions. Developers are seeking good abstractions that can help reduce the time and effort needed to create and evolve dependable systems.

Past research on design abstractions has produced structuring techniques that are based on functional abstractions (e.g., Structured Design), data abstractions that encapsulate behavior and state of a conceptual entity (Object-Oriented development), and service-based abstractions in which data and functional elements pertaining to a set of provided services are encapsulated in units called components (Component-Based development). There are numerous publications that describe how these abstractions can be used to manage complexity, and there is some quantitative context-specific evidence of their effectiveness. Today, the rate of change in software platform and system integration technologies (e.g., middleware technology) is spawning a new generation of complex software. For example, the pervasive and open nature of the internet present developers of internet-based software with challenges that are significantly different from those faced by developers of software built for a machine on a restricted network. Designers of these new systems have to deal with a myriad of complex interdependent dependability concerns (e.g., security, availability, error recovery, service integrity). Research on model-driven and aspect-oriented development can provide the abstraction mechanisms needed to manage the complexity of this new breed of software. An underlying theme in the discussion is the need for design techniques that allow developers to define abstractions in many dimensions.

As one of the large and innovative industrial consortia the Object Management Group (OMG) has responded to the growing complexity of software with an industry-driven initiative called Model Driven Architecture (MDA). MDA technologies are intended to raise the level of abstraction at which developers conceive and implement software. In MDA, software models are the primary artifacts of software development. The key characteristic of an MDA approach is the use of abstractions to separate technology-specific concepts from technology-independent concepts. Technology-independent models are composed with technology-specific models (e.g., models that describe middleware and other infrastructural elements) to obtain a design model that can be transformed to code. Ideally, changing an application to exploit new technology would require composing the technology-independent model with a new set of technology-specific models, and the composition and subsequent transformation to deployable code would be accomplished in a cost-effective and timely manner. This is an appealing view of software development, but accomplishing the ideal requires significant advances in modeling theory. Furthermore, a sound and elaborated modeling theory would allow us to transform, compose and refine models in a much more general way than current MDA proposes.

Effective abstraction mechanisms are critical to the success of model-driven development. Some of the more widely used modeling techniques allow developers to separate concerns to manage software complexity. The choice of abstractions influences the design structure and can make it easier or more difficult to evolve a system. A decision to structure a design with respect to a set of design concepts can make it necessary to spread information pertaining to set of equally important design concepts across design units. The problem is that understanding and evolving the latter set of concepts is difficult because they are scattered and tangled across the design model. Recent work on Aspect-Oriented Programming (AOP) addresses this problem at the programming level. One also needs to support multi-dimensional separation of concerns at other stages of development. Work in this area has given rise to a field of research labeled Aspect-Oriented Modeling (AOM). In an AOM approach, a primary model reflects decisions used to determine the core architecture of a design, and aspects are localized descriptions of design concepts that cannot be encapsulated in the primary design structure. Composing a primary model with aspects results in a comprehensive design model. Examples of design concepts that can be usefully described by aspects are design elements that address pervasive security and fault tolerance concerns. Aspects can also describe technology-specific design concepts, and thus AOM can be used as a vehicle for approximating the MDA vision. Treating aspects as patterns (e.g., security and middleware design patterns) facilitates reuse of aspects.

AOM can also provide support for balancing competing design objectives. Security, fault tolerance and other dependability and functional objectives often compete with each other in the sense that meeting an objective negatively impacts the degree that another can be satisfied. In such situations, developers should consider alternative ways of meeting objectives and make trade-offs in order to balance competing objectives. Obtaining a design that "balances" the concerns can be challenging using existing design techniques. With an AOM approach, the competing design concepts can be modeled as aspects to allow developers to plug in and unplug alternatives.

In 2004 we expect to see growing interest in the integration of aspect-oriented, pattern-based, and model-driven development techniques. The results should provide a good foundation for methods and tools that effectively support model-driven development of complex software systems.

Papers in this issue

We are pleased that *Cris Cobryn* found the time in his very busy schedule to publish an Expert Voice paper. The paper, "**UML 3.0 and the future of modeling**", relates the history of the UML definition with the future standardization efforts, and some open challenges for model-driven development.

In their regular paper "**OCL 1.4/5 vs 2.0 Expressions: Formal semantics and expressiveness**" the authors *María Victoria Cengarle* and *Alexander Knapp* present a type system, an operational and a denotational semantics for the expression sublanguage of the Object Constraint Language. Their results demonstrate that the changes introduced in OCL 2.0 restrict its expressivity to primitive recursion while OCL 1.4/5 was computationally complete. This foundational result shows how important it is to carefully design a modeling language to avoid expressivity problems that strongly impact practical applicability.

The authors *Carlos Rossi, Manuel Enciso, and Inmaculada P. de Guzmán* in their paper "**Formalization of UML state machines using temporal logic**" use a variant of temporal logic that allows the definition of time points, intervals, and dates. They demonstrate how to map the most important features of Statecharts into the temporal logic variant.

An innovative extension to visual notations is presented in the regular paper "**Nesting in Euler Diagrams: syntax, semantics and construction**" presented by *Jean Flower, John Howse, and John Taylor*. Nesting of diagrams is supported in two popular diagram types: Venn diagrams in set theory and Statecharts. In this article the authors provide a general foundation to nesting of diagrams and their interpretation in the context of visual modeling.

In the final regular paper of this issue "**The OsMoSys approach to multi-formalism modeling of systems**" the authors *Valeria Vittorini, Mauro Iacono, Nicola Mazzocca, and Giuliana Franceschinis* introduce an approach to building models that rely on several formalisms. They relate those models at a syntactic level using meta-modeling techniques and thus gain interoperability between different notations.

We hope you enjoy reading these articles,

Robert France, Bernhard Rumpe
Editors in Chief