Software & System Modeling (2006) 5(1): 1–2
DOI 10.1007/s10270-006-0008-x

EDITORIAL

Robert France · Bernhard Rumpe

Modeling the Complex Living World

Published online: 14 March 2006

In the software development domain, models of software systems are typically used to understand and predict properties of the system or to produce implementations. A model is normally created before the software system is implemented and, in many cases, acts as a specification of behavioral and structural properties that must be present in implemented systems. In theory, implementations can be generated from formal models in a manner that ensures their correctness with respect to the models.

In other non-engineering disciplines, models are used differently. For example, physicists use models primarily to understand and explain phenomena that occur in the world around them. They build models that are consistent with their observations of the phenomena, and they test the models to determine their fidelity and the circumstances under which the models make accurate predictions. Unlike software models, formal models of physical systems typically describe continuous behavior (with very few non-continuous disruptions) and therefore use concepts from continuous mathematics (e.g., differential equations).

It may seem that scientists in non-engineering disciplines have very little use for software modeling techniques, but the complex problems that are currently tackled in the Life Sciences area indicate otherwise. Scientists in the Life Sciences tackle highly-complex problems that involve study of organs, cells, proteins and organic molecules that exhibit continuous as well as discrete, non-deterministic behavior that can be described in terms of state transitions. Bio-technological models describe state-based phenomena and are primarily used to understand the phenomena and to predict behavior in a variety of situations. Accurate models pave the way for the engineering of medicines and for the development of sophisticated "biological tools" to further improve our lives.

Scientists in Life Sciences currently use software modeling techniques to describe discrete behaviors. What needs to be determined is whether current modeling techniques actually meet the needs of scientists in the Life Sciences. Specifically, answers to the following questions are needed: Are the concepts provided by current software modeling languages sufficient to describe the discrete aspects of phenomena in Life Sciences? Can scientists in the Life Sciences benefit from the use of domain-specific variants of general purpose software modeling languages? Furthermore, the issue of how to disseminate knowledge about software modeling techniques in the Life Sciences needs to be addressed. Is there a need for software modeling training and education programs that target scientists in the Life Sciences or is there a need for a program that integrates Life Sciences and Computer Science (as extension to bio-informatics)?

We would like to see SoSyM become a vehicle for communicating high-quality work that involves analyzing the application of software modeling techniques in domains such as the Life Sciences. We strongly encourage authors working on novel applications of software modeling techniques in domains such as the Life Sciences, Economics and Social Sciences, to submit papers describing their results to SoSyM.

Contents in this issue

Trygve Reenskaug in his expert voice paper "*The BabyUML Discipline of Programming*" examines the state of art in programming over the last 40 years and offers a way to enhance the control a programmer has its program. He discusses a subset of UML that can be used to develop programs that are more reliable and less error prone, because it allows the programmer to develop a better understanding of the program.

The second part of this issue contains five regular papers. The regular paper "*Partitioning of Perfect Synchronous Reactive Specifications to Distributed Processors using μ-Charts*" by *Peter Scholz* introduces a specific variant of Statecharts that targets the modeling of distributed systems. μ-Charts are used to partition a model into several sub-models that can be implemented and deployed independently on different processors.

Sometimes restricting expressivity of a language allows one to deduce more from its models. In the regular paper "*Exploiting Practical Limitations of UML Diagrams for Model Validation and Execution*" *Friedrich Steimann* and *Heribert Vollmer* discuss how to use UML's limitations to derive decidable static model analysis algorithms. The algorithms provide results that can help a designer gain more insight into what is modeled.

A basic tenet of modeling is the prediction (and later controlling) of costs, timeline and quality of the system to develop. The third regular paper "*A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues*" by *Silvia Abrahao, Geert Poels,* and *Oscar Pastor* describes metrics to determine the size of software based on a model of the development process.

In the fourth regular paper "*A Powertype-based Metamodelling Framework*" *Brian Henderson-Sellers* and *Cesar Gonzalez-Perez* describe a new approach to the construction of metamodels that eases both development and understanding of the constructed metamodel.

Ashley McNeil and*, Nick Simons* in the last regular paper "*Protocol Modelling. A Modelling Approach that Supports Reusable Behavioural Abstractions*" introduce a behavioural modelling approach based on the concept of a protocol machine that aims to provide better encapsulation.

We hope you enjoy reading the articles in this issue.

Robert France
Bernhard Rumpe
Editors in Chief