Softw Syst Model (2011) 10:281–282
DOI 10.1007/s10270-011-0208-x

EDITORIAL

Linking models and their storage artifacts

Bernhard Rumpe · Robert France

Published online: 25 June 2011

Many developers working on model-based development projects experience a problem that highlights the need for better model management technologies. If someone wants to adapt an OO class that was generated from a model, the first thing is to locate the "sources" used to generate the class (i.e., the model elements used to produce the class implementation). This turned out to be a surprisingly difficult and time-consuming task. It is worth shedding some light on why this task typically is difficult if only to remind us that research on software modeling needs to extend its focus to include model management issues if we are to see more widespread adoption of software-modeling approaches.

The problem of locating code generation sources can be traced to inadequate support for managing stored representations (artifacts) of models. Models need to be stored in a manner that supports convenient and efficient retrieval of model elements by both humans and software. This is particularly important when a system is modeled using multiple notations or languages; the need to check consistency across the different modeling views in these systems requires efficient model element retrieval mechanisms. Current modeling tools handle different modeling views of a system by integrating them into a single, integrated artifact that is stored in a database, a file or even in CPU storage. This monolithic approach has a number of drawbacks, including:

(1) Poor support for exploratory modeling in a team development environment: If modelers want to explore alternative ways of modeling an aspect of a system, possibly with known inconsistencies with other parts of the model, they must integrate their models with the shared model, and thus risk jeopardizing the work of team members working on other parts of the system model.
(2) Poor support for separation of parts: Modeling subprojects with clearly specified interfaces are not easily defined. Thus, parallel model development opportunities can be missed in early development phases.
(3) Poor support for version control: Version control technologies are considerably hampered by the shortcomings mentioned above.
(4) Poor support for incremental and thus efficient, agile code generation: Supporting code generation mechanisms tend to be implemented as a monolithic process, in which changes to a model typically requires that the code generation process be run using the entire model as an input.

Simply put, existing support for monolithic integrated internal representations of models may be adequate for some single person or small team development environments, but they fall short when used in environments in which different aspects of a system are modeled in parallel by different team members. Instead of one big, integrated artifact containing every piece of information captured in different modeling views, it is better to have clear and dedicated small modeling languages, that allow modelers to create comprehensible models dedicated to specific purposes. These models should have clearly defined relationships with the storage artifacts that persist the models. Also a set of models needs clear forms of how the models consistently "fit together": i.e. how information captured by model elements is "transported" across models and used in importing models. These relationships

B. Rumpe (✉)
RWTH Aachen, Aachen, Germany
e-mail: bernhard.rumpe@sosym.org

R. France
Colorado State University, Fort Collins, CO, USA
e-mail: france@cs.colostate.edu

must also be conveniently represented at the storage level, for example, as relationships among the contents of storage artifacts.

To achieve the above modeling languages and tools need to provide support for modularizing models through the notion of "modules", as is done in the programming language domain. Many modern and widely used programming languages provide a clear definition of a module (e.g., in the OO paradigm a module is a class while in the procedural paradigm a module is a function or procedure). Furthermore modules are used to encapsulate information that are accessed through clearly defined interfaces. If models provide externally usable information through "model interfaces", these interfaces can be imported by other models and thus allow management of relationships between models.

At this point most of you are wondering how the above discussion on team development and modularity relates to the problem we kicked this editorial off on. What does the notion of modularity at the model level have to do with the problem of retrieving model elements? Note that in the more established programming languages the place where a module's definition and its implementation is stored in the development space can easily be inferred from its name. For example, Java melds package and directory paths with the class and filename. We propose that a similar approach needs to be taken at the model level to make it clear to a human where model elements are stored in the model development environment. At the programming level such naming schemes support more efficient compilation and incremental program development and compilation, and thus we can expect similar benefits at the modeling level, where "compilation" takes the form of code generation.

We need to better understand how best to manage models in a team-based modeling environment. In particular, we need to establish a clear relation between models and their storage artifacts (i.e., their internally stored representations). Some of the pressing questions we must address as researchers are: How can we store models as independent but related artifacts? How can we capture information spread across different modeling views in "model modules" with clearly defined interfaces? How do we manage interfaces between models? Do we store them separately or retrieve them from the model or the generated artifacts?

There are other, maybe better, ways in which the above problems may be tackled and there are also other important model management issues that are not touched upon in this editorial. We welcome submissions that describe solutions to these and other model management issues to SoSyM.

We hope you enjoy reading this issue.

Robert France, Bernhard Rumpe
Editors in Chief