Aspect-UML-Driven Model-Based Software Development

László Lengyel, Tihamér Levendovszky, and Hassan Charaf

Visual Modeling and Transformation System (VMTS) is an n-layer metamodeling environment which supports editing models according to their metamodels, and allows specifying OCL constraints. Models and transformation steps are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel ("visual vocabulary"). Also, VMTS is a UML-based model transformation system, which transforms models using graph rewriting techniques. Moreover, the tool facilitates the verification of the constraints specified in the transformation steps during the model transformation process.

Many concerns pertaining to software development have a crosscutting impact on a system. Using current technologies (Unified Modeling Language and Object-Oriented Programming), such kinds of concerns are difficult to identify, understand, and modularize at design and implementation time, as they cut across the boundaries of many components of a system. Crosscutting concerns typically include design constraints and features, as well as architectural qualities and system-level properties or behaviors, such as transactions, logging and error recovery.

Aspect-oriented (AO) techniques are popular today for addressing crosscutting concerns in software development. Aspect-oriented software development (AOSD) methods enable the modularization of crosscutting concerns within software systems. AOSD techniques and tools, applied at all stages of the software lifecycle, are changing the way in which software is developed in various application domains, ranging from enterprise to embedded systems.

Aspect-oriented modeling (AOM) is a critical part of AOSD that focuses on techniques for identifying, analyzing, managing and representing crosscutting concerns in software design. The design models consist of a set of aspects and a primary model that can be weaved together by AOM weaving. Most current implementations of aspect-oriented programming (AOP) start directly from programming level. First, aspects are identified either by source code reading or document reading. Second, the system will be implemented in the code level based on the current defined aspects.

To meet the limitation of AOP we combine AOP and AOM to utilize the strength of both approaches. AOM is used in the requirement and design phase to ensure a proper aspect-oriented design. It can test any conflicting situations and enhance optimization between aspects.

Some AOM researchers propose that the aspects and the primary model are woven together in a way that the aspect is integrated into the primary model before the code generation. However, a major problem of this solution is that the separation of the aspects and the primary model is lost once the weaving is done. In the VMTS approach, model transformations are used to generate CodeDOM tree from models. The CodeDOM tree is a language-independent model representation of the source code, from which the code is generated automatically. Therefore, in VMTS, the CodeDOM tree corresponds to the code phase. In our process, we propose to keep the separation until the code phase. AOM weaving is only used to test the validity of the aspect models. In this way, we can keep the primary model and aspect models separately in the coding phase. It is then easy to backtrack if there are any changes in the CodeDOM tree since the generated primary code can be mapped directly back to the primary and aspect models. Current work introduces the methods required to realize the approach: (i) VMTS Aspect-UML and (ii) a weaving method that composes aspect models and primary model as well as CodeDOM models. An extended Aspect-UML is needed to express aspect models. Pointcuts, joinpoints and advice cannot be expressed exactly using current UML tools. As aspects are context-specific and need initialize, Aspect- UML needs to be able to customize the aspect model based on different contexts. Another requirement for Aspect-UML is that the connections between aspects and classes should be detailed enough to make the tracking easy. This requires that both the advice and pointcuts are represented in the aspect model. Aspects should be handled on two levels: both at the modeling level (AOM) and the programming language level (AOP). At the AOM level, aspects are identified and woven together by AOM weaving to verify and optimize them. However, models are not woven together for the purpose of code generation based on a combined model. The actual weaving is performed on CodeDOM models.