Technische Universität Braunschweig

ISF

Lecture Model-Driven Software Development

# Learning Model-Driven Software Development

The lecture teaches students the fundamentals of model-driven software development with metamodels using state-of-the-art concepts and tools. Contents are conveyed in a combined lecture and exercise phase followed by a practical group-project phase.

**Technische Universität Braunschweig | Institute of Software Engineering and Automotive Informatics**
Dr.-Ing. Christoph Seidl | c.seidl@tu-braunschweig.de | Phone +49 (0) 531 391-2296

## Lecture Curriculum: Fundamentals of Model-Driven Software Development

### 1. Metamodels

A metamodel uses metaclasses with attributes and references to describe how models may be structured. Models each instantiate a metamodel to represent data in a structured way. Knowledge about models provided by the metamodel may be exploited in subsequent tools, e.g., to generate source code for loading, editing or saving.

### 2. Model Semantics

Static semantics encompasses well-formedness rules, which can be checked by evaluating constraints defined on a metamodel, e.g., that an entity needs to be defined before it can be assigned a value. Operative semantics describes how a model behaves if it is executed, e.g., a state machine model executed by an interpreter.

### 6. Model-to-Text Transformation

Many formats for the generation from models are textual, such as source code, webpages or reports. Model-to-text transformation offers specific means to generate a wide variety of textual formats, e.g., through template engines, which combine a large portion of static text with dynamic text retrieved from structured data in models.

**Models**

### 3. Textual Concrete Syntax

A metamodel may be perceived as the abstract syntax of a domain-specific language, i.e., the machine-processable concepts of the language. A textual domain-specific language may be created by adding a grammar with human-readable terms as concrete syntax. Using MDSD technology, parsers and editors may be generated.

### 5. Model-to-Model Transformation

Even for the same domain, there may be different metamodels so that data in the respective models is not directly compatible. Model-to-model transformation converts models instantiating one metamodel to conform with another metamodel for the purposes of data import/export, format conversion, compilation or optimization.

### 4. Graphical Concrete Syntax

A graphical domain-specific language may be created by defining graphical symbols and their connections as concrete syntax for the entities of the abstract syntax defined in a metamodel. Using MDSD frameworks, viewers and editors for the resulting graphical domain-specific language may be partially generated or implemented swiftly.

## Lecture and Exercise Phase

In class-room lectures, students learn the fundamental concepts of model-driven software development in three major blocks: metamodel creation (blue), concrete syntax (green) and model transformation (purple). Accompanying exercises introduce the most relevant tools for each area and provide a chance for students to get first-hand development experience through completing tasks specific to each topic. This puts students in the position to conceive and realize a model-based tool.

## Group-Project Phase

In a two-month period, groups of students work on a model-driven development project (orange) to get hands-on experience with the concepts and tools presented in the lecture and exercise. In an interative process, students propose and refine the topic of their project along with suitable implementation tools. In groups of four, students design and implement the previously envisioned model-based tool before they present the results to their supervisors and fellow students in a final presentation.