Motivation: Domain Specific Languages (DSLs)

- A DSL is a programming language specialized to a specific application domain:
  - common domain notions and notation
  - raise the abstraction level of solving problems in the domain
  - A DSL is usually implemented as a translation from the domain concepts to a general-purpose programming language:
  - complicated design solutions and algorithms
  - both high-level concepts of the DSL and low-level concepts of the execution platform

- In Model Driven Engineering: this complexity is coded into model transformations and code generators
  - The complexity is hidden from the DSL end-users
  - The DSL is hard to understand, learn, use, verify and maintain.

Challenges: formal specification of the DSL semantics

- A formal specification of the DSL provides an explicit definition of the DSL semantics, which is hidden in the code generator and/or in the model transformations.
- In an industrial context it is not common to have a formal specification of a DSL.
- An extensive tool support of a formal specification can realize its benefits, such as verification and validation possibilities.

Our solution: Model transformations from the DSL to Event-B

Model transformations: composition and instantiation

Generic specification of the DSL expressed in terms of generic specification modules, such as:

- DSL architectural components:
- DSL semantic features:

Instantiate and compose

Model transformations automatically generate an Event-B specification for each concrete DSL program.
Model transformations implement composition and instantiation of the generic specification modules.

- Reuse of verification results
- Scalability of Event-B specification

Resulting Event-B specification supports all semantic features and consists of instances of all necessary components

Event-B and Rodin: an extensive tool support

- Check that the specification is correct using automatic theorem provers.
- Execute Event-B specification in the ProB animator.
- Visualize Event-B specification in the Bmotion Studio.

Future work

- Distinguish reusable specification patterns for specifying DSLs
- Generalize our model transformations to support a framework for applying reusable specification patterns
- This will allow for analyzing and executing more DSLs

References