Overview of DSLE

- Model driven software engineering in general
- Grammars, and meta-models
- DSL Design
- Model transformations
- Code generation

DSL Design

Model-driven engineering

Goal:
- Raising the level of abstraction
  … from the computing domain to the problem domain

MDE combines:
- Domain-specific modeling languages
- Model transformations

---

Domain specific languages

“a language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain” [2]

Pros

| Expression of the solution in terms of domain concepts |
| Enhanced productivity, reliability, maintainability, and portability |
| Domain knowledge contained in language |
| Mostly concise and largely self-documenting |

Cons

| Cost of DSL implementation and education |
| Difficulty of finding the right scope and balancing between domain-specificity and general-purpose constructs |
| Solutions for a limited set of problems |
| Potentially less efficient code |

---

DSL Design

Domain-specific language design:

1. Domain analysis
2. Language design
3. Language implementation

Domain Analysis

Identifying the objects, operations and relationships between them

“A problem domain is defined by consensus, and its essence is the shared understanding of some community”[3]


Language Implementation Strategies

- Stand-alone
- Embedding
- Translation
DSL Design

- Domain Specific Language
  - Programming linguistics:
    - syntax
    - semantics
    - pragmatics (or methodology):
      the way the language is intended to be used
  - Requirements engineering for domain knowledge

Programming language vs.
- Specification language
  - Universal, any problem must be expressible
  - Focused on modeling not on implementation
- Domain specific language
  - Created for a specific problem domain
  - Implementable

Overview of DSLE

- Model driven software engineering in general
- Grammars and meta-models
- DSL Design
- Model transformations
- Code generation

Model transformations

- Models increase the level of abstraction
  - used for both hardware and software design
  - often manually translated into design documents and code
  - no guarantee for consistency between model, design and resulting code

- Models increase the quality of the code
  - via model checking

- Models increase the efficiency of software development
  - code generation
Model transformations

- Mappings are defined on meta-models
  - The mapping is defined with meta-models; one meta-level higher than the input and output models of the transformation
- Transformations are defined on models
  - A transformation implements a mapping

How are the meta-models of Xtext mapped to meta-models describing the abstract syntax?

General architecture:
- Parse/construct an input model (Xtext/GMF)
  - Map concrete syntax model to abstract syntax model
- Apply transformation from input to output model
- Unparse the output model

A model transformation takes input and
  - changes the input, or
  - produces output
  - transforms according to a predefined mapping
  - is used in “run time”

Two main categories of transformations
  - Vertical or horizontal transformations
  - Exogenous or endogenous transformations
Model transformations

- Vertical transformations
  - Source model is at a different level of abstraction than the target model
  - Examples of vertical transformation
    - refinement (specialization)
    - PIM $\Rightarrow$ PSM transformations
    - abstraction (generalization)

- Horizontal transformations
  - Source model has the same level of abstraction as target model
    - not to be confused with “meta-levels”
  - Examples of horizontal transformation
    - refactoring
    - merging

---

Model transformation

- Endogenous transformations:
  - between same meta-models

- Exogenous transformations:
  - between different meta-models

---

Model transformations

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endogenous</td>
<td>Refactoring</td>
<td>Refinement</td>
</tr>
<tr>
<td>Exogenous</td>
<td>Migration</td>
<td>Code generation</td>
</tr>
</tbody>
</table>

Further distinction: syntactic vs. semantic transformation
Model transformations

Approach

- Meta-metamodel
- Transformation Language
- Transformation Definition
- Target Metamodel
- Conforms to
- Conforms to
- Instance of
- Target Model
- Conforms to
- Source Model
- Metamodel
- Conforms to
- Source Model
- Meta-metamodel
- Input
- Transformation
- Output

Model transformations

- Model transformation formalisms
  - ATL
  - Xtend
  - Xtext
  - Xpand
  - QVT Relations
  - QVT Operations
  - QVT Core
  - ASF+SDF
  - Stratego/XT
  - VIATRA
  - Tefkat
  - ETL (Epsilon)
  - GrGen
  - ...

- Platform: Eclipse and EMF

Model transformations

- How are the meta-models of Xtext, for instance, mapped to meta-models describing the abstract syntax?

- General architecture:
  - Parse/construct an input model (Xtext/GMF)
    - Map concrete syntax model to abstract syntax model
  - Apply transformation from input to output model
  - Unparse the output model