Transformations for Abstractions

Eelco Visser

Software Engineering Research Group
Delft University of Technology
language = syntax + transformations
language = syntax + transformations
language = syntax + transformations
language = syntax + transformations

refactor
GPL program

compile
'machine' code

GPL program

optimize
docgen
documentation

GPL program
language = syntax + transformations
adding your own transformations

- model
- abstract
- GPL program
- docgen
- optimize
- GPL program
- compile
- 'machine' code
- refactor
- GPL program
- documentation
transformations provided by vendor

- model
- abstract
- GPL program
- refactor
- compile
- 'machine' code
- optimize
- GPL program
- docgen
- documentation
new transformations not first-class citizens
open prog env for new transformations

- GPL program
- 'machine' code
- GPL program
- model
- abstract
- GPL program
- compile
- optimize
- docgen
- documentation
- refactor
- ?
- ?
domain-specific languages

- DSL program (model)
- generate
- GPL program
- compile
- 'machine' code
embedded domain-specific languages

MetaBorg (OOPSLA'04)
combining dsls and transformations
combining edsls and transformations
research: extensibility of transformations
research: extensibility of transformations

apply transformation to host + guest language
research: extensibility of transformations

extend transformation to edsl without code duplication
mechanisms for extensibility

- parameterization
  - explicit extension points
  - variants, not unanticipated extension

- recomposition
  - no reuse of composition

- modular open extensibility
  - stratego definitions, haskell typedefs
  - only one extension at a time

- open extensibility with concurrent variants
  - mixins?
  - dynamically scoped definitions?
independent extensibility

<table>
<thead>
<tr>
<th></th>
<th>Assimilation</th>
<th>Optimize</th>
<th>Refactor</th>
<th>Trafo?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DSL1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>DSL2</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>DSL3</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

automatically derive extension of transformation?
project: transformations for abstractions

- Definition of domain abstractions
  - semantics of (embedded) domain-specific languages

- Mechanisms for open extensibility
  - open extensibility with concurrent variants

- Design of open transformations

- Independent extensibility
  - derivation of transformation extensions
model-driven software evolution

generate software from combinations of (embedded) domain-specific languages

issue: interaction between (embedded) DSLs
model-driven software evolution
model-driven software evolution

evolution of DSLs
model-driven software evolution

DSL evolution requires migration of DSL programs
model-driven software evolution

derive DSL programs from (legacy) GPL programs
model-driven software evolution

abstract

migrate
evolve

develop higher-level abstractions
Contribute

• Delft University of Technology
  – Software Engineering Research Group
  – 4 job openings
• Transformations for abstractions
  – 1 postdoc position (3 years)
• Model-driven software evolution
  – 1 postdoc position (3 years)
  – 2 PhD student positions (4 years)