Model-Driven Software Evolution
NWO/JACQUARD project 638.001.610

Eelco Visser
Arie van Deursen
conventional software development

'model' 'design'

'code'

GPL program

'compile'

'machine' code
conventional software maintenance

'model' 'design'

understand

modify

GPL program

compile

'machine' code

abstractions encoded in program
maintenance at low level of abstraction
domain-specific languages
model-driven engineering

raise the level of abstraction to a technical or application domain
automatically generate implementation code from model
problem1: interaction
multiple models / multiple dsls

generate software from combinations of domain-specific languages
model/model interaction

consider models as components / modules
what is interface of a model? what is the scope of model elements
model encapsulation; separate compilation
customization of generated code

not all customizations can be realized in models
generated code may need to be adapted
customize 'from the outside'

customization should never require direct modification of generated code.
Customization code must modify/interact with generated code.
What is the interface? Avoid exposing generation scheme.
customization code should be considered as part of the generator input. It should interact with (interface of) models, not with generated code.
embedded domain-specific languages

MetaBorg (OOPSLA'04)
DSLs for abstraction over libraries/frameworks
fine-grained interaction with 'host' code
language conglomerates mix DSL and GPL code
problem 2: evolution
evolution scenarios
regular evolution: adapt software to new requirements
implementation simply regenerated after modification of models
language evolution

language (syntax and/or transformations) evolve
model migration

language evolution requires migration of models
changes in the platform requires evolution of transformations
model extraction

derive DSL programs from (legacy) GPL programs
abstraction evolution

develop higher-level abstractions
modse objectives

- technology
  - model development environment
- generation
  - from model to code
- evolution
  - from code to model
- evaluation
model development environment

- connecting technological spaces
  - uml, sdf, xml, ...
  - grammars for language combinations
- unifying model and code transformation
  - model extraction from code
  - code generation
- language definitions in development env.
  - making a new dsl should be as easy as making a new class
generation – from model to code

• modeling business logic
  – scope and expressivity of DSLs
  – balance between generality and domain specificity

• model interaction
  – separation of concerns -> dependencies
  – modularity: encapsulation, interfaces
  – how to refer to elements in other languages?

• model composition
  – composition of whole systems from models
evolution – from model to code

• incremental model introduction
  – migrate part of legacy code base to models
  – models and code co-exist

• model reconstruction
  – harvest models from existing (legacy) code
  – agnostic: search for recurring patterns
  – reconstruct models for known DSLs

• model-based testing
  – validation of migration to models
evaluation

• risk/benefit analysis
  – return on investment: when does effort of dsl design and implementation pay off?
  – goal of MDE is to lower the threshold
  – factors for success and counter indicators

• methodological embedding
  – decision making process for adopting MDE
  – guidelines based on case studies and literature
partners

- Delft University of Technology, SERG
  - Eelco Visser, Arie van Deursen
- Getronics PinkRoccade
  - Thiel Chang, Jelle Gerbrandy
- Fortis / Amersfoortse
  - Lex van der Geest, John Adegeest, Duncan Doyle
- Ordina
  - Jos Warmer
- Interactive Objects
  - Thijs Reus
- ATOS Origin
  - Kees Kranenburg
job openings

• model-driven software evolution
  – 2 postdoc positions (3 years)
  – 2 PhD student positions (4 years)

• transformations for abstractions
  – 1 postdoc position (3 years)