Tracing Abstractions through Generation

Software Transformation Systems Workshop,
GPCE 2004

Karl Trygve Kalleberg*

University of Bergen  
<karltk@ii.uib.no>

Utrecht University  
<karltk@cs.uu.nl>

*)Work funded by the Norwegian Research Council
CodeBoost

Legend:
- framework
- extension point
- data

**Legend:**
- Pretty printer
- Source code
- C++ parser
- Semantic analysis
- Configuration files
- trafo_A
- trafo_B
- Pretty printer
- Optimized source code

**Software Transformation Systems Workshop 2004**

**Tracing Abstractions through Generation**
SDS: Software Development Foundation

Legend:
- framework
- core format
- data

- Python source code
- C++ source code
- Java source code
- Lisp source code

- Python parser
- C++ parser
- Java parser
- Lisp parser

- CSF
- sdoc

- HTML
- DocBook
- man pages
- LaTeX

Command Line Interface

Tracing Abstractions through Generation
Spoofax in-a-sketch

Basic pipeline

Legend:
- in progress framework
- extension point

C++
- Unparser
- Boxing
- Parser

Java
- Unparser
- Boxing
- Parser

C
- Unparser
- Boxing
- Parser

Stratego
- Unparser
- Boxing
- Parser

trafo_A

trafo_B

Documentation System

Documentation

Code Analyzer

Reports

Software Visualizer

Class Graphs

Call Graphs

Query Tools

Command Line Interface

Eclipse Extension

Tracing Abstractions through Generation
Problem description

Legend:
- domain configuration
- domain-specific code
- transformation
- compiler

Tracing Abstractions through Generation
Some sketched solutions

- **Transformation phase**
  - Use syntactically correct, semantically aware transformations
  - Support interactive replay

- **Maintain trace of abstraction throughout complete pipeline**
  - I.e. reverse arrows

- **Compilation phase**
  - Relate line numbers in generated source to domain abstractions
  - Problematic for glue code

- **Deployment phase**
  - Relate target abstractions to domain abstractions

- **Runtime phase**
  - Relate runtime exceptions to domain abstractions

- **Debugging phase**
  - Support language embedding
Concluding remarks

- Not fundamental research, so why bother?
  - Code generator pipelines seldom transparent
  - Another reason for people to write their own, custom transformation systems
  - Hampers productivity and happens often

- No obvious, established and employed techniques
- Can a “best-practice” be suggested?