Transformation Systems for DSLs, Architecture Styles, and Graphical Languages

David Wile
Teknowledge Corp.
Dwile@teknowledge.com

Two Points

¥ There are many approaches that provide (some of) the benefits of Domain Specific Languages
¥ Each of them is amenable to adaptation of the various technologies developed for context free language entry, analysis, and manipulation — generally, transformation.

Therefore: we should use past research efforts to guide future development in the DSL arena
Supporting the DSL Spectrum

¥ Language tailored to the problem domain
   —Focus on its idioms and jargon
   —Assume pre-defined infrastructure support
   —Thereby avoiding clumsy, overly general constructs

¥ Expensive to support fully
   —Alternative approaches
   —Varying degrees of support per approach

Approaches

¥ Abstract syntax
   —OO — Java, .NET
   —COM / CORBA
   —XML

¥ Syntactic
   —YACC
   —Synthesizer Generator, Popart, SDF

¥ Graphical
   —Acme — Architecture Styles
   —PowerPoint Briefing Associate (Ontology-based visualization)

¥ Interpreter / Language Extension
   —Haskell
   —Access
   —Excel
   —Generic programming
Syntax Support Tools

¥ Abstract Syntax — an intermediate representation capturing the essential concepts of the domain
¥ Language Specification — a set of constraints or templates to restrict designs
¥ Parsing — adherence of a design to the language specification
¥ Syntax-Direction — automated aid to constructing specifications that adhere
¥ Type Checking — imposing uniformly a set of more global constraints beyond the (generally local) syntactic constraints

Semantics Support Tools

¥ Semantics Specification Mechanisms and Issues
  — Attribute Grammars
  — Transformations — within a language
  — Translations — between languages
  — (Other) Homomorphisms — into algebraically similar structures
  — Establishing transformation validity
¥ Traversal Mechanisms
  — Metaprogramming Calculi — programs as data
  — Strategies — heuristics for transformation
  — Visitor patterns — a calculus for OO representations of AST transformations
¥ Debugging Aids — errors related to source specifications and data structures
Example of Lost Art

¥ Popart Translation: implicit homomorphism
   — $H \text{op}[a_1, a_2, \ldots, a_n] = (H \text{op})[H a_1, H a_2, \ldots, H a_n]$
   — Automatically look for a way to translate the pieces before composing the whole, via $H \text{op}$.

¥ Translate from, e.g. Java to Lisp
   — CF Syntax for each language
   — NT types for pattern variables

T-Java-to-Lisp = **translator** [Java,Lisp]

**rules**
\[
\begin{align*}
\text{if} & \quad (!\text{Bool#t} \quad (!\text{Stmt#thn}) \Rightarrow (\text{COND} \quad (!!\text{Sexp} \quad !!\text{Sexp} \quad !!\text{Stmt#thn}))) \\
!\text{ID#I}(!!\text{Exp#args}) & \Rightarrow (!\text{Atom#I} \quad !!\text{Sexp} \quad !!\text{Exp#args}) \\
!\text{Fexp#e}(!!\text{Exp#args}) & \Rightarrow (\text{APPLY} \quad !!\text{Sexp} \quad !!\text{Exp#args}) \\
\end{align*}
\]

* * *

Design Support Tools

¥ Re-engineering aids (inverses for each semantics specification mechanism)

¥ Design recording aids
   — Historical Development — keeping track of a design history
   — Pedagogical Development — when can a design history be replayed?
   — Development strategies — what should be tried first, and why?
   — Requirements-based Development — why are things as they are?
Points

¥ Many approaches for providing DS(L) support
¥ Past experience shows what peripheral tool support mechanisms can be beneficial
¥ Read relevant literature from before 1990!