Domain Analysis and Generative Implementation

ICSR5 1998

James M. Neighbors
Bayfront Technologies, Inc.
neighbor@BayfrontTechnologies.com

Generative Process

- Lessons from the waterfall model of Software Engineering.
- System construction using Domain Analysis not a simple flow through tools (it is intellectual property development not EDP).
- System construction must support *progressive deepening*.
- Evolving system in representations from analysis to code *at any given time*.

Domain Analysis

- Domain Analysis results in a domain that supports the specification and refinement of similar member systems.

- Each Draco domain contains:

Parser	intradomain	domain language source to internal form
Display	intradomain	internal form to source
Optimizations	intradomain	semantics, rules of exchange in a domain language
Components	**interdomain**	semantics, operational meaning, multiple refinements
Generators	intradomain	semantics, generation by program
Analyzers	intradomain	semantics, gather information about domain statement
Tactics	intradomain	combining above to refine out of a domain
Strategies	**interdomain**	combining above to refine full system across domains

- Interdomain connections are consciously limited and managed.

What is different about Draco?

- Actual **source form language** (e.g., SQL, OpenGL, VHDL) - not a library/catalog.

- Components contain **multiple refinements into other domains**, provides:
  1. high-level domain specific optimizations
  2. variety in implementation goals
  3. variety in implementation architectures

- Conditions and assertions on refinements address **reusability questions**
  1. Can a system description be refined to only the target domains?
  2. If so, what is a possible implementation?
  3. If not, what additional domains or refinements are necessary?

- Scale guaranteed by a **conscious tradeoff** between generator power and the ability to analyze the generator in operation.
  1. restricted refinement mechanism
  2. restriction of most domain parts to intradomain
  3. restricted power condition and assertion language
  4. only important when composing domains during refinement

Questions on DA Based Generators

- Is the generator general? Can a new domain be added without reprogramming the mechanism?
- Does the generator support domain composition? Do new domains reuse each other and not displace a single domain?
- Is the generator scaleable to programming in the large? Can the mechanism be analyzed in action on large problems using a large set of domains?

Conclusions

- Automatic Programming power function
- *Software Reuse* recognizes power function gains by reusing system artifacts.
- *Domain Analysis* recognizes power function gains by reusing analysis and design.
- *Draco* approach recognizes power function gains by having domains reuse each other.
- Power function fails to recognize the expense of putting technique in place.
- Availability & costs of software production and education will force issue.