History, Architecture & Evolution of Compilers

SENG 480A / CSC 576A (H. Muller)

Today: Holger Kienle (kienle@csr.uvic.ca)

Thursday:
History, Architecture & Evolution of Web Sites
(Daniel German)

Outline

- history and evolution
  - of PLs
  - of software engineering
  - of compiler engineering
- architecture of compilers
  - architectural styles
  - examples of compiler architectures
- advanced compiler architectures
  - frameworks

Evolution of PLs

generations:

- 1st: machine languages
  - specific to processor (family)
  - absolute addresses
- 2nd: assembly languages
  - specific to processor (family)
  - symbolic addresses / macros
  - simple transformation to machine code
- 3rd: general-purpose PLs
  - not specific processor (family)
  - domain-independent
  - complicated transformation to machine code
- 4th: problem-specific VHLL / DSLs

Evolution of Compiler Engineering

- craft: first compilers (Fortran54/Algol58/Cobol59)
  - scanning and parsing not well understood (BNF:1959; Chomsky:1950)
  - 18 person-years to develop the FORTRAN compiler

"The entire project was carried out by a loose cooperation between autonomous, separate groups . . . each group **invented** and programmed the necessary techniques for doing its assigned job." (Fortran)

"By today's standards, of course, most of the items which were felt to be difficult to compiler are trivial, but they were not that easy in those days." (Cobol)

"The number of such cases increased exponentially with the number subscripts; this was a prime factor in our decision to limit them to three" (Fortran)

"the people in my office responsible for implementing the compiler had what they considered to be an excellent algorithm for handling 3 subscripts but it did not work beyond 3." (Cobol)

"By the summer of 1956 what appeared to be the imminent completion of the project started us worrying (for perhaps the first time) about documentation." (Fortan) ;-)

Evolution of Compiler Engineering

- production:
  - more and more PLs and implementations emerge. . .
  - techniques for scanning and parsing emerge
  - BNF used for Cobol60
  - BNF = context-free grammar around 1963

- scientific foundations:
  - "Finite Automata and Their Decision Problem", Rabin and Scott, 1959
  - "On the Translation of Languages from Left to Right", Knuth, 1965
  - "Formal Languages and Their Relation to Automata", Hopcroft and Ullman, 1969
  - "Principles of Compiler Design", Aho and Ullman, 1978
  - type theory
  - . . .

  "It takes a good twenty years from the time that work starts on a theory until it provides serious assistance to routine practice." [SG96]

Evolution of Compiler Engineering

- professional engineering:
  - scanner (1968)
  - parser (yacc:1975)
  - compiler generators / toolkits
  - optimization frameworks
  - PL concepts
  - . . .

  "the compiler writer might adopt the organization of a known compiler for a similar language and implement the corresponding components, using component-**generation** tools or implementing them by hand. It is relatively rare that a completely new compiler organization [= **architecture**] is required." [ASU86]

Evolution of Compiler Engineering

- professional engineering (cont'd):
  - compiler vendors (Greenhills, PGI, . . .)
    "STMicroelectronics (NYSE: STM) today announced an agreement to acquire Portland Group Inc (PGI), a vendor of compilers and software development tools to the high-performance parallel computing market. [. . .]
    Combining ST's advanced ST100 core designs with PGI's unique compiler expertise provides the best possible performance for the target applications in mobile phones, wideband network access, data storage, automotive and multimedia."
    (`http://www.pgroup.com/stpgi_announce.htm`)
  - component vendors (EDG)
    "Our customers are companies that want to develop a compiler or a source-analysis tool, companies such as computer manufacturers, chip manufacturers, and software tool developers. They license front ends from us, combine them with software of their own (e.g., a code generator), and sell the resulting products."
    (`http://www.edg.com/customers.html`)

Architecture

Who cares about it?

- coop student?
- (sub-module) programmer?
- chief architect?
- management?

"competitive success flows to the company that manages to establish proprietary architectural control over a broad, fast-moving, competitive space"

"We call [a collection] of standards and rules an 'architecture' "

[MF93]

Architectural Styles

- set of components
  (e.g., data repository, process, procedure)
- topological layout of components
- set of semantic constraints
  (e.g., data repository is read only)
- set of connectors that mediate communication/coordination between components
  (e.g., DB access, RPC/socket, subroutine call)

(style = class of concrete architectures)

building analogy: Gothic / Greek Revival architectural style

[BCK98]

Compiler Architectures

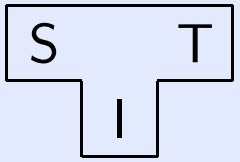
- AhoUllman77 (original dragon book)
- AhoSethiUllman86 (current version of dragon book)
- Wolfe96
- Kienle98
- ShawGarlan96, page 83ff
- Plödereder94
- SUIF

Compiler Architectures

- what are the commonalities?
  - box and arrow diagrams
  - composition in phases
- what are the differences?
  - level of detail
  - emphasis/view on (part of) the architecture
  - intended audience (compiler user/builder)
- can we expect the same characteristics for a different domain?

Other Compiler Architectures

- compiler environment
- T-diagrams [ASU86]:



Advanced Compiler Architectures

- integrated developer environments (IDEs)
  - Montana (now IBM Visual Age C++) [Kar98] [Mar99]
    repository: CodeStore
- compiler frameworks
  - Montana
  - SUIF compiler system (Monica Lam, Stanford University)
- virtual machines
  - JITs
  - HotSpot

Compiler Frameworks

what is a framework?

- software architecture + implementation + hooks
- provides generic capabilities in some domain
- custom application specific code added by following hooks (framework use cases)

design issues:

- what is common (i.e., constrained)?
  what is variable?
  $\longrightarrow$ product lines
- how do you balance the architecture between the two?

(H. James Hoover and Daqing Hou, University of Alberta)

Compiler Frameworks

- Montana [Kar98]
  - objective: open tool environment
  - extensions
    - supplied in DLLs
    - automatically loaded at startup as part of config file processing
  - three extension mechanisms:
    1. incorporation extension – adding of phases (e.g., style checker)
    2. event-based notification (e.g., removal of declaration)
    3. dependency graph extensions (sophisticated make)
  - computation on-demand paradigm (AST construction on-the-fly)
  - extension mechanisms: templates, interfaces, DLLs

"we do not use inheritance as an extension to implement an extension framework."

"There is a **tension** in the design of the extension mechanism between providing convenient hooks for tool writers and not compromising the compilation time."

Compiler Frameworks

- SUIF
  - objective: share research results in compilers (reuse)
  - support at three levels:
    1. compose new compiler with existing components
       - scripting (e.g., to specify pass ordering)
    2. develop new passes
       - extensible pass framework
       - data-flow analysis framework
    3. develop new IR
       - extensible IR
       - *Hoof* specification language
       - persistence, printing, cloning, introspection, . . .
  - extension mechanisms: subclassing, templates, DSLs, DLLs (modules)

Problems with Frameworks

- frameworks are typically large and complex
- lots of invisible rules and conventions
- hard to specialize when no previous applications or in-house experience
- specialization interface not defined and documented
- no established techniques to define and document specialization
- interface separation of trivial and non-trivial parts

(Kai Koskimies, Tampere University of Technology, Finland)

References

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley, 1986.

[BCK98] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, 1998.

[Kar98] Michael Karasick. The architecture of Montana: An open and extensible programming environment with an incremental C++ compiler. *ACM SIGSOFT sixth international symposium on Foundations of software engineering*, pages 131–142, November 1998.

[Mar99] Johannes Martin. Leveraging IBM VisualAge for C++ for reverese engineering tasks. *CASCON '99*, pages 83–95, November 1999.

[MF93] Charles R. Morris and Charles H. Ferguson. How architecture wins technology wars. *Harvard Business Review*, 71(2):86–96, March-April 1993.

[SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

colophon:

set in LaTeX+ `pdfscreen` (on Debian GNU/Linux)