Lecture 28:
Software Architecture

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 1999

Today's Lecture

- Software Architecture
  - Specification
  - Examples
    - Chemical Abstract Machine
    - C2

Architecture Specification

- Design Elements
- Form
  - Relationships among elements
- Rationale
  - Justification or arguments for choices of elements and form
- Constraints
  - Properties and weights

Design Elements

- Processors/Functions/Transformers/Actors
- Data/Information
- Connectors/Glue

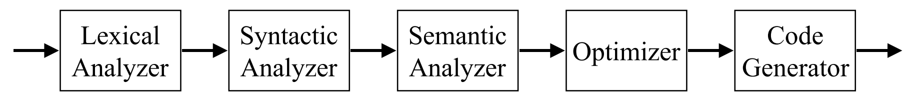
- A Useful Metaphor

  Consider Polo, Water Polo, and Soccer: Similar in processors and data, but differ in connectors

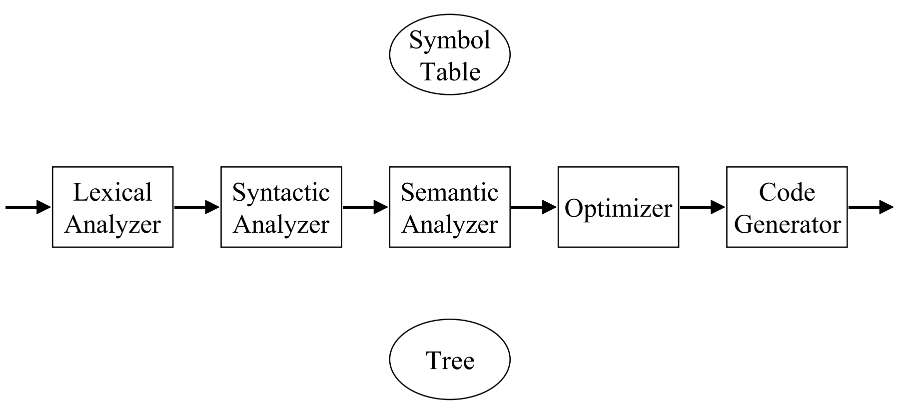
Form

- Approaches
  - Formal
  - Prose
  - Picture
  - Prototype
  - Analogy
  - Contrast
- Abstraction
  - Complete
  - Modular
  - “White lie”
- Audience/Purpose
  - Adoption
  - Comparison
  - Building
  - Analysis
  - Education

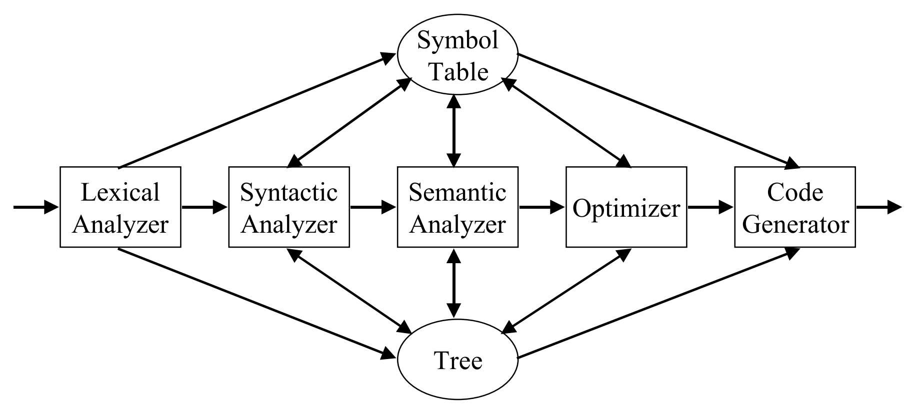
Example: Language Compiler



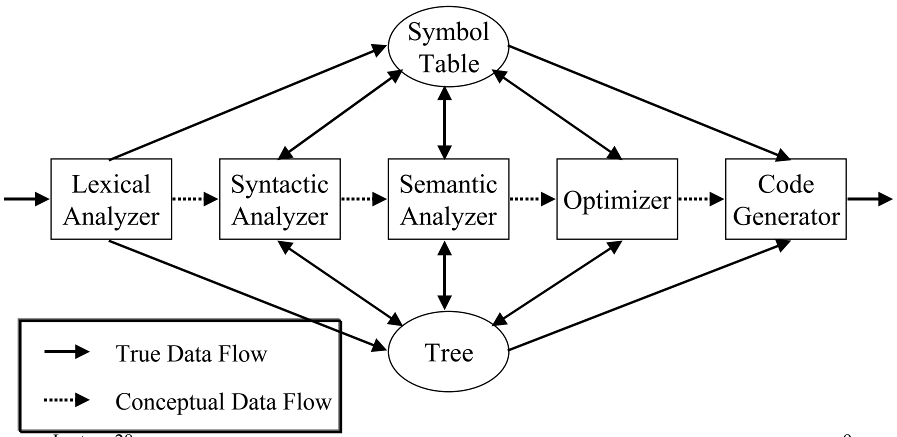
Example: Language Compiler



Example: Language Compiler



Example: Language Compiler



Example: Language Compiler

processing element *Lexer*
imports data element *Characters*
exports data element *Tokens*

processing element *Parser*
imports data element *Tokens*
exports data element *Phrases*

connecting element *Pipe*
connects *Lexer* to *Parser*

Formal Specification

- Structure (Form)
  - How is the system organized?
- Function
  - What does the system compute?
- Compatibility
  - When is a system properly composed?
- Specializations
  - How are generic systems constrained?

Chemical Abstract Machine

- A Convenient Metaphor
  - Components are like molecules
  - Systems are like solutions
  - Molecules interact (i.e., react)
  - Rules govern interaction
  - State of system is like state of solution
- Mathematical Foundation
  - Term rewriting

Term Rewriting

- Syntax for Expressions (Terms)
- Initial Expression
- Rewrite Rules

Basic CHAM Definitions

- Molecules m_1, m_2, …
- Solutions S_1, S_2, …
  – Finite multiset of molecules
- Transformation Rules T_1, T_2, …
  – General laws and specific rules
  – Heating, cooling, and reaction rules
- Transformation Relation $S \rightarrow S'$

General Laws

- Reaction Law
  $M_1,M_2,\ldots,M_k \rightarrow M_1',M_2',\ldots,M_n'$
- Chemical Law
  $S \rightarrow S'$ equivalent to $S+S'' \rightarrow S'+S''$
- …

Sequential Multi-Phase Compiler

Syntax

M ::= P | C | M ◆ M
P ::= text | lexer | parser | semantor
| optimizer | generator
D ::= char | tok | phr | cophr | obj
C ::= i(D) | o(D)

Sequential Multi-Phase Compiler

Initial Solution

S1 = text ◆ o(char),

i(char) ◆ o(tok) ◆ lexer,

i(tok) ◆ o(phr) ◆ parser,

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

- Transformation Rules

$T_1 \equiv$ text ◆ o(char) $\rightarrow$ o(char) ◆ text

$T_2 \equiv$ o(d) ◆ m_1,i(d) ◆ m_2 $\rightarrow$ m_1 ◆ o(d),m_2 ◆ i(d)

$T_3 \equiv$ o(obj) ◆ generator ◆ i(cophr) $\rightarrow$

i(char) ◆ o(tok) ◆ lexer,

i(tok) ◆ o(phr) ◆ parser,

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Identify TR 1 match

S1 = *text* ◆ *o(char)*,

i(char) ◆ o(tok) ◆ lexer,

i(tok) ◆ o(phr) ◆ parser,

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 1, Identify TR 2 Match

S1 = **o(char) ◆ text,**

i(char) ◆ o(tok) ◆ lexer,

i(tok) ◆ o(phr) ◆ parser,

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 2, Identify next TR 2 Match

S1 = text ◆ o(char),

o(tok) ◆ lexer ◆ i(char),

i(tok) ◆ o(phr) ◆ parser,

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 2, Identify next TR 2 Match

S1 = text ◆ o(char),

lexer ◆ i(char) ◆ o(tok),

o(phr) ◆ parser ◆ i(tok),

i(phr) ◆ o(cophr) ◆ semantor,

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 2, Identify next TR 2 Match

S1 = text ◆ o(char),

lexer ◆ i(char) ◆ o(tok),

parser ◆ i(tok) ◆ o(phr),

o(cophr) ◆ semantor ◆ i(phr),

i(cophr) ◆ o(cophr) ◆ optimizer,

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 2, Identify next TR 2 Match

S1 = text ◆ o(char),

lexer ◆ i(char) ◆ o(tok),

parser ◆ i(tok) ◆ o(phr),

semantor ◆ i(phr) ◆ o(cophr),

o(cophr) ◆ optimizer ◆ i(cophr),

i(cophr) ◆ o(obj) ◆ generator

Sequential Multi-Phase Compiler

Apply TR 2, Identify TR 3 Match

S1 = text ◆ o(char),

lexer ◆ i(char) ◆ o(tok),

parser ◆ i(tok) ◆ o(phr),

semantor ◆ i(phr) ◆ o(cophr),

optimizer ◆ i(cophr) ◆ o(cophr),

o(obj) ◆ *generator* ◆ *i(cophr)*

Sequential Multi-Phase Compiler

Final Step, Identify TR 1 Match, TR 2 below

S1 = *text* ◆ *o(char),*

lexer ◆ i(char) ◆ o(tok),

parser ◆ i(tok) ◆ o(phr),

semantor ◆ i(phr) ◆ o(cophr),

optimizer ◆ i(cophr) ◆ o(cophr),

i(char) ◆ *o(tok)* ◆ *lexer*, …

(Plain lines are ignored in next iteration)

Formal Specification

- Structure (Form)

  How is the system organized?
- Function

  What does the system compute?
- Compatibility

  When is a system properly composed?
- Specializations

  How are generic systems constrained?

Benefit: Analysis

- Consistency of Style Constraints
- Satisfaction of Style by Architecture
- Satisfaction of Requirements by Architecture and of Architecture by Implementation
- Consistency of Structure and of Behavior
- Effects of Changes

Example: C2 Architectural Style

- Evolved from the Chiron User-Interface Development System
- Components and Connectors
  - each potentially with their own thread of control
- Constraint
  - Components can “see” “up” an architecture not “down”
- Benefit: Subsystems are Substitutable
- Research being conducted on C2 today...