

CSci 780 Advanced Software Engineering

An Introduction to Software Architecture

David Garlan and Mary Shaw

10/10/2003

1

Outline

- Architecture
 - Problem
 - Definition
 - Uses
- *A Taxonomy of Styles*
- Case Studies: KWIC, Compiler
- "Future" Directions
- Summary
- Discussion

10/10/2003

2

Problem

- As systems grow in size, low-level algorithm and data structure design becomes less relevant
 - Throughput, communication protocols, system evolution, component compatibility, scalability
- The global *architecture* is then the key problem
 - Structural issues
 - Gross organization
 - Global control structure
- Paper provides an informal taxonomy of common styles, and some case studies

10/10/2003

3

Definitions of Architecture

- Perry & Wolf:
elements (what) + form (how) + rationale (why)
- Garlan & Shaw:
 - Components: the subsystems
 - Interfaces: the external view of a module
 - Connections: module communication
 - Topology: organization of components and conn's
 - Constraints: limitations on comp's, conn's, change
- Kruchten:
 - design and impl. of the high-level software structure
 - abstraction, (de)composition, style, and *aesthetics*

10/10/2003

4

Why Architecture?

- Helps to focus on large-scale system design
- Separation of design concerns
- Can identify good domain-specific architectures
- Design abstraction
- Have a common language
 - Managerial basis for cost estimation & process mgmt
 - Reuse
 - Consistency and dependency analysis
 - Technical basis for design

10/10/2003

5

What Are They Used For?

- Representation: aid communication
- Design process: ease system decomposition
- Analysis (static and dynamic): avoid errors
- Evolution (specification-time, execution-time):
change modules, develop system families
- Refinement: transform diagram into code
- Traceability: ensure requirements fulfilled,
ensure valid relationships among views
- Simulation/Executability: check properties

10/10/2003

6

Taxonomy of Architectural Styles

Architectural Types and Styles

- Type: the domain of the software system
 - Control system, database system, web browser, etc
- Style: A abstract set of modules and connections which restrict architecture
 - Pipes and filters, client-server, layered, etc.

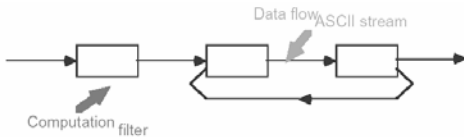
Styles

- What is the topology (configuration)?
- What is the underlying computational model?
- What are the essential invariants?
- What are some common examples?
- What are the advantages and disadvantages?
- What are some common specializations?

The Template

- Modules: ???
- Connections: ???
- Topology: ???
- Examples and Specializations: ???
- Pros: ???
- Cons: ???

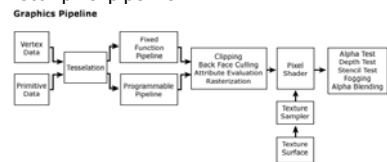
Pipes and Filters



- M: independent data-transforming filters
- C: data from one module to another
- T: connectors define a data-flow graph

Pipes and Filters (cont)

- Examples
 - Compilers
lexer → parser → semantic analysis → code gen.
 - Unix pipes (pipeline specialization)
`data.pl | grep -i fdep | grep -v FDep | less`
 - DirectX pixel pipeline



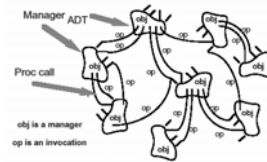
Pipes and Filters (cont)

- **Pros:** Filters don't know about architecture, easy to modify, simple, support concurrency, throughput/deadlock analysis is possible
- **Cons:** Not interactive, synchronization problematic, performance is limited

10/10/2003

13

Object-Oriented



- **M:** Domain models with hidden state & interfaces
- **C:** Methods are invoked on the objects, maintaining invariants. Shared memory common
- **T:** Inherited interfaces. Arbitrary interactions

10/10/2003

14

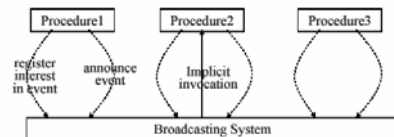
Object-Oriented (cont)

- **Examples**
 - Employee management software
 - Object brokers (CORBA)
 - Abstract data types
- **Pros:** Implementation hiding, "natural" decomposition according to domain
- **Cons:** Hidden dependencies, objects must know identity of other objects

10/10/2003

15

Implicit Invocation (Event-Based)



- **M:** A module *registers* with another module to receive some callback event
- **C:** When something happens (an *event*), interested modules are notified
- **T:** Modules connected by a shared bus

10/10/2003

16

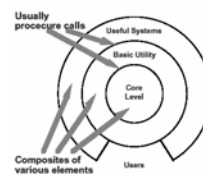
Implicit Invocation (Event-Based) (cont)

- **Examples**
 - GUIs
 - Some DBMSs
 - Some IDEs
- **Pros:** Decoupled (announcers don't need to know listeners), easy reuse, easy evolution
- **Cons:** Less component control (e.g. ordering of computations), reasoning hard

10/10/2003

17

Layered



- **M:** Layers provide services. Inner layers hidden
- **C:** Usually procedure calls. Layers build a "virtual machine" out of the previous
- **T:** Nested

10/10/2003

18

Layered (cont)

- Examples
 - Operating systems
 - TCP stack
 - Portability layers

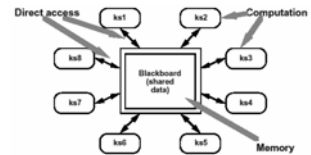


- Pros: ~~data~~ increasing abstraction during design, simple interactions, reuse
- Cons: ~~data~~ performance, layers difficult to design

10/10/2003

19

Blackboard (Shared Repository)



- M: A shared controller-repository module, plus multiple computational agents
- C: Agents monitor state changes and react. Or they simply make controlled changes.
- T: Clients surround repository

10/10/2003

20

Blackboard (Shared Repository) (cont)

- Examples
 - Version control systems
 - Expert systems
 - Programming environments

- Pros: ~~data~~ consistency easy, interactions encapsulated in either blackboard or agents
- Cons: ~~data~~ data-oriented, performance

10/10/2003

21

And Many More...

- Process control
- Peer-to-peer
- Client-server
- State transition systems
- Table-driven interpreters

- Domain-specific architectures

10/10/2003

22

Reality: Heterogeneous Architectures

- Most systems are not purely one style
- Hierarchy: large blocks use pipeline, internally each block uses another style
- Single component uses mixture of connectors: one component repository, and pipes
- Completely elaborate one level in another style

10/10/2003

23

Case Studies

10/10/2003

24

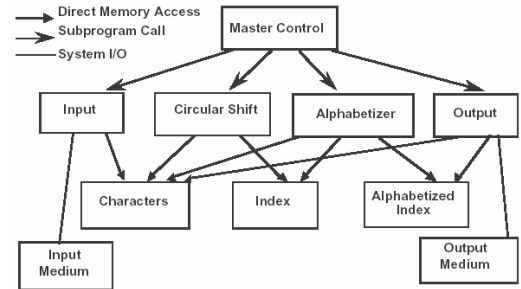
KWIC (Surprise!)

- Analyze system with respect to 4 architectures
 - Main program/subroutine with shared data
 - Abstract data types
 - Implicit invocation
 - Pipeline

10/10/2003

25

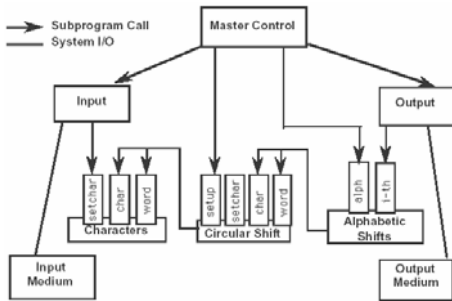
Shared Data Architecture



10/10/2003

26

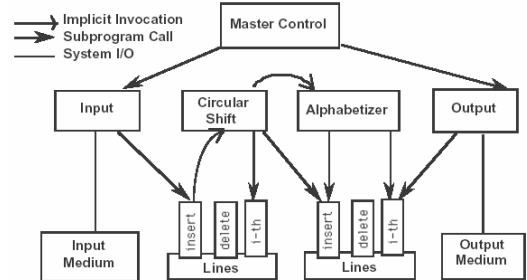
Data Abstraction Architecture



10/10/2003

27

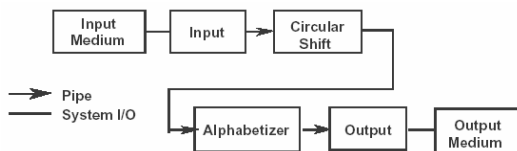
Implicit Invocation Architecture



10/10/2003

28

Pipe and Filter Architecture



10/10/2003

29

Comparison of Solutions

	Shared Memory	ADT	Events	Dataflow
Change in Algorithm	-	-	+	+
Change in Data Repn	-	+	-	-
Change in Function	+	-	+	+
Performance	+	+	-	-
Reuse	-	+	-	+

10/10/2003

30

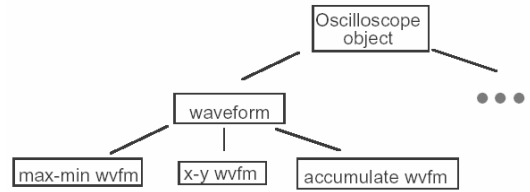
Tektronix Oscilloscope

- No architecture meant re-design for new models
- Performance problems
- Little reuse
- Attempted several styles:
 - Object-oriented
 - Layered
 - Pipe-and-filter
 - Specialized pipe-and-filter

10/10/2003

31

Object-Oriented Architecture

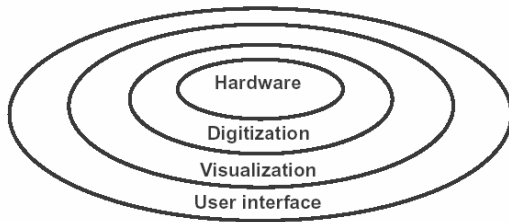


Several types, but not useful in assembling design

10/10/2003

32

Layered Architecture

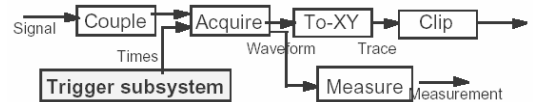


Intuitive, but need to affect all layers

10/10/2003

33

Pipe-And-Filter Architecture

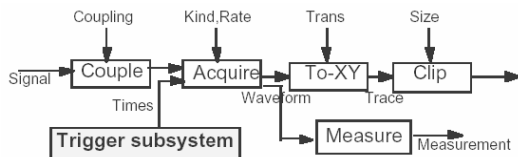


Good fit, but needed control interface

10/10/2003

34

Modified Pipe-And-Filter Architecture



Slow filters can ignore data; several types of pipes

10/10/2003

35

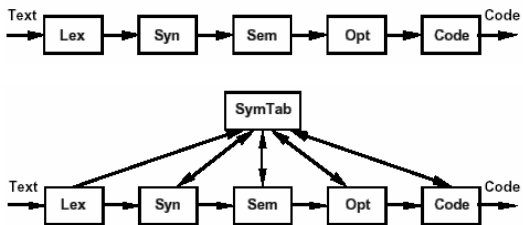
Compilers

- Compiler architecture evolved:
 - First: Pipeline (actually batch sequential)
 - Then: blackboard-like (symbol table, then parse tree)
 - Today: blackboard

10/10/2003

36

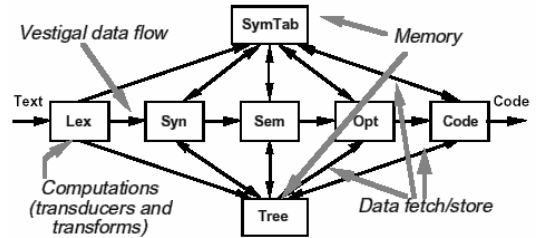
Original Pipeline Architecture



10/10/2003

37

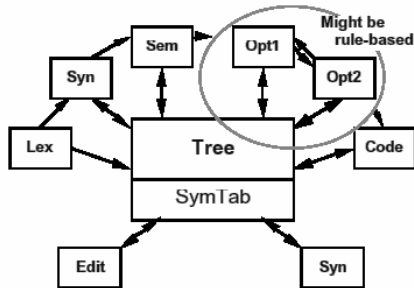
Blackboard-Like Architecture



10/10/2003

38

Blackboard (?) Architecture



10/10/2003

39

Conclusion

10/10/2003

40

Future

- Better taxonomies
- Formal models
- Better understanding of primitive semantic entities
- Notations
- Tools and environments to develop designs
- Tools to extract designs
- Better understanding of role of architectures in the life-cycle

10/10/2003

41

More Recent work

- Architectural description languages (ADLs)
 - Models, tools, notations to describe components and connectors
 - ACME, Aesop, C2, Darwin, Rapide, Weaves, Wright
- Middleware
 - Layer between apps and OS supporting service location, location/access transparency, object compatibility
 - CORBA, COM/DCOM

10/10/2003

42

More Recent Work (cont)

- Frameworks
 - Reusable design, with abstract classes for “plugging” in concrete implementations
 - MFC (e.g. SDI/MDI), domain-specific stuff
- Design patterns
 - Capture, name, catalog common design solutions
 - Singleton, observer, visitor

10/10/2003

43

Summary

- Architectural Design is about determining the modules, interfaces, and communication
- We want to aim for a decomposition which is simple and hides complexity in recursively defined modules
- We can benefit from styles which others have found to be useful

10/10/2003

44

Discussion

- Architecture = Framework = Pattern?
- How do we choose an architecture?
- How do we know an architecture will succeed?
- How much design should we do?
- What is the role of requirements in architectural design?

10/10/2003

45