CSE 503: Software Engineering

Connecting Architecture to Implementation

Jonathan Aldrich
University of Washington
Department of Computer Science and Engineering
Winter 2002

Design → Implementation

- Architecture captures system design
- But does it match the implementation?
  - What if the program evolves?
  - May leave out important details
  - May be misleading
- Must keep architecture consistent if we want it to continue to be useful!

One Approach: ADL Tools

- Rapide: simulates architecture with code
  - Flags error if event sequence doesn’t match
- C2: run time library support
- UniCon: code generation from architecture

- Fundamental issue:
  - No guarantee that architecture is accurate picture of code

Another Approach: Modules

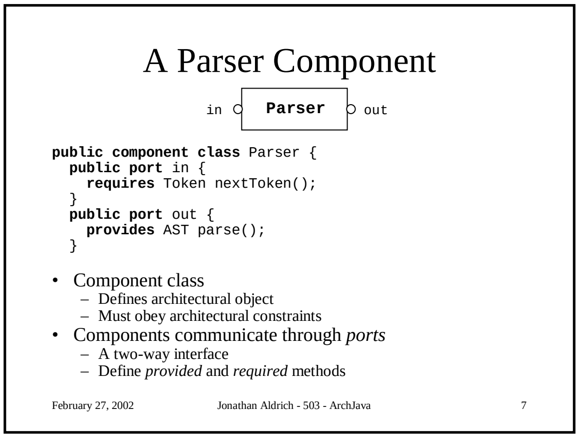
- Basic module systems
  - File system, packages, libraries
- Advanced module systems
  - ML, Units, Knit, Jiazzi
- Strengths
  - Encapsulate components
  - Linking shows connections
  - Very flexible

Module Weaknesses

- Modules show only static structure
  - Interconnections between component instances
  - Dynamic changes to structure
- Modules don’t show all control & data flow
  - Especially with objects (or first-class functions)

ArchJava

- *Specifies* architecture within Java code
  - Similar to other ADLs
- *Verifies* that control flow conforms to arch.
  - Our key technical contribution
- Is *flexible*
  - Supports dynamically changing architectures
  - Allows common implementation techniques
- May aid in *software evolution tasks*
  - Two case studies on 12,000-line programs

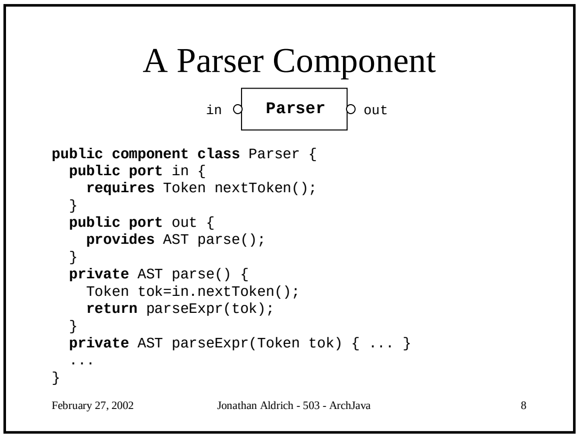


A Parser Component

in
Parser
out

```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
```

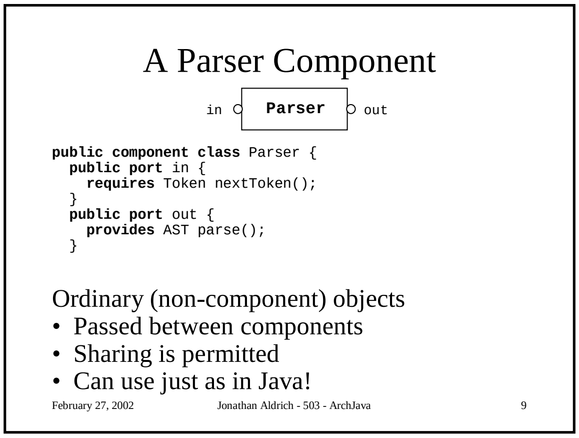
- Component class
  - Defines architectural object
  - Must obey architectural constraints
- Components communicate through *ports*
  - A two-way interface
  - Define *provided* and *required* methods



A Parser Component

in
Parser
out

```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
  private AST parse() {
    Token tok=in.nextToken();
    return parseExpr(tok);
  }
  private AST parseExpr(Token tok) { ... }
  ...
}
```



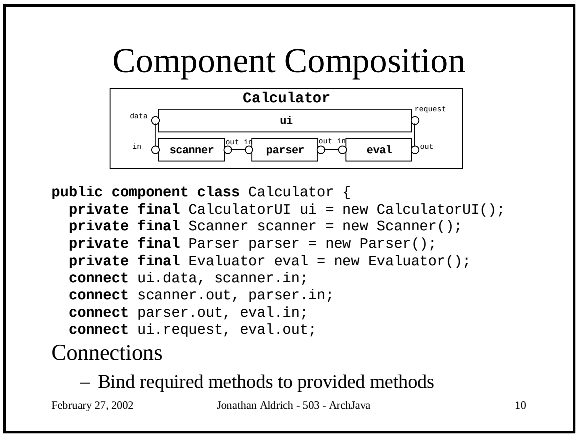
A Parser Component

in
Parser
out

```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
```

Ordinary (non-component) objects

- Passed between components
- Sharing is permitted
- Can use just as in Java!



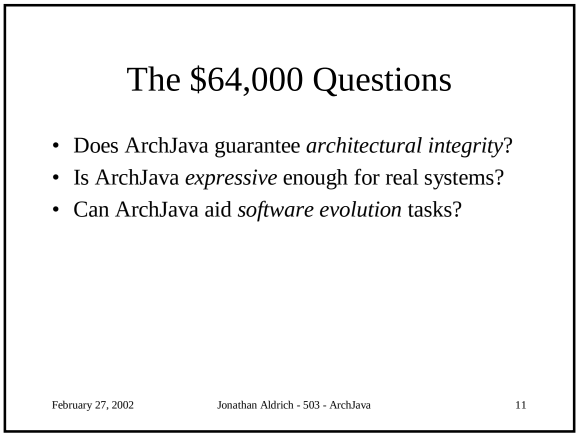
Component Composition

Calculator
data
ui
request
in
scanner
out in
parser
out in
eval
out

```
public component class Calculator {
  private final CalculatorUI ui = new CalculatorUI();
  private final Scanner scanner = new Scanner();
  private final Parser parser = new Parser();
  private final Evaluator eval = new Evaluator();
  connect ui.data, scanner.in;
  connect scanner.out, parser.in;
  connect parser.out, eval.in;
  connect ui.request, eval.out;
```

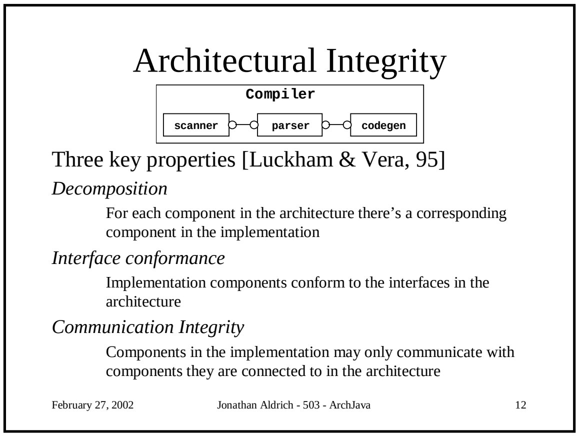
Connections

- Bind required methods to provided methods



The $64,000 Questions

- Does ArchJava guarantee *architectural integrity*?
- Is ArchJava *expressive* enough for real systems?
- Can ArchJava aid *software evolution* tasks?



Architectural Integrity

Compiler
scanner
parser
codegen

Three key properties [Luckham & Vera, 95]

Decomposition

For each component in the architecture there’s a corresponding component in the implementation

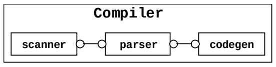
Interface conformance

Implementation components conform to the interfaces in the architecture

Communication Integrity

Components in the implementation may only communicate with components they are connected to in the architecture

The ArchJava Approach



Put the architecture into the implementation

Decomposition: ***true by definition!***

For each component in the architecture there's a corresponding component in the implementation

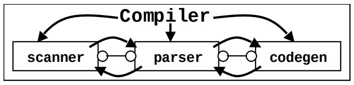
Interface conformance: ***just typechecking!***

Implementation components conform to the interfaces in the architecture

Communication Integrity: ***still hard!***

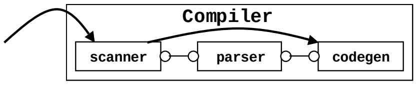
Components in the implementation may only communicate with components they are connected to in the architecture

Communication Integrity



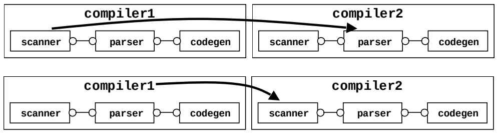
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its subcomponents

Communication Integrity



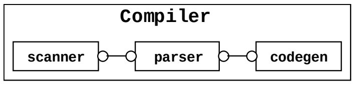
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its subcomponents
- Architecture forbids
  - External calls to subcomponents
  - Calls between unconnected subcomponents

Communication Integrity



- Other integrity violations

Comm. Integrity in ArchJava



- No method calls permitted from one component to another *except*
  - From a parent to its nested subcomponents
  - Through connections in the architecture
- Supports reasoning about control flow
  - Current work: Data flow
    - Shared object references

Enforcing Architectural Integrity

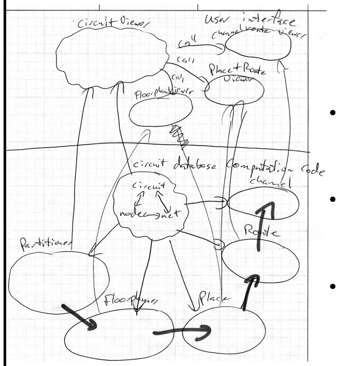
- Q: How does ArchJava prohibit illegal component method calls?
- A: Through its type system
  - Component classes follow special type rules
  - Advantages:
    - Consistency: rules checked on every compile
    - Can prove soundness
  - Drawbacks? Alternatives?

Enforcing Architectural Integrity

- Integrity for direct method calls:
  - All calls are to `this` or to a subcomponent
- Components can only get typed references to their subcomponents
  - No component types in port interfaces
  - No fields of component type in objects
  - Casts to component type check the parent

The $64,000 Questions

- Does ArchJava guarantee *architectural integrity*?
  - ***Yes!*** *(for control flow)*
- Is ArchJava *expressive* enough for real systems?
  - Two case studies
    - 12,000 lines of Java code each
    - Asked developer to draw architecture
    - Tried to specify architecture in ArchJava
- Can ArchJava aid *software evolution* tasks?



Aphyds Architecture

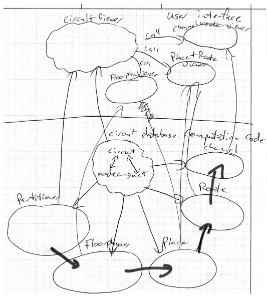
- UI above
  - Main window
  - 3 secondary windows
- Circuit DB below
  - Central DB
  - 5 comp. Modules
- Arrows
  - Data & control flow

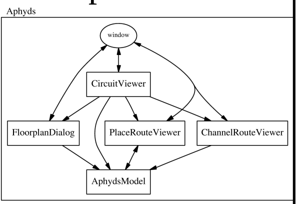
Aphyds Architecture

- Informal drawing
  - Common in practice!
- Leaves out details
  - What's inside the components, connections?
  - `CircuitViewer` has internal structure
- Some surprises
  - Missing paths
  - Component lifetimes

Hypothesis: Developers have a conceptual model of their architecture that is mostly accurate, but this model may be a simplification of reality, and it is often not explicit in the code.

Architectural Comparison





Automatically Generated Architectural Visualization

Advantages of ArchJava

- Complete
  - Can "zoom in" on details
- Consistency checking
  - Original architecture had minor flaws
- Evolves with program
- Low cost
  - 30 hours, or 2.5 hours/KLOC
  - Includes substantial refactoring
  - 12.1 KLOC => 12.6 KLOC

Hypothesis: Applications can be translated into ArchJava without excessive effort or code bloat.

The $64,000 Questions

- Does ArchJava guarantee *architectural integrity*?
  - ***Yes!*** *(for control flow)*
- Is ArchJava *expressive* enough for real systems?
  - ***Yes!*** (*for one small but realistic system*)
- Can ArchJava aid *software evolution* tasks?
  - Three experiments
    - Understanding Aphyds communication
    - Refactoring Aphdys
    - Reparing a defect

Program Understanding

> *Communication between the main structures is awkward, especially the change propagation messages*
>
> – Aphyds developer

- Inter-component communication analysis
  - Message purpose, callers, callees, triggers
  - Goal: refactor program source
- Difficult in original program
  - Confusing method names
  - Transitive method dependencies
  - Methods had multiple purposes
    - e.g. assign data & refresh screen
  - Hard to tell what methods called by each object

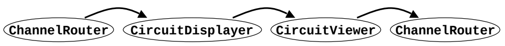
Program Understanding

- Communication analysis easier in ArchJava
  - Provided ***and*** required interfaces
  - Connections show relationships
  - Ports show relevant methods
  - Ports group related methods
- Several refactoring opportunities
  - Window refresh, data invalidation
    - *Developer's problem areas!*

Hypothesis: Expressing software architecture in ArchJava highlights refactoring opportunities by making communication protocols explicit.

Implicit Refactoring

- Law of Demeter [Lieberherr et. al.]
  - Only talk with your immediate neighbors
  - Reduces system coupling
- Example violation

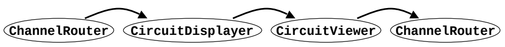


```
getDisplayer().getViewer().ChannelRouterMenuItem.setEnabled(b);
```

- Problems
  - Depends on every link in chain
  - Programs are fragile, change is difficult

Implicit Refactoring

- Communication Integrity ≈ Law of Demeter
  - Components only talk with connected components
- Example violation



```
getDisplayer().getViewer().ChannelRouterMenuItem.setEnabled(b);
```

- Illegal in ArchJava! Instead…

```
port window {
  requires void enableMenuItem(int menu, boolean enabled);
  ... }
window.enableMenuItem(CHANNEL_ROUTE, b);
```

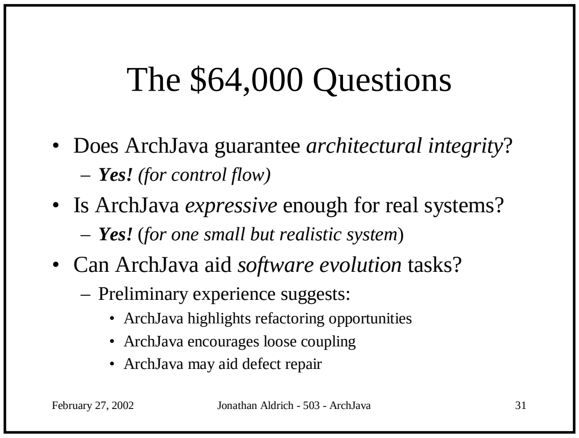
Hypothesis: Enforcing communication integrity helps to reduce system coupling

Defect Repair

- Fix same Aphyds bug
  - First in ArchJava, then Java
- ArchJava required more coding
  - Had to add new ports & connections
- Java took longer
  - Difficult to find object involved in fix
  - Even though I'd already fixed the bug in ArchJava!

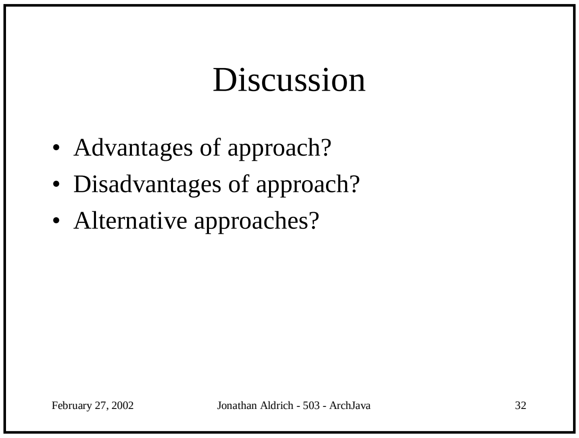
```
getDisplayer().placeroutedialog1.placeRouteDisplayer1
        .getCircuitGlobalRouter().doGlobalRouting();
```

Hypothesis: An explicit software architecture makes it easier to identify and evolve the components involved in a change.



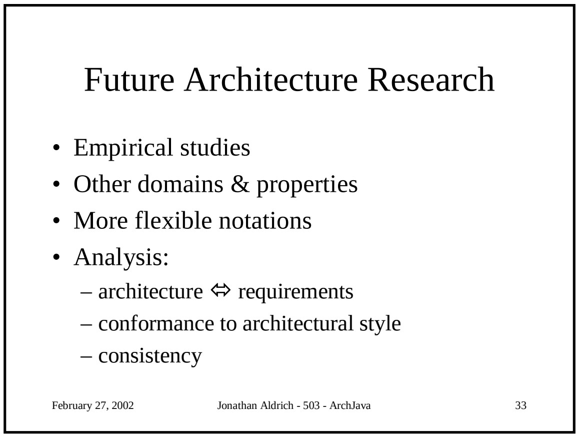
The $64,000 Questions

- Does ArchJava guarantee *architectural integrity*?
  - ***Yes!*** *(for control flow)*
- Is ArchJava *expressive* enough for real systems?
  - ***Yes!*** (*for one small but realistic system*)
- Can ArchJava aid *software evolution* tasks?
  - Preliminary experience suggests:
    - ArchJava highlights refactoring opportunities
    - ArchJava encourages loose coupling
    - ArchJava may aid defect repair



Discussion

- Advantages of approach?
- Disadvantages of approach?
- Alternative approaches?



Future Architecture Research

- Empirical studies
- Other domains & properties
- More flexible notations
- Analysis:
  - architecture ⇔ requirements
  - conformance to architectural style
  - consistency