Case Studies for Evaluating Programming Languages

Jonathan Aldrich

17-396/17-696/17-960:
Language Design and Prototyping

Carnegie Mellon University

Thought Question

- What can we learn from trying out a language? Can this be done scientifically?

Case Study

- A research method that deeply examines a particular situation to gain understanding
- Used for
  - Generating hypotheses
  - Answering *how* and *why* questions
  - Evaluating hypotheses in a real-world setting
- Limitation: no statistical generalization
  - But does support *analytical generalization*

Why Use a Case Study?

- You want to gain a deep understanding of your language in a real world context
  - Does it have the effect you expect?
  - What surprising effects does it have?
  - How and why does it have those effects?
  - How does it stand up to the complexities of the real world?

Data Gathering

- Case studies typically a mixed method
  - Count things – how big, how many, how long?
  - Observe things – the process, artifact…
  - Use triangulation: multiple sources and kinds of evidence that point to the same facts
- Case studies exist in a context
  - Beneficial for external validity – realistic
    - Results likely meaningful real world
  - Challenge for internal validity – hard to control
    - Hard to be sure in identifying causes

Case Studies with PLs

- Example case: writing a program in a new PL
- Data that could be gathered
  - How long did it take?
  - How many lines of code?
  - How were particular new constructs used? What were the benefits/limitations of those constructs in context?
  - Did the PL affect the design? Help find bugs?
- Compare to the same program in another PL

Discussion: Is This Science?

What Makes It Science?
(vs. an experience report or illustrative example)

- Research questions identified
- Data is collected consistently, according to a plan
- Inferences connect data to research questions
- Explores, explains, describes, or (causally) analyzes a phenomenon
- Systematically addresses threats to validity

[adapted from Easterbrook *et al.*]

How to Design a Case Study (1)

- Identify research questions precisely
  - Draw on relevant theory
- Identify hypotheses
  - Sometimes called "propositions" for case studies
  - Alternatively, your goal may be to form hypotheses
    - Exploratory studies – still need purpose (what kind of hypotheses?) and criteria for success
- Identify the unit of analysis
  - Precisely define the case – what is the study's scope?

[adapted from Easterbrook *et al.*]

How to Design a Case Study (2)

- Data collection
  - What information will you collect? How will you do it?
  - How will you decide what to include/exclude?
- Linking logic
  - Logic that relates data to hypotheses
  - Example: pattern matching
    - Describe several patterns, e.g. that represent alternative explanations
    - Compare case study to patterns: which one fits best?
- Interpretation criteria
  - How will you analyze the data and interpret findings?

[adapted from Easterbrook *et al.*]

Analytical Generalization

- Compare qualitative findings to a theory
  - Does the data support or refute the theory?
    - Note: in the case of partial support, may motivate possible changes to the theory
  - Is one theory better supported than another?
- Empirical induction
  - Evidence builds when several case studies all support a theory (compared to rival theories)
- Power comes from detail
  - Looks at underlying mechanism; tries to explain
  - Many pieces of data come together to support (or refute) a theory
- Compare: statistical generalization
  - Sample from, generalize to a population

[adapted from Easterbrook *et al.*]

Case Study Replication

- Replicating case studies can
  - Add confidence to conclusions
  - Help broaden a theory and its support
- Selection guided by theory
  - Predict similar results
  - Predict contrasting results but for predictable reasons
- *Not* random sampling from a pool!

[adapted from Easterbrook *et al.*]

Case Study Analysis Principles

- Rely on theory
  - Tells you what data is relevant and how to test it
  - Alternatively, derive possible theories from data (in an exploratory study)
- Consider rival explanations
  - Can you gather evidence to confirm/reject alternatives to the theory under investigation?

[adapted from Easterbrook *et al.*]

Questions?

- Let's look at an example…
- This example is from a long time ago, when I was a graduate student
  - It was well-respected at the time, and the paper even won a 10-year retrospective award
  - But this was early in the world of applying case studies in PLs—so there are also things to criticize!

ArchJava

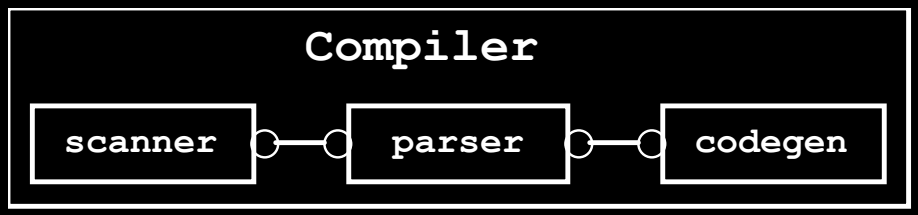
Connecting Software Architecture to Implementation

Jonathan Aldrich
Craig Chambers
David Notkin

University of Washington

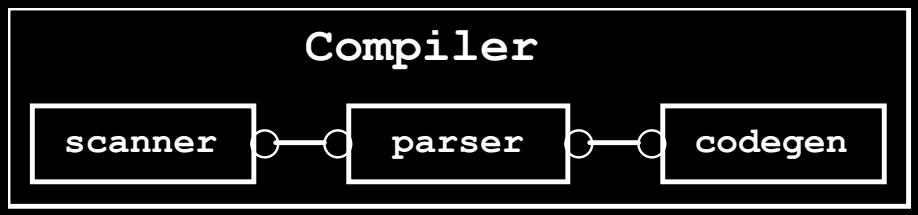
ICSE '02, May 22, 2002

Software Architecture



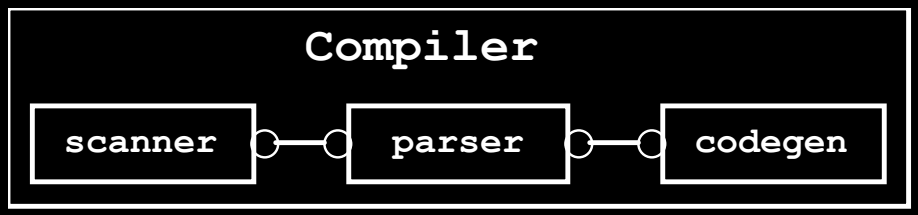
- High-level system structure
  - Components and connections
- Automated analysis
- Support program evolution
  - Source of defect
  - Effect of change
  - Invariants to preserve

Architecture and Implementation



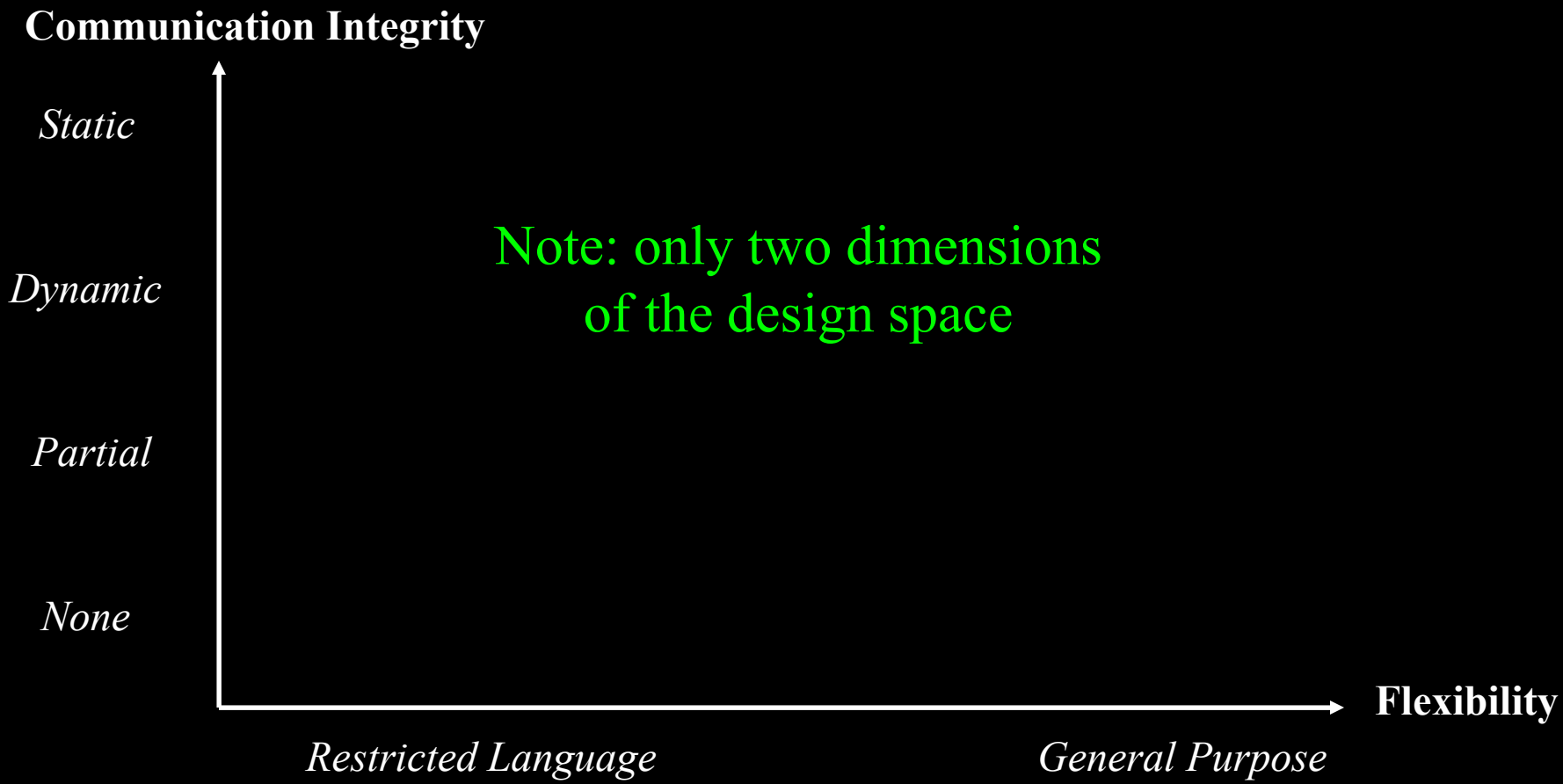
- Inconsistency caused by evolution
  - Architecture documentation becomes obsolete
- Problems
  - Surprises
  - Misunderstandings lead to defects
  - Untrusted architecture won't be used

Architecture and Implementation

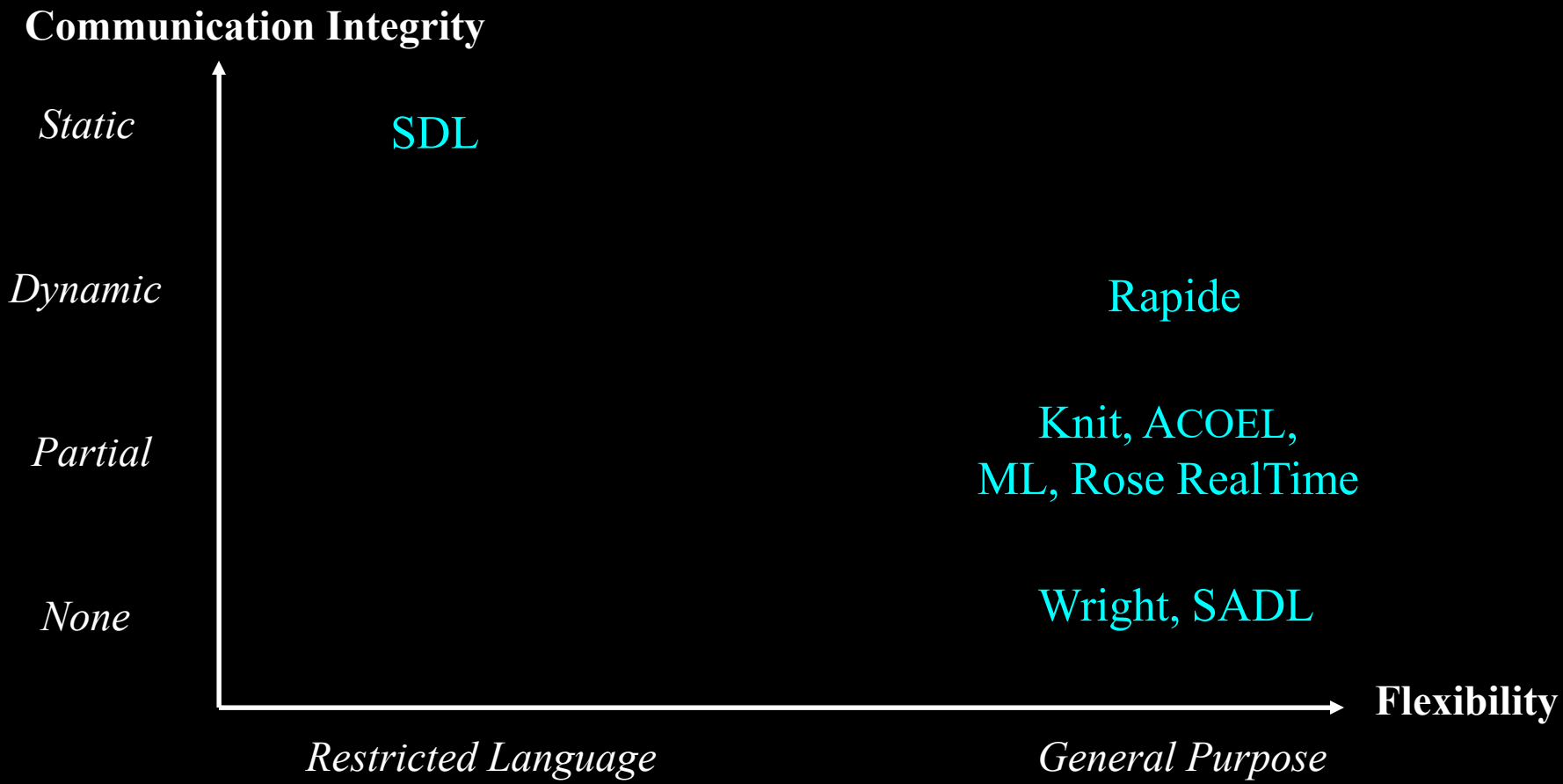


- Does code conform to architecture?
- Communication integrity [LV95,MQR95]
  - All communication is documented
    - Interfaces *and* connectivity
  - Enables effective architectural reasoning
    - Quickly learn how components fit together
    - Local information is sufficient

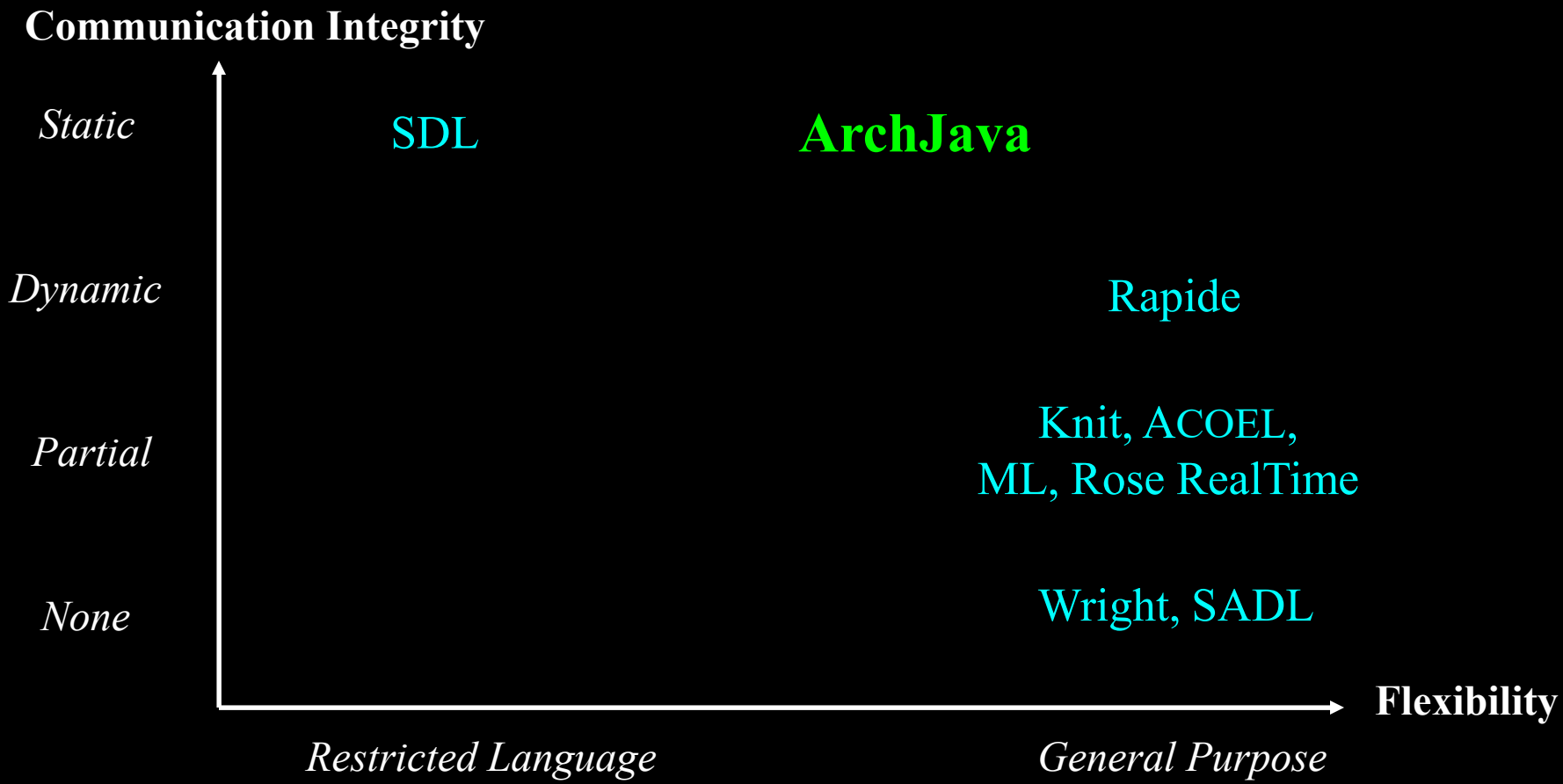
Architectural Approaches:
Checking vs. Flexibility



Architectural Approaches:
Checking vs. Flexibility



Architectural Approaches:
Checking vs. Flexibility



ArchJava

- Approach: add architecture to language
  - Control-flow communication integrity
    - Enforced by type system
  - Architecture updated as code evolves
  - Flexible
    - Dynamically changing architectures
    - Common implementation techniques
- Case study: is it *practical* and *useful*?

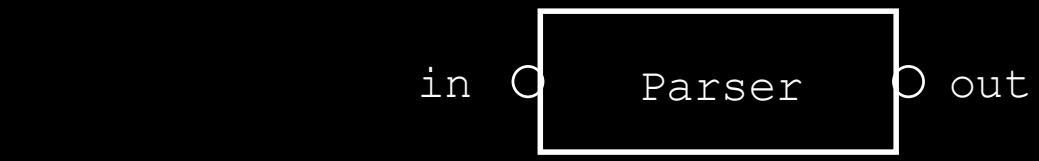
A Parser Component


```
public component class Parser {
```

Component class

- Defines architectural object
- Must obey architectural constraints

A Parser Component

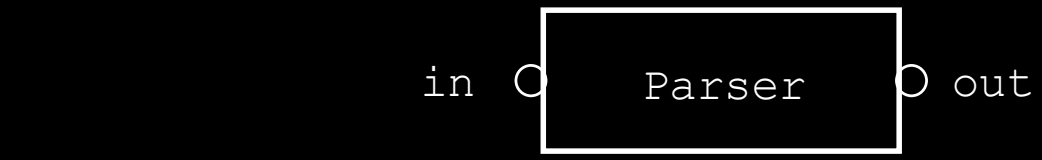



```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
```

Components communicate through *Ports*

- A two-way interface
- Define *provided* and *required* methods

A Parser Component

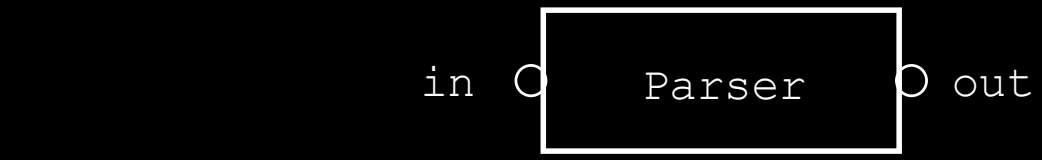



```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
```

Ordinary (non-component) objects

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

A Parser Component




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

Can fill in architecture with ordinary Java code

Hierarchical Composition


```
public component class Compiler {
  private final Scanner scanner = new Scanner();
  private final Parser parser = new Parser();
  private final CodeGen codegen = new CodeGen();
```

Subcomponents

- Component instances inside another component
- Communicate through connected ports

Hierarchical Composition


```
public component class Compiler {
  private final Scanner scanner = new Scanner();
  private final Parser parser = new Parser();
  private final CodeGen codegen = new CodeGen();
  connect scanner.out, parser.in;
  connect parser.out, codegen.in;
```

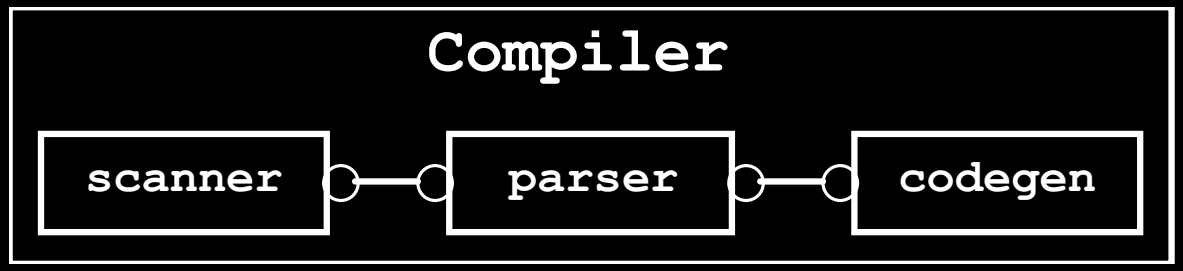
Connections

- Bind required methods to provided methods

Evaluation Questions

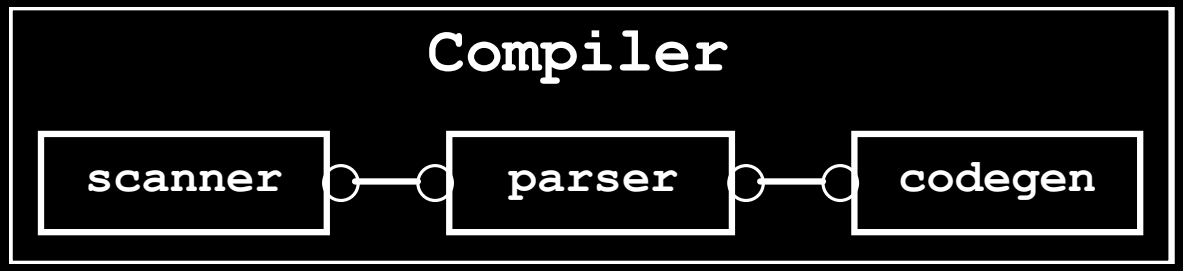
- Does ArchJava guarantee *communication integrity*?
- Is ArchJava *expressive* enough for real systems?
- Can ArchJava aid *software evolution* tasks?

Communication Integrity



A component may only communicate with the components it is connected to in the architecture

Communication Integrity

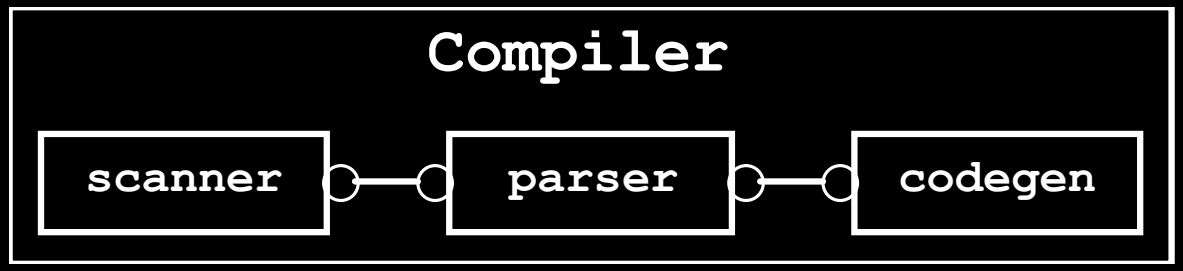


A component may only communicate with the components it is connected to in the architecture

ArchJava enforces integrity for control flow

- No method calls permitted from one component to another *except*
  - From a parent to its immediate subcomponents
  - Through connections in the architecture

Communication Integrity



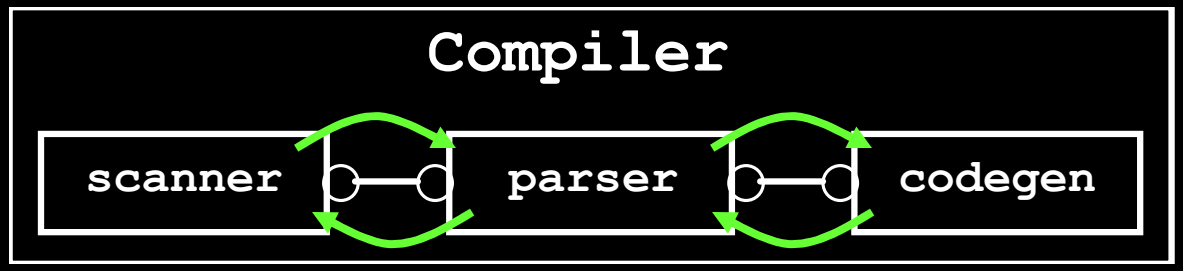
A component may only communicate with the components it is connected to in the architecture

ArchJava enforces integrity for control flow

Other communication paths

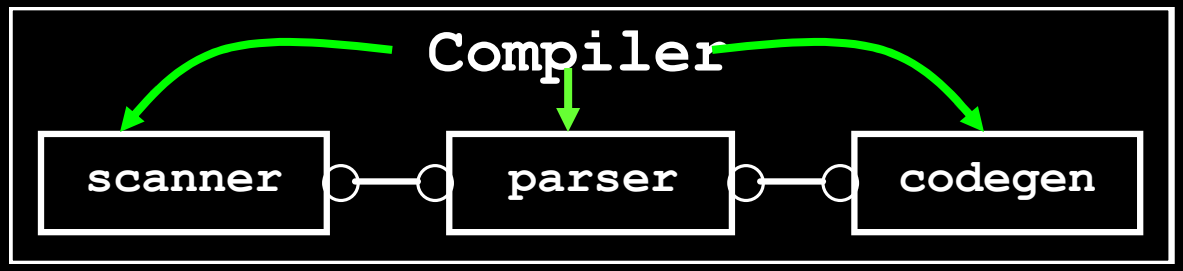
- Shared data (current work)
- Run-time system

Control Communication Integrity



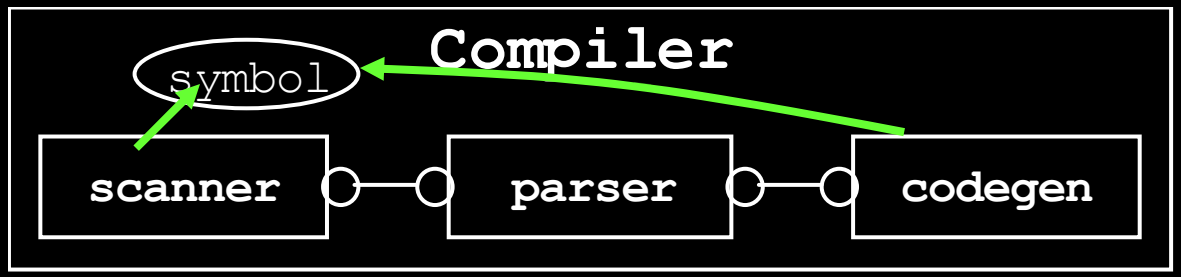
- Architecture allows
  - Calls between connected components

Control Communication Integrity



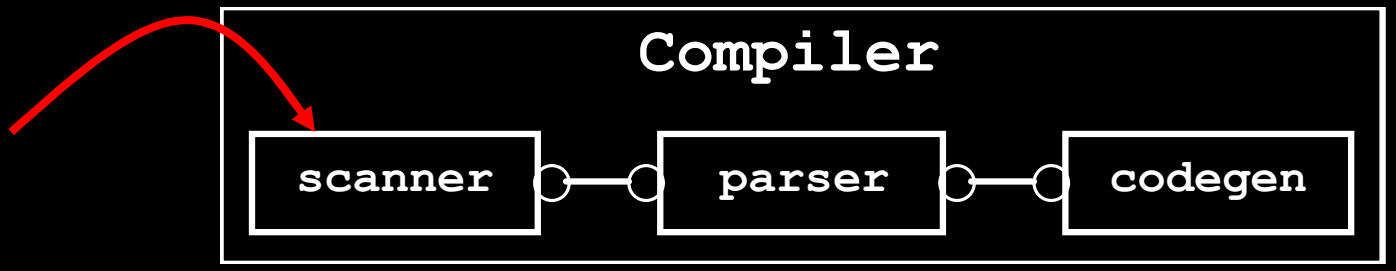
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its immediate subcomponents

Control Communication Integrity



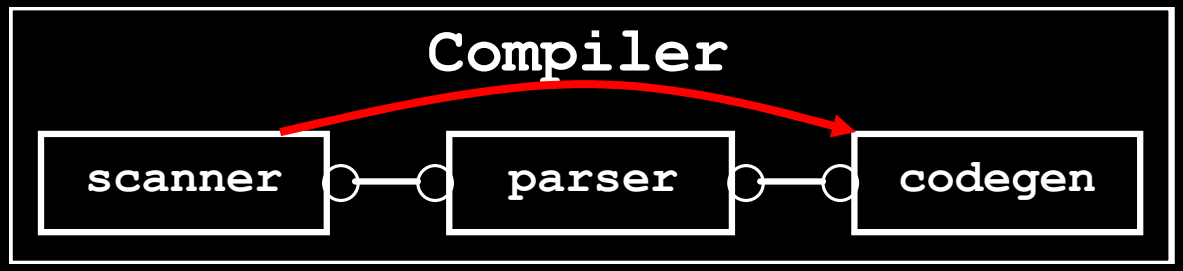
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its immediate subcomponents
  - Calls to shared objects

Control Communication Integrity



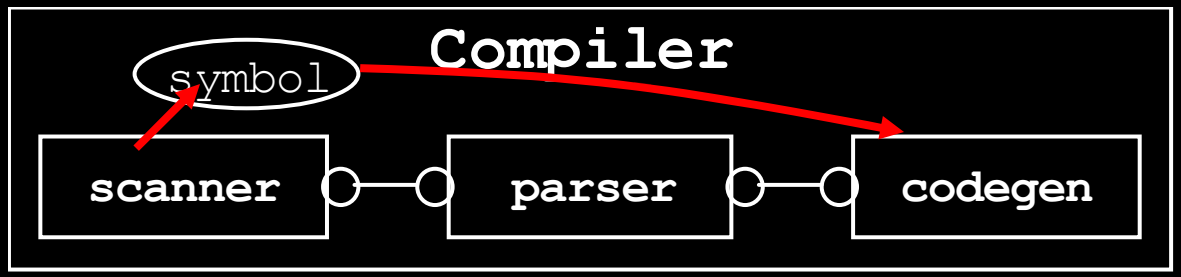
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its immediate subcomponents
  - Calls to shared objects
- Architecture forbids
  - External calls to subcomponents

Control Communication Integrity



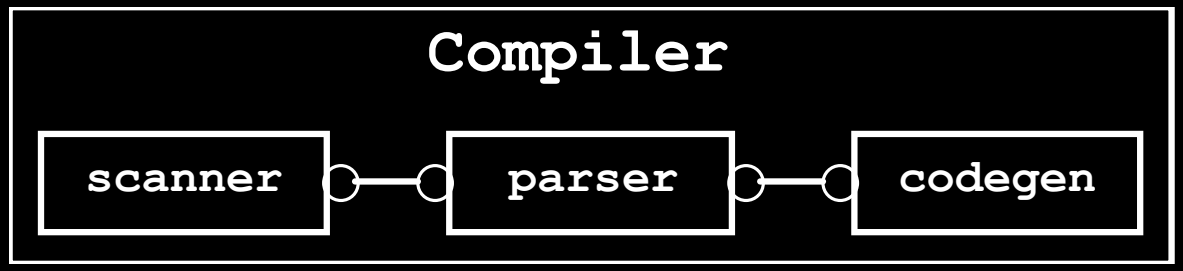
- Architecture allows
  - Calls between connected components
  - Calls from a parent to its immediate subcomponents
  - Calls to shared objects
- Architecture forbids
  - External calls to subcomponents
  - Calls between unconnected subcomponents

Control Communication Integrity



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

Control Communication Integrity



- Architecture allows
  - Calls between connected components
  - Calls from a parent to its immediate subcomponents
  - Calls to shared objects
- Architecture forbids
  - External calls to subcomponents
  - Calls between unconnected subcomponents
  - Calls through shared objects
- Benefit: local reasoning about control flow

Enforcing Control-flow Integrity

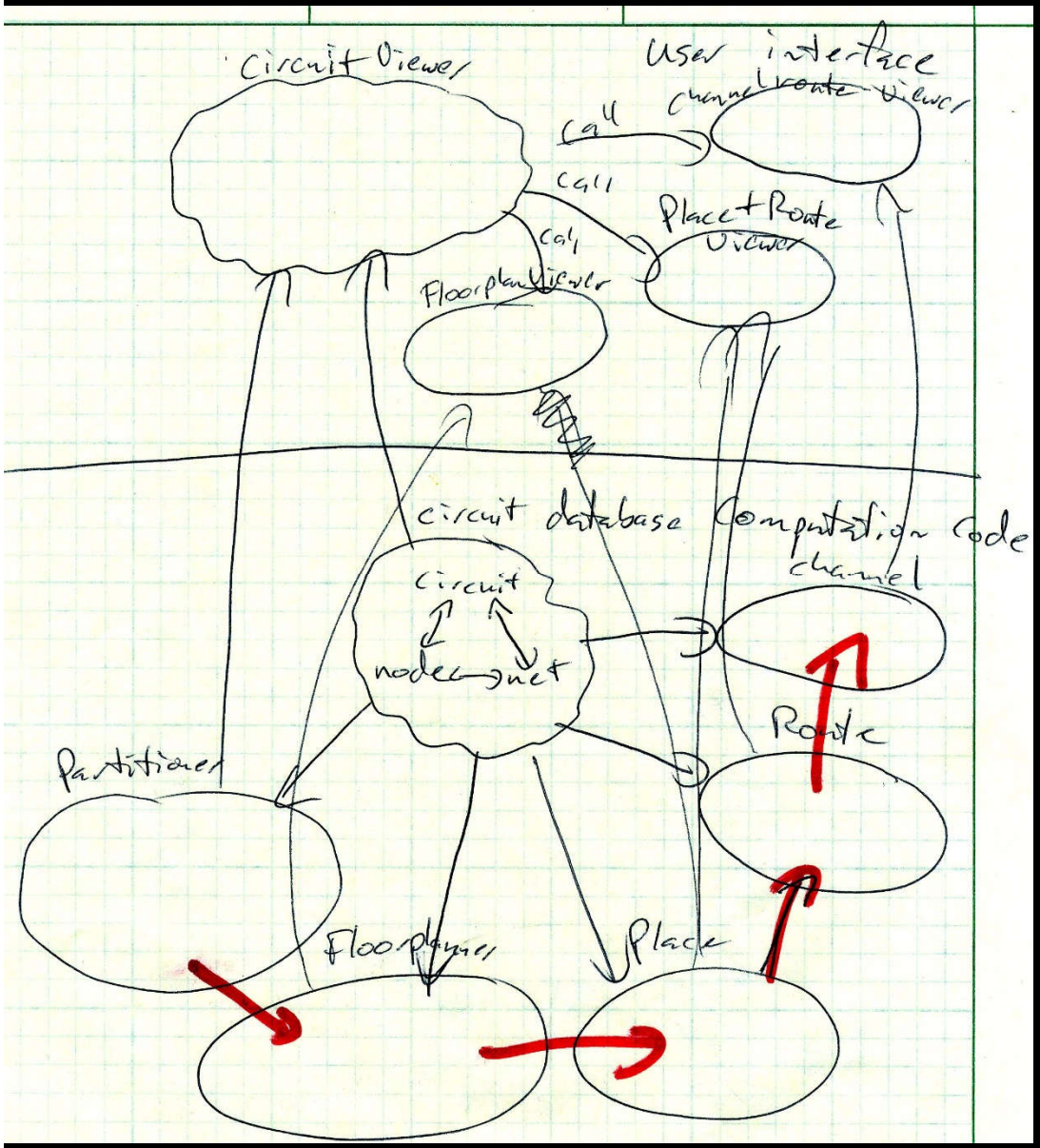
- Type system invariant
  - *Components can only get a typed reference to subcomponents and connected components*
  - Prohibits illegal calls
- Informal description in ICSE paper
  - Formalization and proof to appear in ECOOP '02

Evaluation Questions

- Does ArchJava guarantee *control communication integrity*?
  - ***Yes, using the type system***
- Is ArchJava *expressive* enough for real systems?
- Can ArchJava aid *software evolution* tasks?

Evaluation Questions

- Does ArchJava guarantee *control communication integrity*?
  - ***Yes, using the type system***
- Is ArchJava *expressive* enough for real systems?
- Can ArchJava aid *software evolution* tasks?
- Case study: Aphyds
  - 12,000 lines of Java code
  - Original developer drew architecture for us
  - Our task: express the architecture in ArchJava



Aphyds Architecture

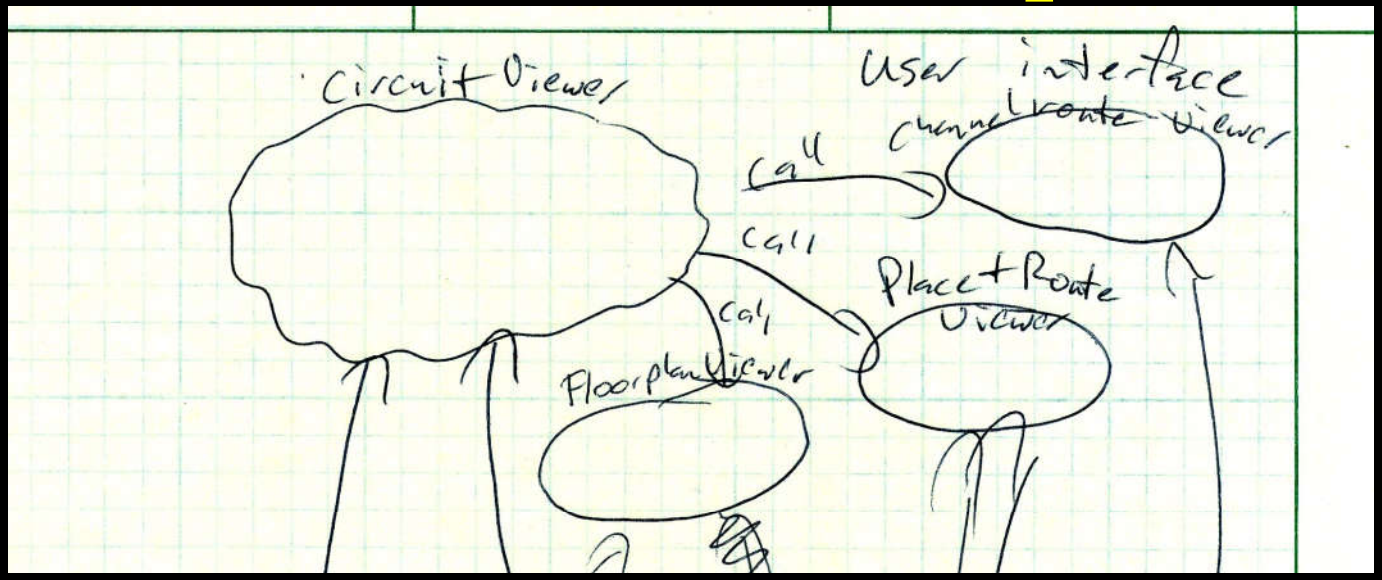
- UI above
  - Main window
  - 3 secondary windows
- Circuit DB below
  - Central DB
  - 5 comput. modules
- Arrows
  - Data & control flow

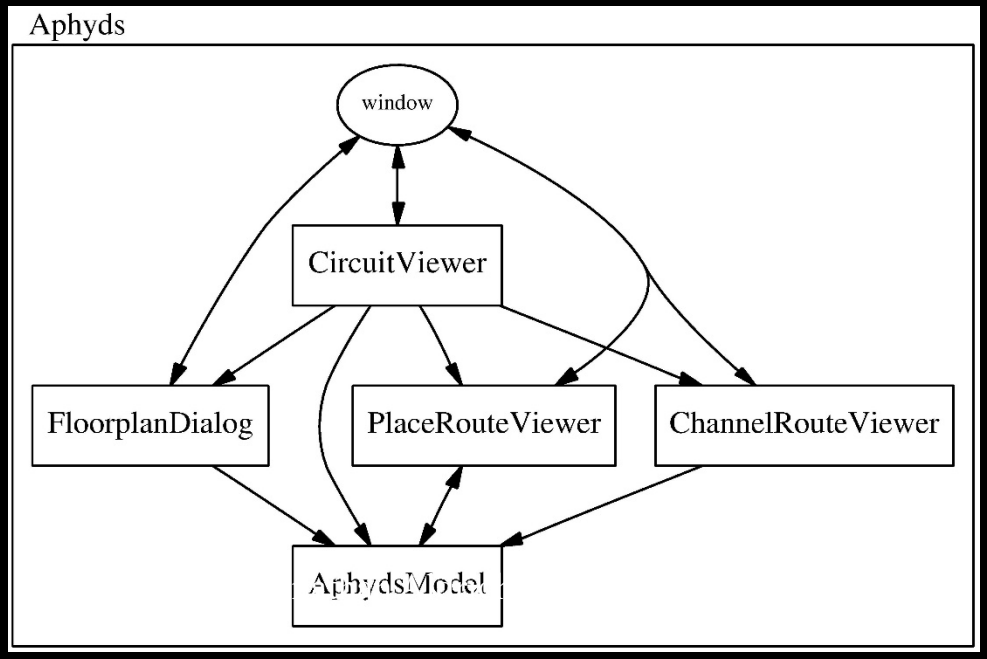
Aphyds Architecture

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

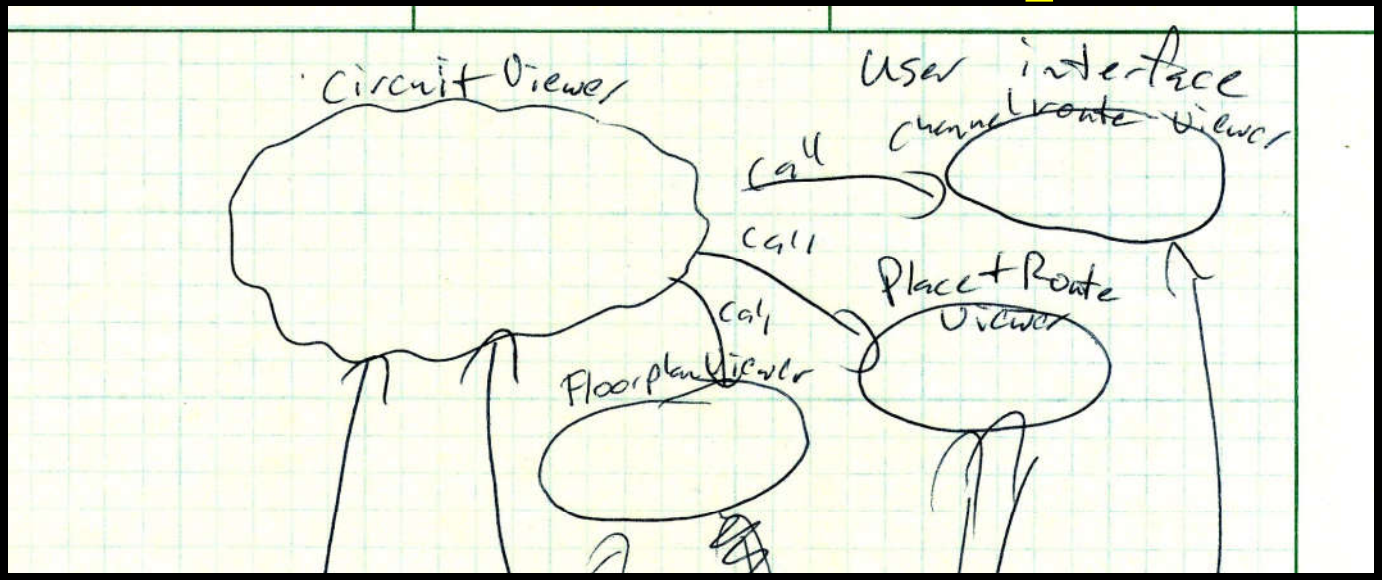
Hypothesis: Developers have a conceptual model of their architecture that is mostly accurate, but this model may be a simplification of reality, and it is often not explicit in the code.

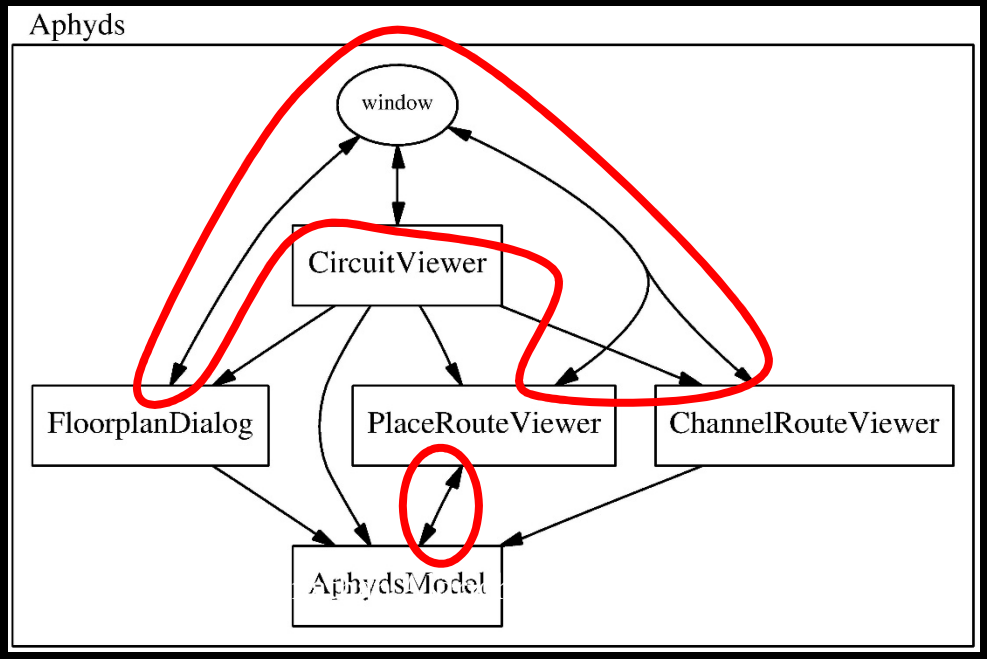
UI Architecture Comparison





UI Architecture Comparison





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.

Evaluation Questions

- Does ArchJava guarantee *control communication integrity*?
  - ***Yes***
- Is ArchJava *expressive* enough for real systems?
  - ***Yes (further validated other case studies)***

Evaluation Questions

- Does ArchJava guarantee *control communication integrity*?
  - ***Yes***
- Is ArchJava *expressive* enough for real systems?
  - ***Yes (validated by 2 other case studies)***
- Can ArchJava aid *software evolution* tasks?
- Three experiments
  - Understanding Aphyds communication
  - Reengineering Aphyds' architecture
  - Repairing a defect

Program Understanding

Communication between the main structures is awkward, especially the change propagation messages

– Aphyds developer, initial interview

- Communication analysis aided by ArchJava
  - Ports group related methods
    - provided ***and*** required interfaces
  - Connections show relationships
- Discovered refactoring opportunities

Program Understanding

Communication between the main structures is awkward, especially the change propagation messages

– Aphyds developer, initial interview

- Communication analysis aided by ArchJava
  - Ports group related methods
    - provided ***and*** required interfaces
  - Connections show relationships
- Discovered refactoring opportunities

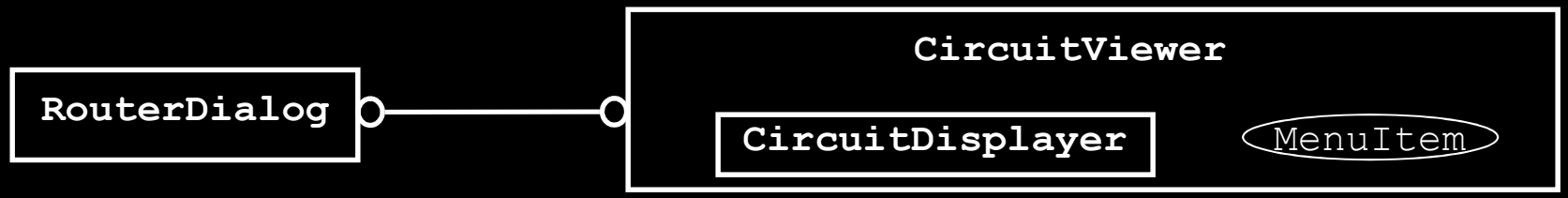
Hypothesis: Expressing software architecture in ArchJava highlights refactoring opportunities by making communication protocols explicit.

Reengineering Aphyds


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

- Highly coupled code
  - Depends on every link in chain
  - Programs are fragile, change is difficult
- Law of Demeter [Lieberherr et al.]
  - Design guideline
  - "Only talk with your neighbors"

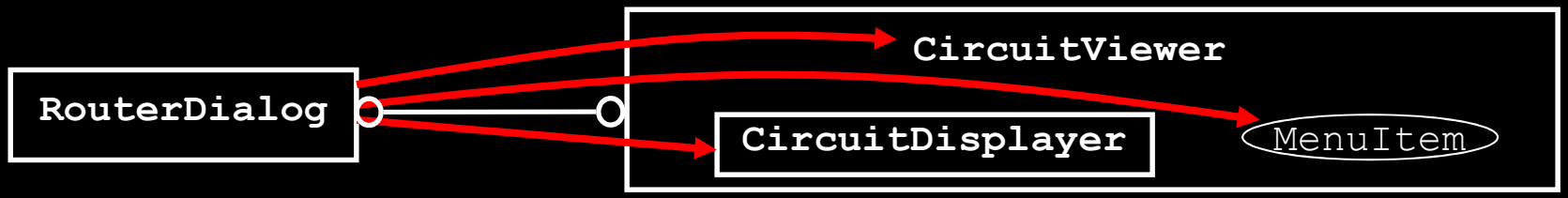
Reengineering Aphyds




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

- Control communication integrity
  - Components only talk with connected components

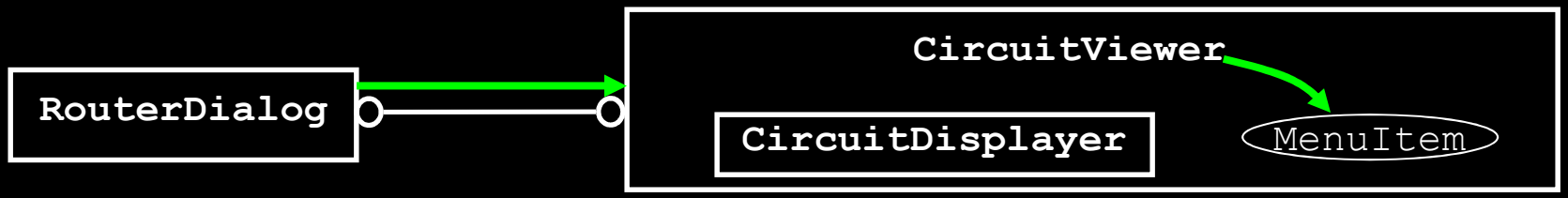
Reengineering Aphyds




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

- Control communication integrity
  - Components only talk with connected components
- Compile-time error in ArchJava
  - RouterDialog can only reference local connections

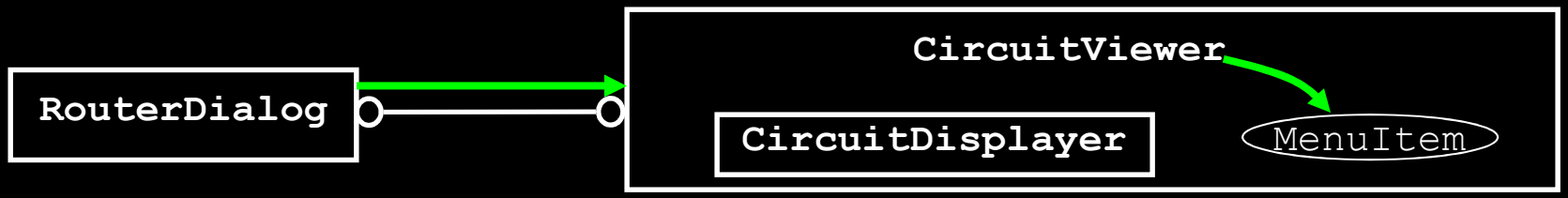
Reengineering Aphyds




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

- Control communication integrity
  - Components only talk with connected components
- Compile-time error in ArchJava
  - RouterDialog can only reference local connections
  - Call through architecture, reducing coupling

Reengineering Aphyds




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

- Control communication integrity
  - Components only talk with connected components
- Compile-time error in ArchJava
  - RouterDialog can only reference local connections
  - Call through architecture, reducing coupling

Hypothesis: Enforcing communication integrity helps to reduce system coupling

Defect Repair

- Fix same Aphyds bug
  - First in ArchJava, then Java
- ArchJava version required more coding
  - Had to add new ports & connections

Defect Repair

- Fix same Aphyds bug
  - First in ArchJava, then Java
- ArchJava version required more coding
  - Had to add new ports & connections
- Java version took longer
  - Difficult to find object involved in fix
    - Had to traverse a sequence of hard-to-find field links
  - Even though we had already fixed the bug in ArchJava

Defect Repair

- Fix same Aphyds bug
  - First in ArchJava, then Java
- ArchJava version required more coding
  - Had to add new ports & connections
- Java version took longer
  - Difficult to find object involved in fix
    - Had to traverse a sequence of hard-to-find field links
  - Even though we had already fixed the bug in ArchJava

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

Evaluation Questions

- Is ArchJava *expressive* enough for real systems?
  - ***Yes***
- Can ArchJava aid *software evolution* tasks?
  - Potential benefits observed
    - Highlights refactoring opportunities
    - Encourages loose coupling
    - May aid defect repair

Conclusion

- ArchJava integrates architecture with Java code
- Control communication integrity
  - Keeps architecture and code synchronized
- Initial experience
  - ArchJava can express real program architectures
  - ArchJava may aid in software evolution tasks
- ***Download the ArchJava compiler and tools***

http://www.archjava.org/

Discussion

- Consider the ArchJava case study?
- What did it accomplish?
- How would you criticize it?
- Forms of validity
  - Construct: are concepts operationalized and measured correctly?
  - Internal: properly establishing causal relationships?
  - External: to what domain can the findings be generalized?
  - Reliability: is the study repeatable with the same results?

Sources/References

- ArchJava: Connecting Software Architecture to Implementation. Jonathan Aldrich, Craig Chambers, and David Notkin. Proc. International Conference on Software Engineering (ICSE '02), May 2002. http://www.cs.cmu.edu/~aldrich/papers/icse02.pdf
- Case Studies for Software Engineers. Steve Easterbrook and Jorge Aranda. Tutorial at ICSE 2006. http://www.cs.toronto.edu/~sme/case-studies/case_study_tutorial_slides.pdf
- Case Study Research: Design and Methods. Robert K. Yin. SAGE Publications, 2017.