ArchJava: Connecting Software Architecture to Implementation

By Jonathan Aldrich, Craig Chambers, and David Notkin

Presented By: Udaya Ramya Godavarthy

Texas State University

September 28, 2021

Table of Contents

Software Architecture

- Describes the structure of a system
- Enables more effective design, program understanding and formal analysis
- Existing approaches decouple implementation code from architecture
- Leads to inconsistencies, confusion, violation of architectural properties
- Inhibits software evolution

What is Software Architecture?

It is the organization of a software system as a

- Collection of components
- Connections between components
- Showing constraints on component interaction

Architecture Description Languages (ADL)

- Describing architecture in a formal architecture description language (ADL) can aid in the specification and analysis of high-level designs.
- Limitations:
  - Existing ADL's are loosely coupled to implementation languages
  - Causes problems in analysis, implementation, understanding and evolution of software systems.
- In summary, while architectural analysis in existing ADLs may reveal important architectural properties but these properties are not guaranteed to hold in implentation.

Module Interconnection Languages (MIL)

- MILs support system composition from separate modules.
- ADLs make connectors explicit to describe data and control flow between components.
- MILs describe the uses relationship between modules.
- MILs cannot be used to describe dynamic architectures, where component object instances are created and linked together at run time.
- MILs provide encapsulation by hiding names.

ArchJava Language

- ArchJava is an extension to Java
- Seamlessly unifies software architectural structure and implementation in one language
- Allows flexible implementation techniques
- Ensures traceability between architecture and code
- Guarantees communication integrity even in presence of advanced architectural features like run time component creation and connection
- Supports dynamic changes in distributed architectures

Key Benefits of ArchJava

- include better program understanding
- reliable architectural reasoning about code
- keeping architecture and code consistent as they evolve
- encouraging more developers to take advantage of software evolution

Language Constructs

To allow programmers to describe software architecture, ArchJava adds new language constructs:

- components
- connections and
- ports

Components

- A component is a special kind of object that communicates with other components in a structured way.
- Components communicate at same level through explicitly declared ports
- Regular method calls are not allowed

Ports

- A port represents a logical communication channel between a component and one or more components that it is connected to.
- Ports declare three sets of methods specified using requires, provides, and braodcasts keywords

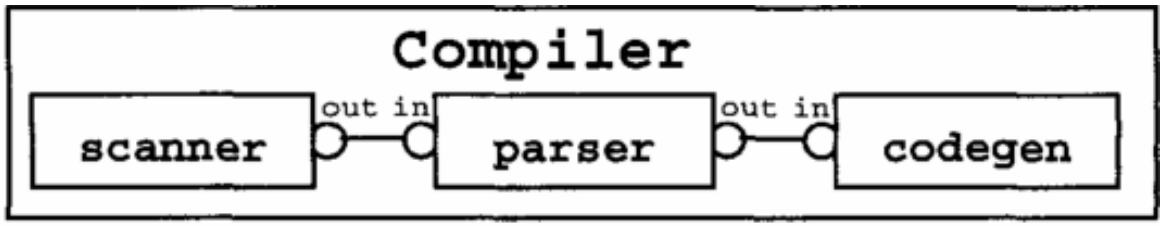
```
public component class Parser {
  public port in {
    provides void setInfo(Token symbol,
                          SymTabEntry e);
    requires Token nextToken()
                    throws ScanException;
  }
  public port out {
    provides SymTabEntry getInfo(Token t);
    requires void compile(AST ast);
  }

  void parse(String file) {
    Token tok = in.nextToken();
    AST ast = parseFile(tok);
    out.compile(ast);
  }

  AST parseFile(Token lookahead) { ... }
  void setInfo(Token t, SymTabEntry e) {...}
  SymTabEntry getInfo(Token t) { ... }
  ...
}
```

Figure: A parser component in ArchJava

Connections

- Connection consistency checks ensure that each required method is bound to a unique provided method
- The symmetric connect primitive connects two or more port together, binding each required methods to provided method with the same name and signature.
- Alternative connection sematics implemented by writing 'smart connector' components

Compiler Architecture and its ArchJava Representation




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

  connect scanner.out, parser.in;
  connect parser.out, codegen.in;

  public static void main(String args[]) {
    new Compiler().compile(args);
  }

  public void compile(String args[]) {
    // for each file in args do:
    ...parser.parse(file);...
  }
}
```

Communication Integrity

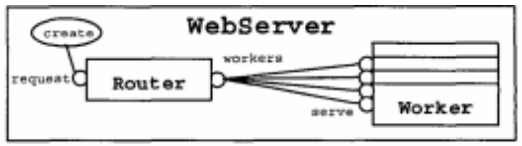
- Components in an architrecture can only call each other's methods along declared connections between ports.
- Components are only able to directly invoke methods of their subcomponents.

Dynamic Architectures

- Covered by creating and connecting together a dynamically determined number of components.
- Connected using a connect expression at run time
- Each connect expression must match a connection pattern declared in the enclosing component.
- A port interface describes a port that can be instantiated several times to communicate through different connections.
- Just as Java does not provide a way to explicitly delete objects, ArchJava does not provide a way to explicitly remove components and connections.
- Components are garbage-collected when they are no longer reachable through direct references or connections.

A Web Server Architecture

- Router subcomponent receives incomming HTTP requests and passes them to Worker components that serve the request.
- The requestWorker method of the web server dynamically creates a Worker component and then connects its serve port to the workers port of the Router.




```
public component class WebServer {
  private final Router r = new Router();
  connect r.request, create;
  connect pattern Router.workers, Worker.serve;

  private port create {
    provides r.workers requestWorker() {
      final Worker newWorker = new Worker();
      r.workers connection
        = connect(r.workers, newWorker.serve);
      return connection;
    }
  }
  public void run() { r.listen(); }
}

public component class Router {
  public port interface workers {
    requires void httpRequest(InputStream in,
                              OutputStream out);
  }
  public port request {
    requires this.workers requestWorker();
  }
  public void listen() {
    ServerSocket server = new ServerSocket(80);
    while (true) {
      Socket sock = server.accept();
      this.workers conn = request.requestWorker();
      conn.httpRequest(sock.getInputStream(),
                       sock.getOutputStream());
    }
} }

public component class Worker extends Thread {
  public port serve {
    provides void httpRequest(InputStream in,
                              OutputStream out) {
      this.in = in; this.out = out; start();
    }
  }
  public void run() {
    File f = getRequestedFile(in);
    sendHeaders(out);
    copyFile(f, out);
  }
  // more method & data declarations...
}
```

Limitations of ArchJava

- Only applicable to programs in a single language and running on a single JVM.
- Architectures in ArchJava are more concrete than architectures in ADLs, restricting the ways in which a given architecture can be implemented.
- Because of focus on communication integrity, this does not support other architectural reasoning such as connection protocols, architectural styles, or component multiplicity.

Evaluation

In order to determine whether the ArchJava language meets its design goals, following questions were addressed:

1. Can ArchJava express the architecture of a real program of significant complexity?
2. How difficult is it to reengineer a Java program in order to express its architecture explicitly in ArchJava?
3. Does expressing a program's architecture in ArchJava help or hinder software evolution?

Hypothesis

Hypothesis 1:

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.

Hypothesis 2:

Programming languages that prohibit sharing data between components are too inflexible to express the natural architecture for many programs.

Hypothesis cont'd

Hypothesis 3:

Describing an existing program's architecture with ArchJava may involve significant restructuring if the desired architecture does not match the implementation well.

Hypothesis 4:

Refactoring an application to expose its architecture is done most eficiently in small increments.

Hypothesis 5:

Applications can be translated into ArchJava with a modest amount of effort, and without excessive code bloat.

Hypothesis cont'd

Hypothesis 6:

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

Hypothesis 7:

Using separate ports and connections to distinguish diflerent protocols and describing protocols with separate provided and required port interfaces may ease program understanding tasks.

Hypothesis cont'd

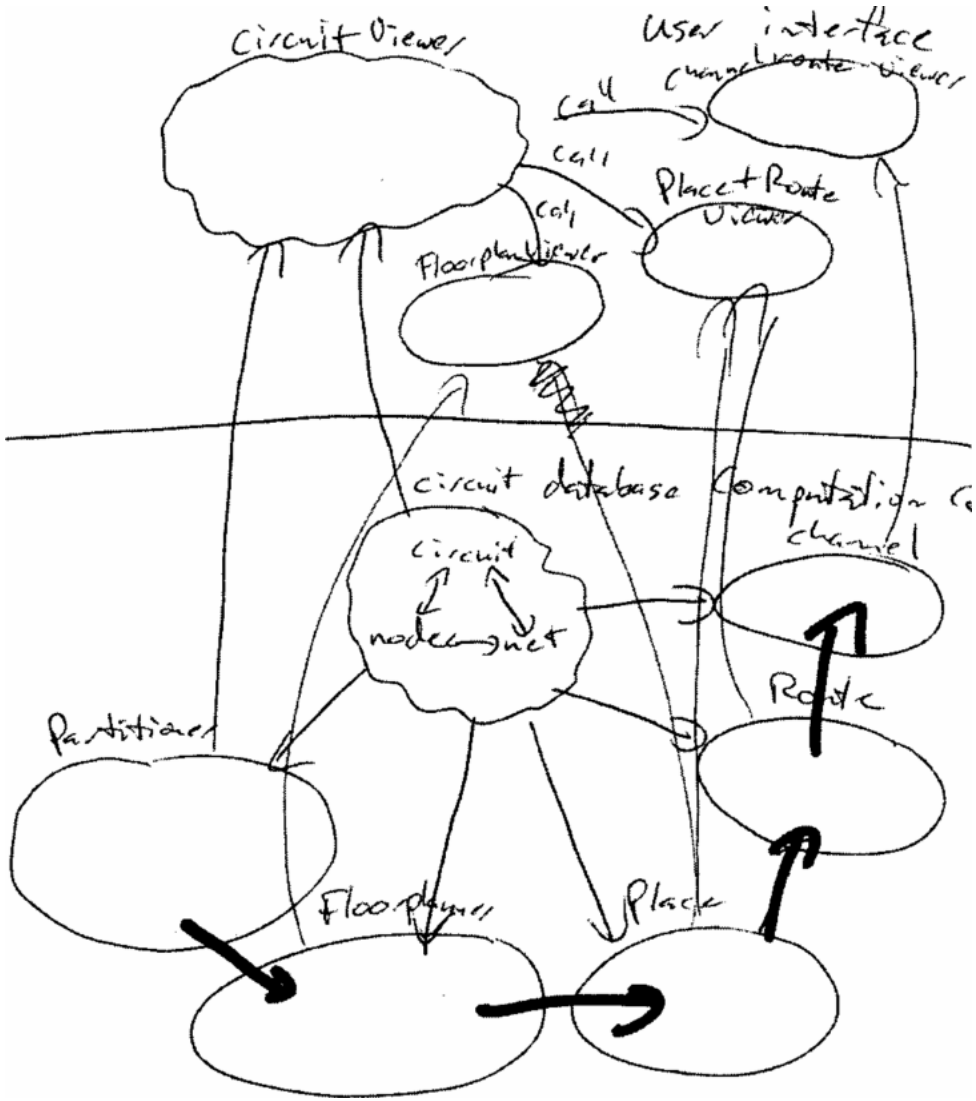
Hypothesis 8:

Communication integrity in ArchJava encourages local communication and helps to reduce coupling between components.

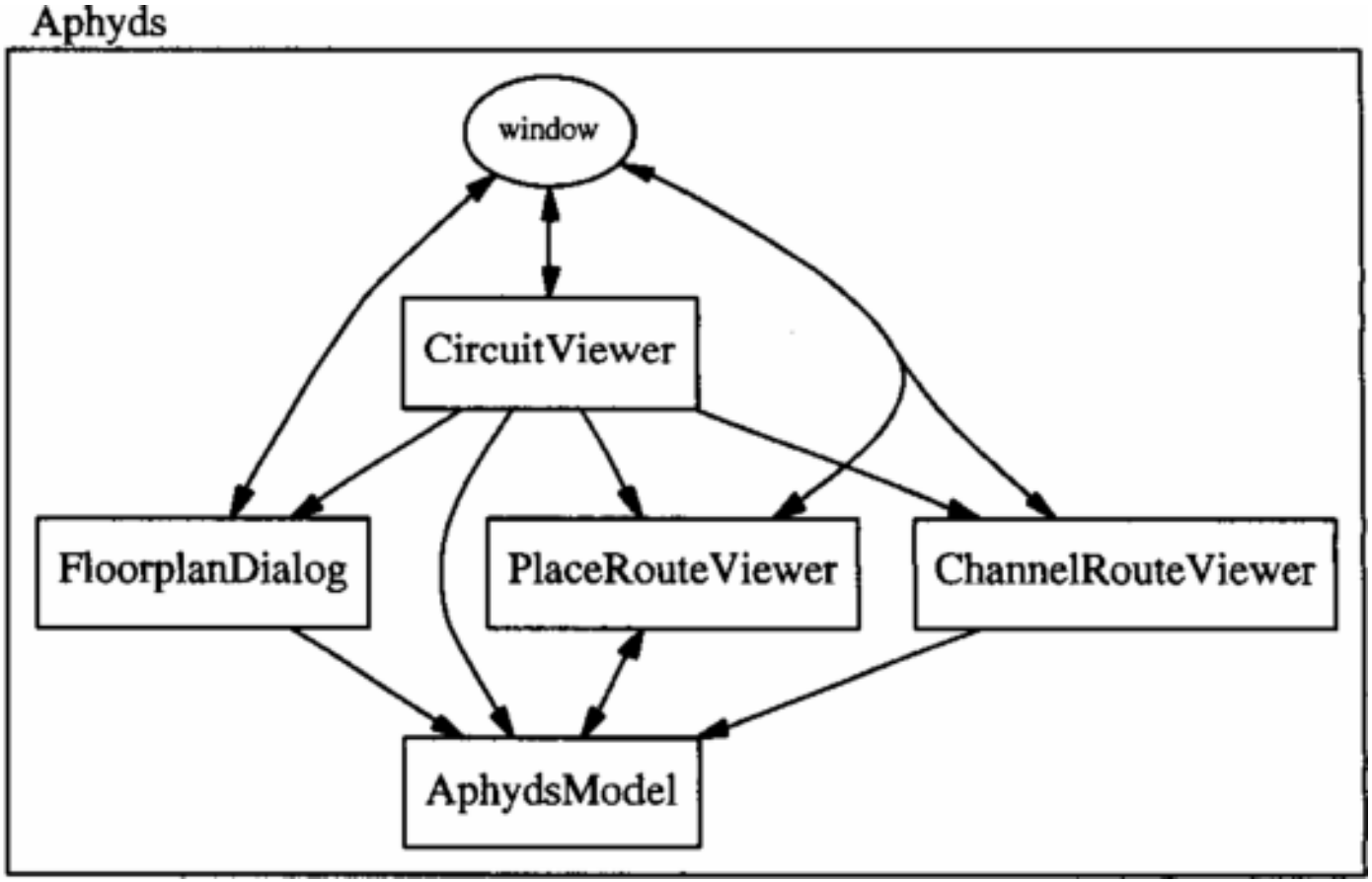
Hypothesis 9:

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

Aphyds Architecture



Aphyds Architecture cont'd



References

Aldrich, Jonathan, Craig Chambers, and David Notkin. “ArchJava: Connecting software architecture to implementation.” *Proceedings of the 24th International Conference on Software Engineering*. ICSE 2002. IEEE, 2002.