Lecture 5
Software Architecture
Announcement

• Project proposal was due yesterday.
• I received your email submission. No worries.
• The project proposal will be graded.
• The next checkpoint (Feb 23rd) for Option A students are not mandatory.
Announcement

- Don’t forget to put a header [EE382V] when emailing me.
- Please cc TA when you send me an email for _all_ your correspondences.
Announcement

• I will select some good student reviews (3pt) and upload them --- of course I will make them anonymous.
Today’s Presentation

- Advocate: Christopher Spandikow
Today’s Lecture on Software Architecture

- We read the software architecture paper by David Garlan and Mary Shaw at CMU.

- Around the same time, Alexander Wolf and Dewayne Perry (back then they were at Bell Lab) also wrote a paper on the idea of software architecture.

- Dr. Perry is an active researcher in Software Architecture and he is here in our department.

- Some of today’s slides are borrowed from Rob DeLine at Microsoft Research, who did his Ph.D under the supervision of Mary Shaw at CMU.

- Some of today’s slides are borrowed from Vibha Sazawal at UMD, who worked with Jonathan Aldrich at CMU, a creator of ArchJava.
What is a software architecture?

• According to Google Images

- wui-webapp
- wui-common
  - entry mapping
  - components
  - repository
  - post-processing
- interldap-authorization
- LQL
- interldap-core
  - schema
  - ldap facade
- Tapestry 5
- Spring
What is a software architecture?

- According to Google Images
What is a software architecture?

- According to Google Images
What is software architecture?

- CMU-SEI definition
  - software elements, the externally visible properties of those elements and the relationships among them
What do these figures mean?

- Boxes
- Lines
- Grouping
What do these figures mean?

- Boxes => Component
- Lines => Connections
- Grouping, backgrounds, fences => Composition
Components (boxes)

- Places where computation takes place
- Places where data is stored
- Box shapes distinguish component types
Connections (lines, arrows)

- Some kind of interaction among components
- Often binary, sometimes n-ary
- Line attributes distinguish connection types
Composition (grouping, backgrounds, fences)

- Show commonality and boundaries
Carving out a new level of abstraction

• In the early age of programming languages...

```
sum := 0;
i := 0;
while (i < 10) {
    sum := sum + i;
i := i + 1;
}
return;
```

10: stconst r0, 0
11: stconst r1, 0
12: stconst r2, 10
13: sub r2,r0,r4
14: bz r4, 18
15: add r1,r0,r1
16: incr r0
17: br 12
18: ret

sum := 0;
i := 0;
while (i < 10) {
    sum := sum + i;
i := i + 1;
} return;

“structured programming”
Architecture as a new abstraction

- Researchers are carving out a higher-level abstraction

```java
s = socket(...);
bind(s, ...);
listen(s, ...);
while (true) {
    x = accept (s, ...);
    receive(x, ...);
    close(x);
}
```
What kinds of jargon have you heard of?

- Client / Server?
- Three-tier architecture
- Implicit invocation / event-driven
- Manager and agent
- Pipeline
- Peer to peer
- Model view controller
- Regular programs built in procedural languages
Software Architecture Styles

- Pipe and filter
- Client and server
- Object oriented
- Publish and subscribe
- Layers
- Microkernel
- Web services
Example: Pipe and Filter

- A filter reads streams of data on its inputs and produces streams of data on its outputs by applying a local transformation.

- Component (Filter)

- Connector (Pipe)

- Constraints
  - filters must be independent => no shared states among filters
  - filters do not know the identity of other filters
  - outputs are the same regardless of ordering of filters
Example: Pipe and Filter

- **Advantages:**
  - programmers can understand the overall input and output behavior as a simple composition of filters
  - reuse: any two filters can be hooked together
  - different types of filters can be easily added or deleted

- **Disadvantages:**
  - not good for interactive applications as each filter provides a complete transformation of input data to output data
  - each filter has to parse and unparse the data
Pipe Line Architecture

- a linear sequence of filters
- e.g. a compiler architecture
Example: Event-based, Implicit Invocation

- Instead of invoking a procedure directly, a component can announce or broadcast one or more events.
- Other components in the system can register an interest in an event by associating a procedure with the event.
- e.g. Java Swing GUI
- Component: modules whose interfaces provide both a collection of procedures and a set of events
- Connector: traditional procedure calls as well as bindings between event announcements and procedure calls
Example: Event-based, Implicit Invocation

- **Constraints**
  - Announcers of events do not know which components will be affected by those events
  - Components cannot make assumptions about order of processing

- **Advantages**
  - Any components can be introduced into a system by registering for the events

- **Disadvantages**
  - Component relinquish control over the computation performed by the system
  - Ordering is difficult to understand, difficult to expect when finished
  - Shared event data
Architecture Description Languages (ADL)

- In the 90s, researchers created many architectural notations.
- grew out of module interconnection languages (1975)
- focus on recording system structure (typically static structure)
- different goals, but many shared concepts
Common Concepts in ADL

- Components (computation)
- Connectors (common disagreement: aren’t these just components?)
- Compositions (combinations of elements to form new elements)
- Architectural Styles (constraints on elements and their composition)
UniCon

Focus on encapsulating complex construction rules
- Editor lets you drag-and-drop elements and hook them up

- Given a system description, UniCon’s compiler produces low-level interaction code build instructions (makefile) that invokes needed tools

Wright

- Focus on making interaction formal
- components interact through ports
- connectors interact through roles
- attachments are made by binding ports to roles
- ports and roles are formally defined as CSP (communicating sequential processes).
- i.e., a **process** description language for defining connector types as a protocol of interaction of components
- what is a process? a “thing” that engages in communication/interaction events in a sequence. an event can have associated data.

Allen & Garlan, “Formalizing architectural connection”, ICSE 1994
Wright Component Description Example

component Split =
    port In = read?x -> In [] read-eof -> close -> ✓
    port Left, Right = write!x -> Out   □ close -> ✓

comp spec =
    let Close = In.close -> Left.close -> Right.close -> ✓
    in    Close []
    In.read?x → Left.write!x →
    (Close [] In.read?x → Right.write!x → computation)

Component type is described as a component-specs plus a set of ports
Wright Connector Description Example

connector Pipe =
role Writer = write!x → Writer ⊲ close → ✓
role Reader = let ExitOnly = close → ✓
in let DoRead = (read?x → Reader [] read-eof → ExitOnly)
in DoRead ExitOnly

 glue = let ReadOnly = Reader.read!y → ReadOnly
[] Reader.read-eof → Reader.close → ✓ []
Reader.close → ✓
in let WriteOnly = Writer.write?x → WriteOnly [] Writer.close
→ ✓
in Writer.write?x → glue [] Reader.read!y → glue
[] Writer.close → ReadOnly [] Reader.close → WriteOnly
spec ∀ Reader.read!y . ∃ Writer.write?x . i=j v x=y
∧ Reader.read-eof ⇒ (Reader.close ∧ #Reader.read = #Writer.write)

Roles: obligation of each participating component.

A glue spec: protocol description (coordination among roles)

Connector type is described as a set of roles and a glue specification.
Wright Connector Description
Example

connector Pipe =
role Writer = write!x → Writer ⊑ close → ✓
role Reader = let ExitOnly = close → ✓
in let DoRead = (read?x → Reader [] read-eof → ExitOnly)
in DoRead ExitOnly
  glue = let ReadOnly = Reader.read!y → ReadOnly
     [] Reader.read-eof → Reader.close → ✓ []
  Reader.close → ✓
in let WriteOnly = Writer.write?x → WriteOnly [] Writer.close
  → ✓
in Writer.write?x → glue [] Reader.read!y → glue
  [] Writer.close → ReadOnly [] Reader.close → WriteOnly
spec ∀ Reader.read!y . ∃ Writer.write?x . i=j ∨ x=y
  ∧ Reader.read-eof ⇒ (Writer.close ∧ #Reader.read = #Writer.write)

Roles specify possible behaviors (the steps that can make up a protocol and possible ordering). Glue describes how behaviors are combined across roles.
Wright System Description

A system composes components and connectors

system Capitalize
  component Split = ...
  connector Pipe = ...
...
instances
  split: Split; p1, p2: Pipe;
attachments
  split.Left as p1.Writer;
  upper.In as p1.Reader;
  split.Right as p2.Writer;
  lower.In as p2.Reader;
  ...
end Capitalize.
ADL to ArchJava

- Existing ADLs decouple implementation code from architecture, allowing inconsistencies, causing confusion, violating architectural properties, and inhibiting software evolution.
ArchJava

- ArchJava is an extension to Java that seamlessly unifies software architecture with implementation.
- It also ensures that the implementation conforms to architectural constraints.
- It ensures traceability between architecture and code and support the co-evolution of architecture and implementation.

ArchJava: Connecting Software Architecture to Implementation, Jonathan Aldrich, Craig Chambers and David Notkin [ICSE 2002]
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) { ... }
...
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) { ... }
  ...
}

A **component** can only communicate with other components through explicitly declared ports; regular method calls between components are not allowed.

A **port** represents a logical communication channel between a component and other components that it is connected to.
ArchJava Component Example

```java
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) { ... }
    ...
}
```

**provides:** a provided method is implemented by the component and is available to be called by other components connected to this port.

**requires:** each required method is provided by some other component connected to this port.

**broadcasts:** the same as required except that they can be connected to any number of implementations and must return void.
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);...
    }
}
ArchJava Connector Example

```
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);...
    }
}
```

**connect**: this primitive connects two or more ports together, binding each required method to a provided method with the same name and signature. Connection consistency checks are performed to ensure that each required method is bound to a unique provided method.
ArchJava: Connector TypeChecking

- ArchJava is integrated with Java
- ArchJava makes dependencies explicit, reduces coupling, and promotes understanding of components in isolation
- ArchJava gives you a mechanism for expressing and checking connections but those connections are modeled as individual method calls
Take away message

- Software Architecture is a high-level abstraction of software design.
- A software architecture is usually specified by its components, connections, and composition mechanism.
- Active research in architecture description languages, architectural styles, and enforcing architecture at an implementation level.