Architectural Refinement: from *ACME* to *ArchJava*

Marwan Abi-Antoun

Credits

- Contributors:
  - Dr. Jonathan Aldrich
  - Dr. David Garlan
  - Bradley Schmerl
  - Tony Tseng
- Paper in submission "*Semantic Issues in Architectural Refinement*"
- OOSPLA Demo next week by Dr. Aldrich

From Architecture to Implementation

- Solution space expands with decrease in abstraction level
- Narrow down implementation space with specific middleware (or runtime libraries)
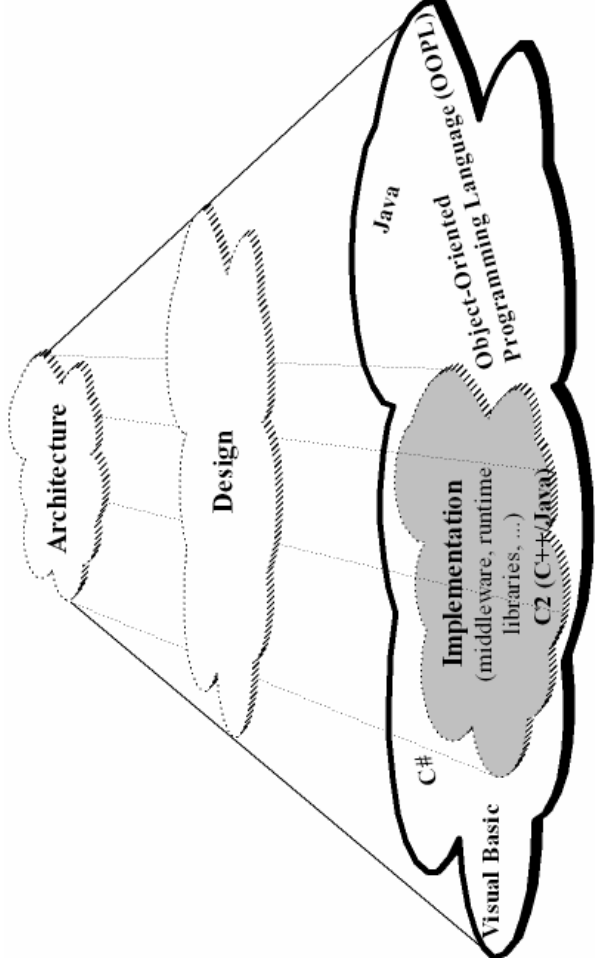
Abstract and Concrete Views

Abstract View

**Concrete View**
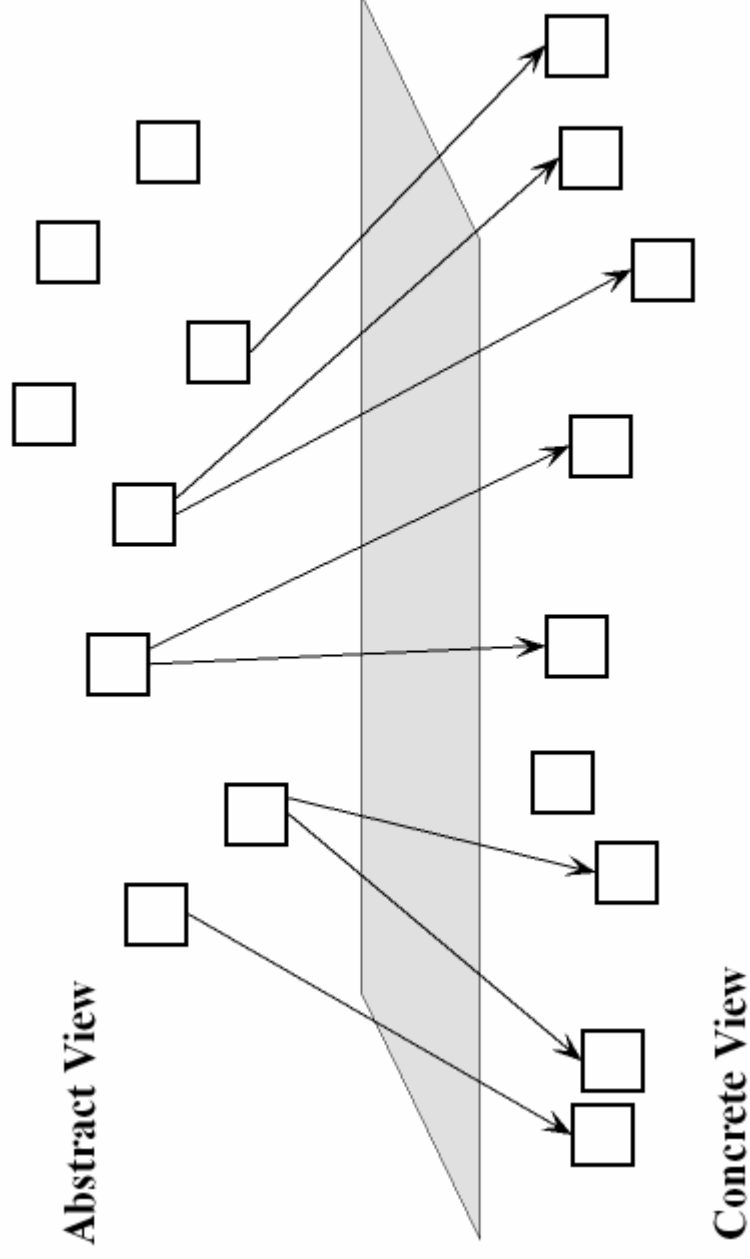
Key Challenges

- Mapping of typing relationships
- Refining connectors
- Handling information loss and overlap
- Illustrated with candidate representations:
  - Abstract: ACME
  - Concrete: ArchJava

ACME in a Nutshell

- Hierarchical representation
  - System, Components, Representations, ...
- Architectural Style or Family
  - Set of components, connectors,...
  - Set of rules
- Built-in checking of architectural rules
- Flexible predicate based type system

Abstract View in ACME

Representation

Component

Connector

Port

Role

Attachment

Binding

+ Property   + Constraint
+ Style .....
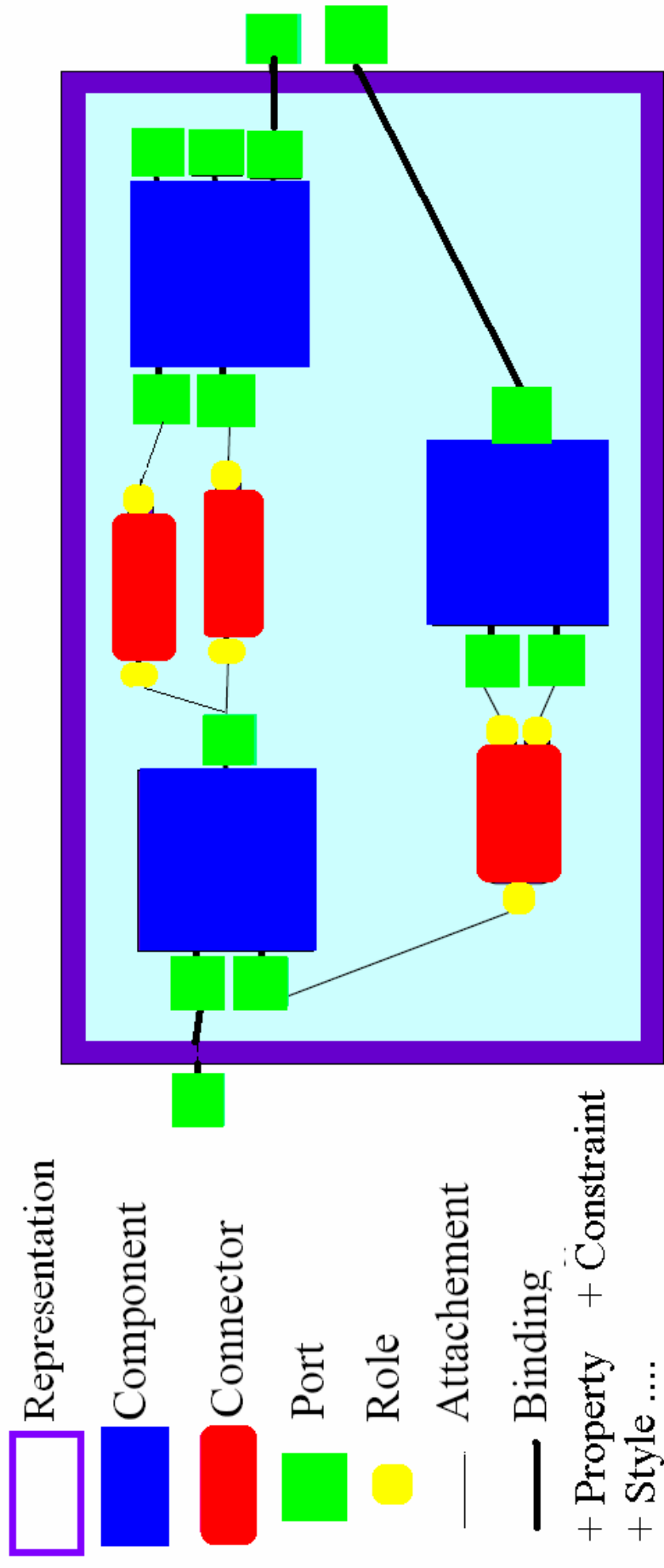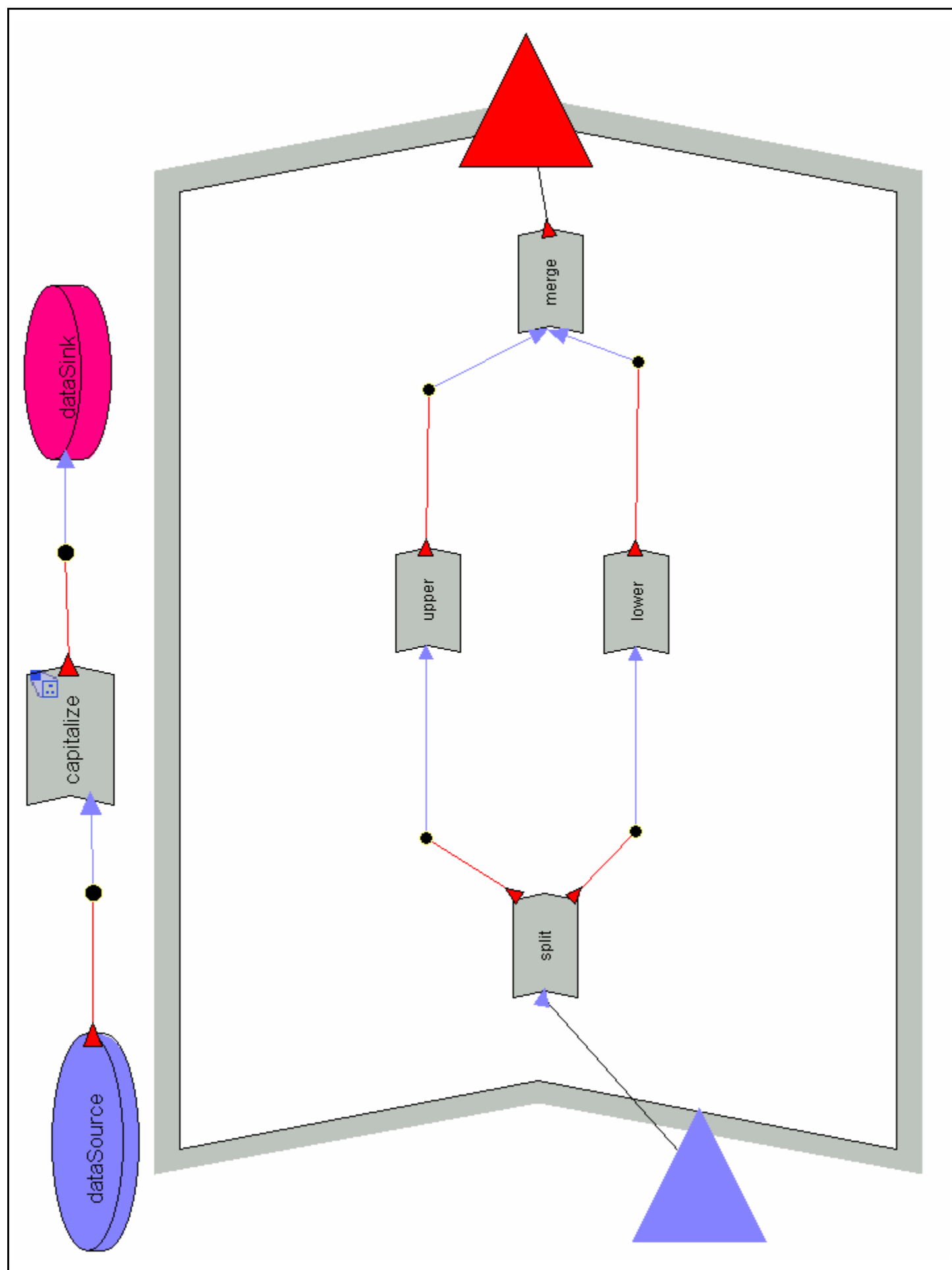
ArchJava in a Nutshell

- Extension of Java programming language
- Specify architecture within the code
  - Built-in Components, Connectors, Ports, Glue...
- Guarantee architectural conformance
  - Type system enforces inter-component communication
  - Component substitutability
- No one-to-one correspondence to ACME:
  - Constraints, Properties, Styles, ...

Concrete View in ArchJava

```
package capitalize;

import java.io.*;

public component class Capitalize {
    private final Split s = new Split();
    private final Upper u = new Upper();
    private final Lower l = new Lower();
    private final Merge m = new Merge();
    private final CharBuffer b = new CharBuffer();

    connect s.out, u.in, l.in;
    connect u.out, m.in1;
    connect l.out, m.in2;

    glue in to b.in;
    connect b.out, s.in;
    glue out to m.out;

    public port in {
        requires int getChars(char b[]);
    }

    public port out {
        provides char getChar() throws IOException;
    }
}
```

Key Ideas

- Indicate choices of refinement in the architectural model and not in the tool
  - Explicitly stored and maintained in the model
  - Can be refined incrementally
  - Used to determine readiness for refinement
  - Fine-grained control over refinement
- Use ACME family (style) as "mixin" style controlling refinement

Mapping Components

ACME Construct	ArchJava
System	Component Class
Component	
Instance	Component Class
	Component Instance
Type	Component Class
Port	Port
Property Set {string}	Required Methods
Property Set {string}	Provided Methods

Mapping Connectors

ACME Construct	ArchJava
Connector	
Instance	Connector Class
	Connector Instance
	Implicit
	Explicit
Type	Connector Class
Role	

Other Mappings

ACME Construct	ArchJava
Binding	Glue
Attachment	Connector Instance
Representation	Java package
Property	
Constraint	
Style	

Architectural Validation

- Components/Connectors/Ports explicitly declaring types

- No dangling ports or roles

- For connected ports, provided and required methods must match

- Component type cannot require more methods than parent type (substitutability)

Current Tool Limitations

- Can only generate ArchJava for an ACME system and not an ACME family

- Cannot exclude portions of architecture model to support iterative refinement

- No support for multi-way connections

- Not generating enough information to simplify round-trip engineering

Open Issues

- ArchJava Limitation:
  - Component Substitutability too restrictive
- Adding too much detail to abstract view
  - E.g., family should not encode signature of required and provided methods
- Tolerating incompleteness
  - Support iterative refinement
- ACME Limitation:
  - No true "mixin" support

References

- ACME:
  - http://www.cs.cmu.edu/~acme
- ArchJava:
  - http://archjava.fluid.cs.cmu.edu/