

# Improving System Dependability by Enforcing Architectural Intent

Marwan Abi-Antoun    Jonathan Aldrich

David Garlan    Bradley Schmerl

Nagi Nahas    Tony Tseng

# Designing Dependability

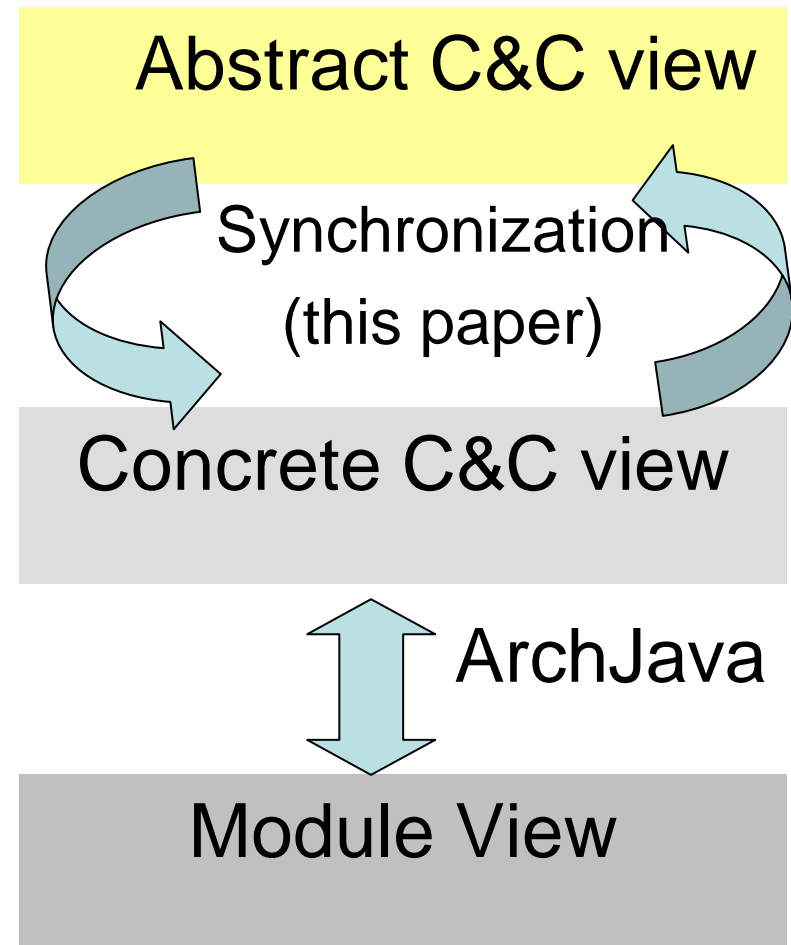
- Dependability analyses
  - Performance, reliability, fault-tolerance...
- Many architectural description languages, reasoning techniques
  - Examples: Rapide, Wright, Meta-H

# Delivering Dependability

- Designed dependability achieved only if implementation conforms to design
- Implementation violations of architectural intent
  - Architectural structure
  - Architectural types and styles
- Ideally:
  - Architects work at appropriate level of abstraction
  - Design is faithful abstraction of implementation

# Our Approach: Synchronize Abstract and Concrete C&C Views

- Abstract C&C view
  - Architect's design view
  - Problem-specific
  - May elide information
  - Example: Acme
- Concrete C&C view
  - Actual communication between implementation components
  - Example: ArchJava



# Relating Conceptual Views to Implementation-Level Views

- Match Architectural structure
  - Inserted, deleted, renamed, moved elements
  - Do not rely on unique identifiers
  - Do not require names to match
- Match Architectural types and styles
- Lightweight, scalable, semi-automated, incremental

# Bridging the Gap

- Matching Types (and Styles)
- Matching Structure

# Matching Type Structures between Abstract and Concrete C&C Views

## Acme Types

- Predicate-based type system
- Types = abstract logical predicates
- Architectural Style
  - Constraints (invariants or heuristics)
- Interfaces optional
  - Properties on ports

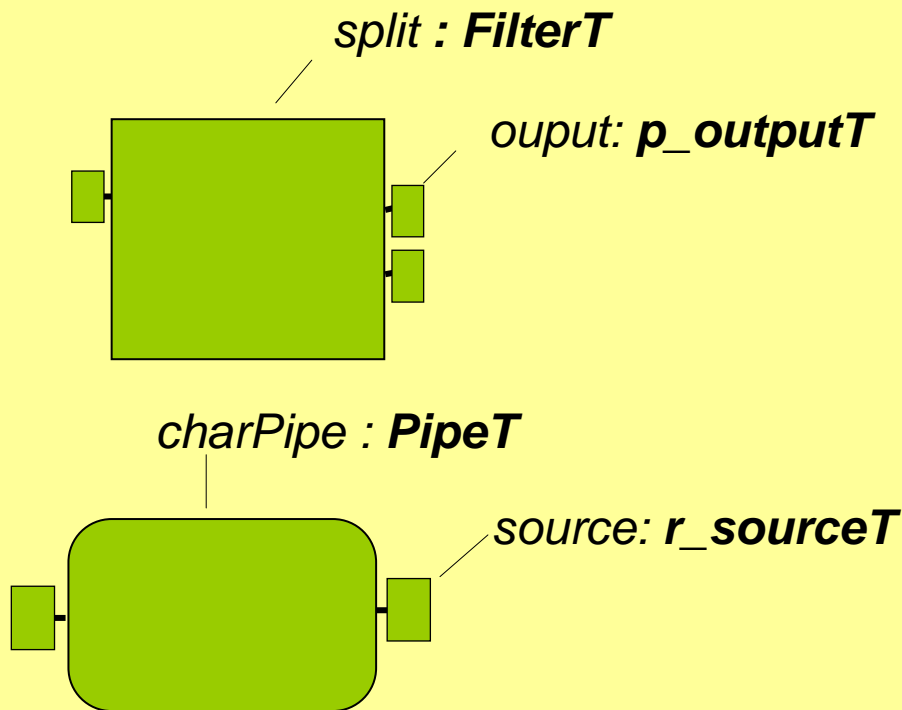
## ArchJava Types

- Conventional type system
- Types = concrete interfaces
  - provided and required functionality
- Some types not first-class
  - Port types, role types..

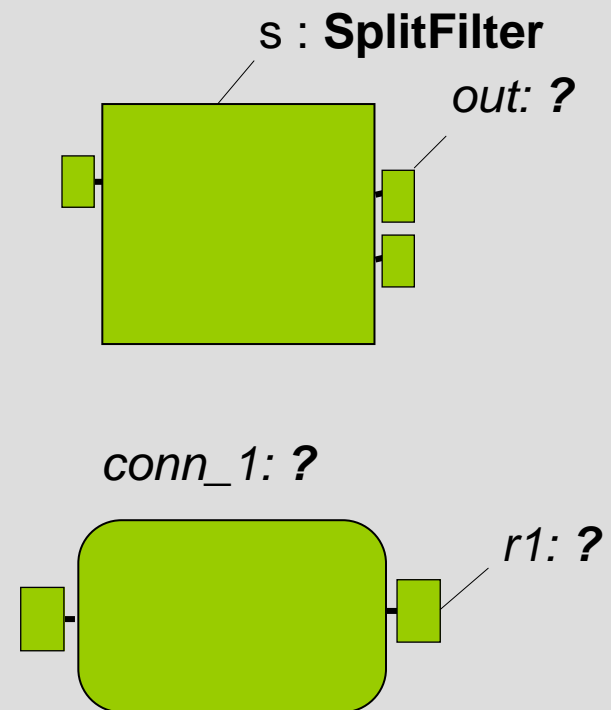
# Matching Type Structures

## Abstract C&C View

system: *PipeAndFilterStyle*



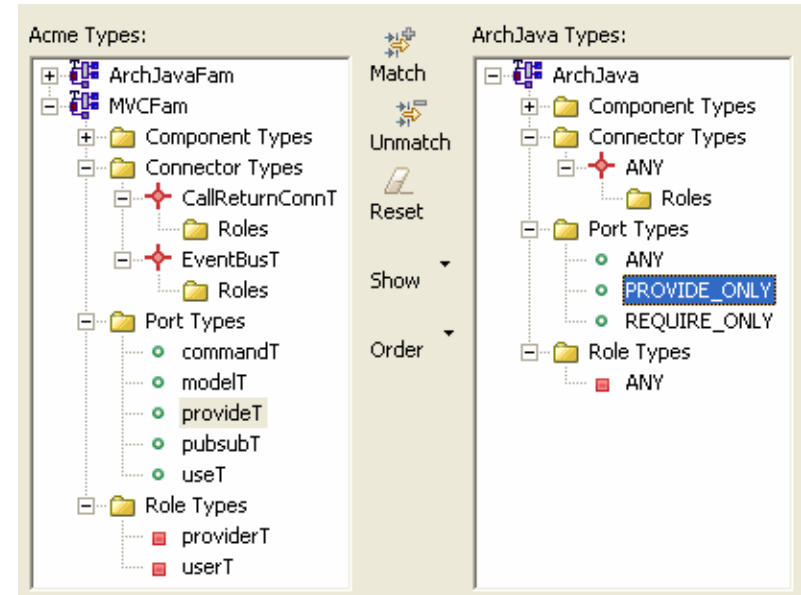
## Concrete C&C View



1. First-class types missing in ArchJava for connectors, ports, roles
2. Acme types at higher level of abstraction

# Matching Styles and Types

- Match explicit types if available
- Assign types to instances when no explicit type
- Special wildcards
- Infer types when possible
  - Using style information



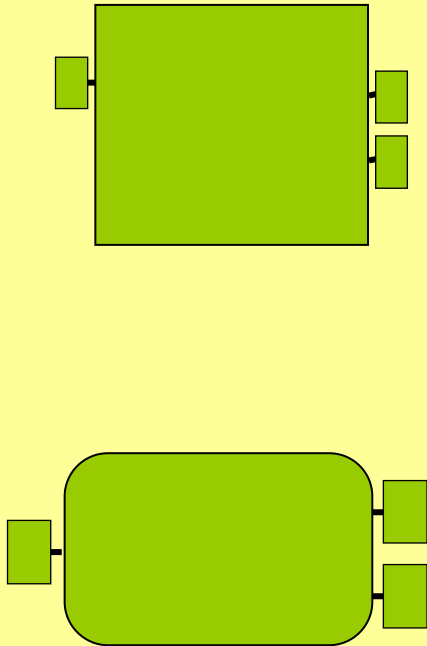
# Structural Differences

- Incidental renames
- Independent evolution
  - May forget to update other representation
- Design & Implementation
  - Different structures may be appropriate
    - E.g. hide representation inside a new component
- Types of differences
  - Renames
  - Inserts
  - Deletes
  - Moves
- Detection important for maintaining design properties

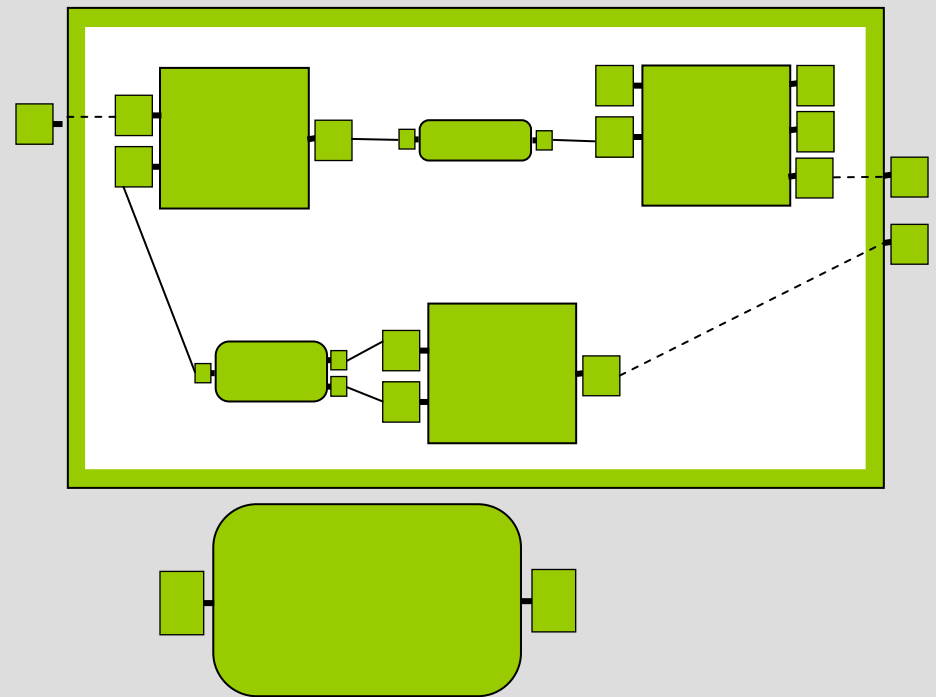
***Strategy: Automated detection of differences***

# Insert/Delete Differences

## Abstract C&C View

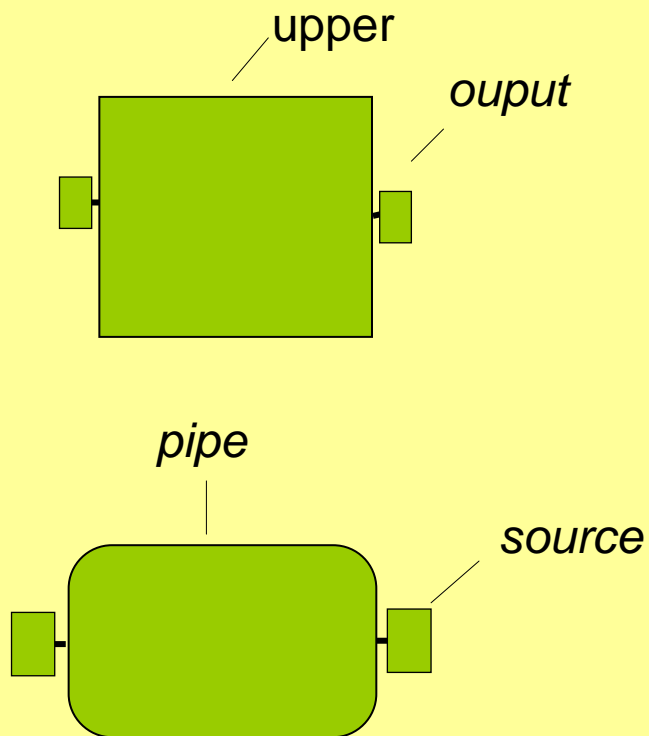


## Concrete C&C View

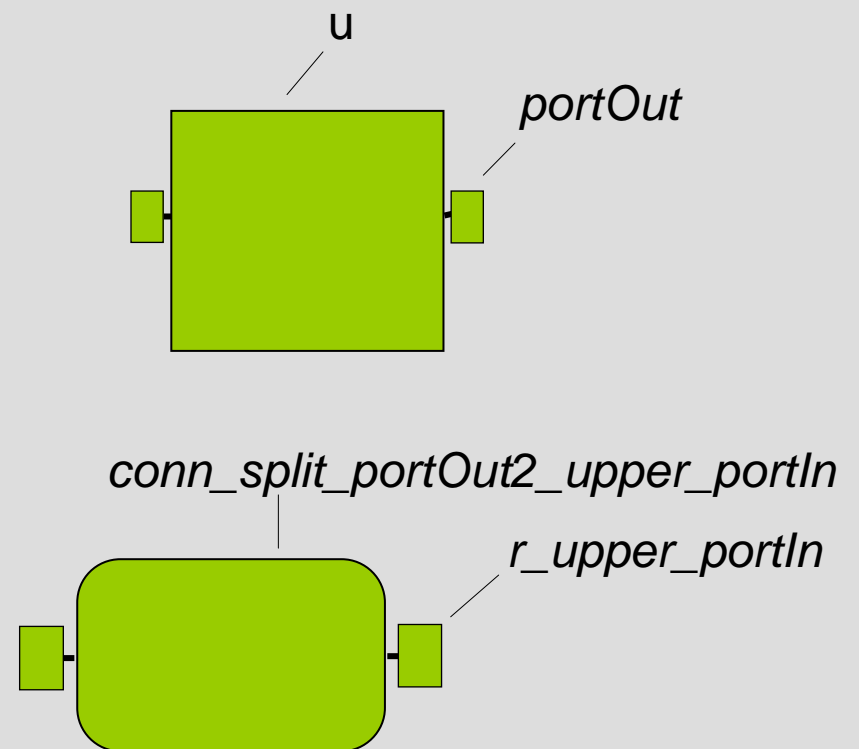


# Naming Differences

## Abstract C&C View

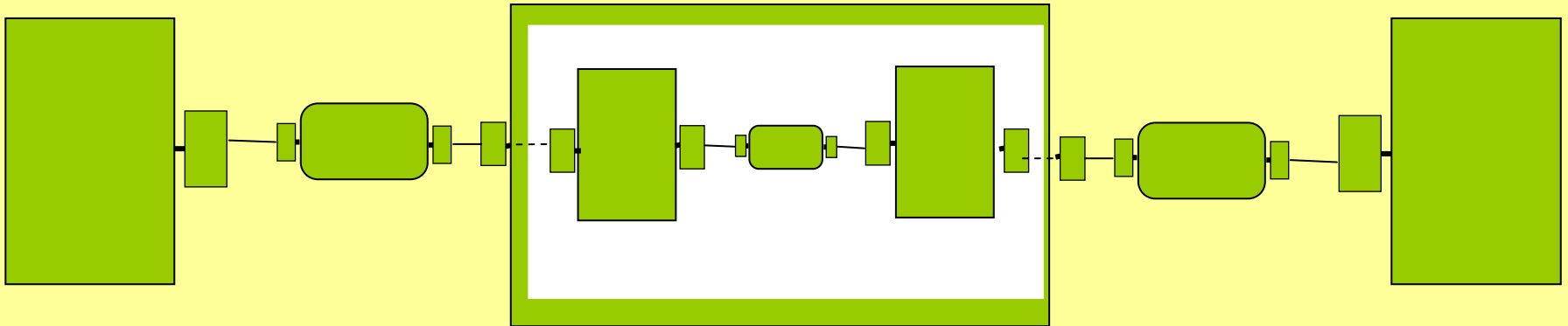


## Concrete C&C View

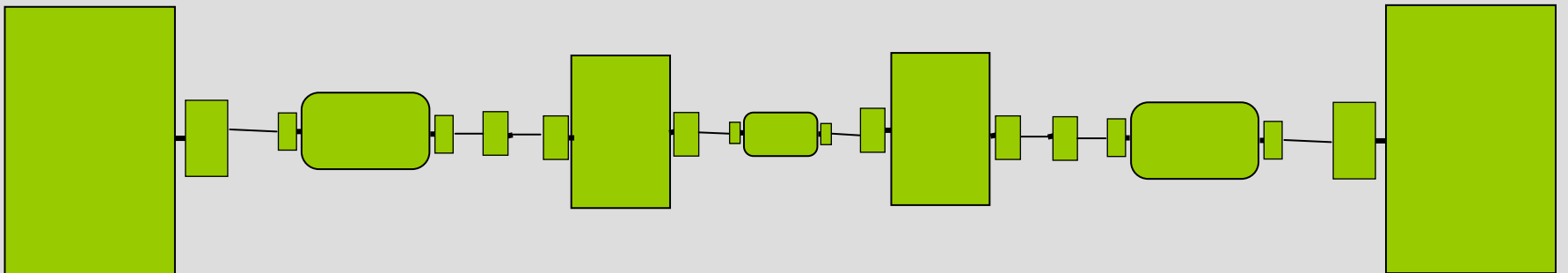


# Move Differences






## Abstract C&C View

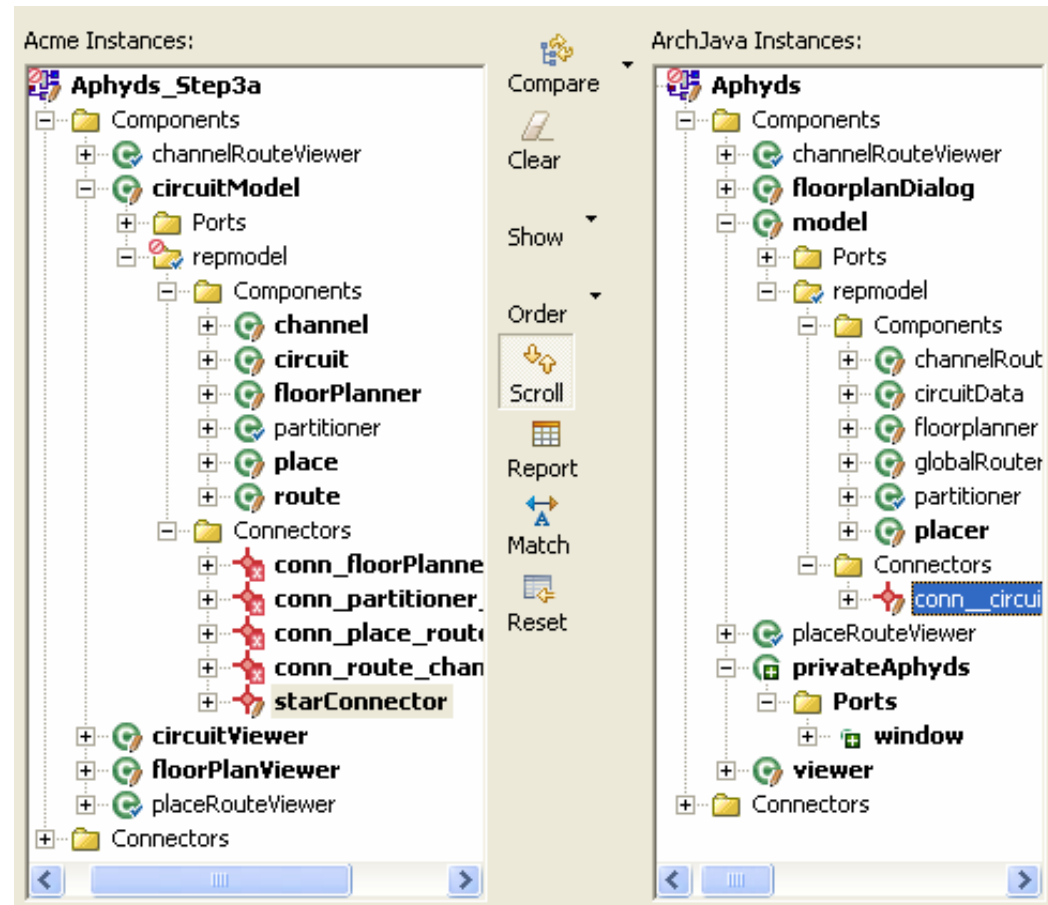


## Concrete C&C View



# Matching Architectural Structure

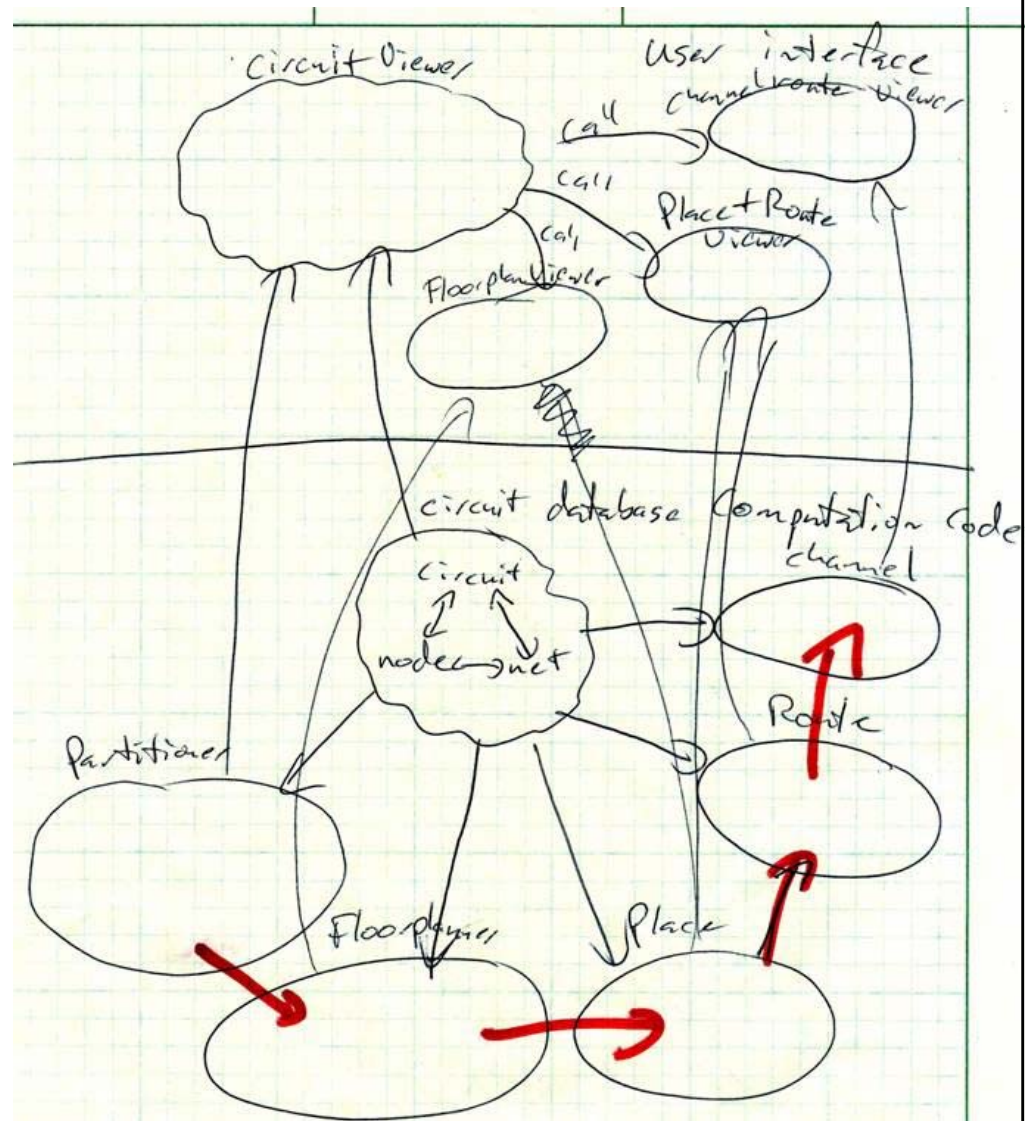
- Detect
  - Match 
  - Insert 
  - Delete 
  - Rename 
  - Move 



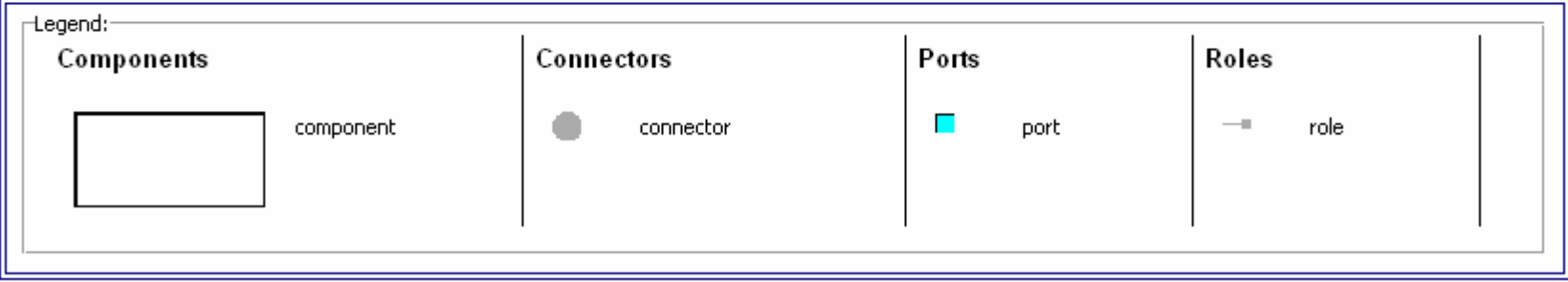
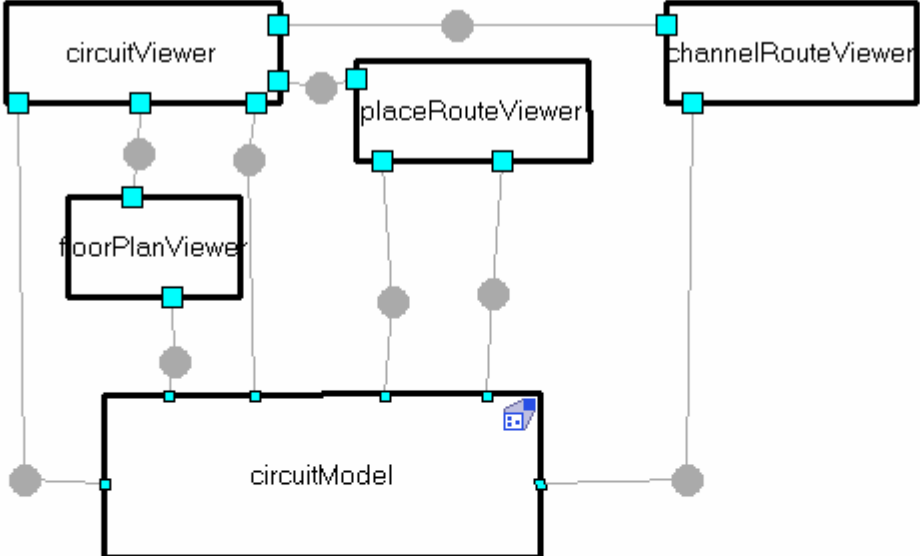
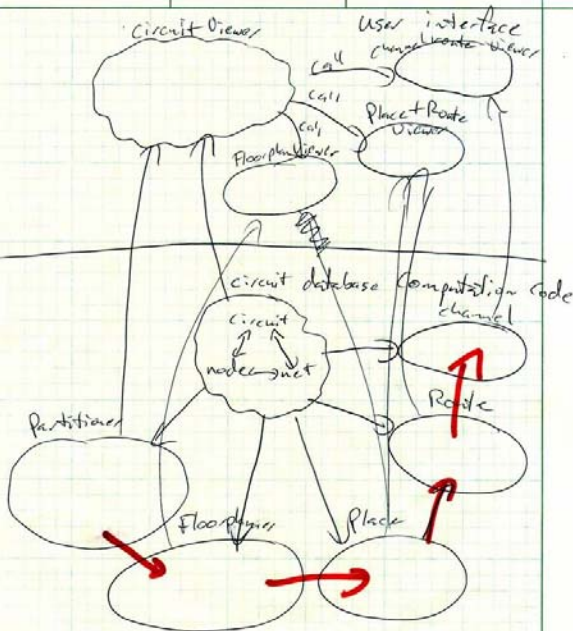
- Automated Tree-to-Tree Correction
  - Unordered attributed labeled trees

# Extended Example

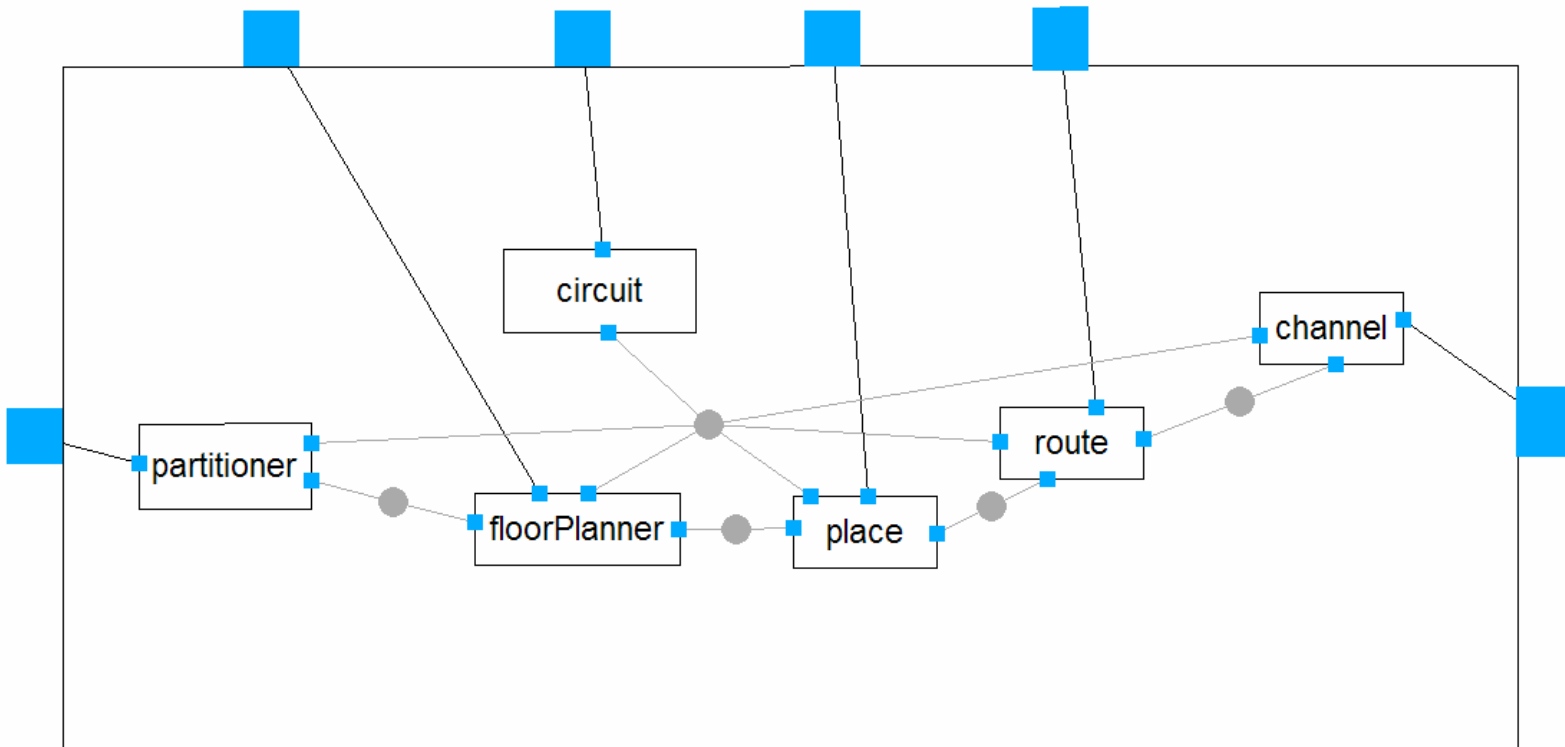
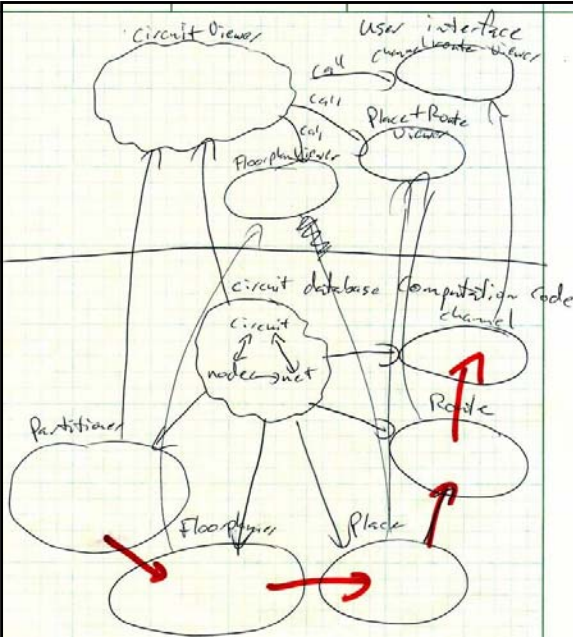
- ArchJava architecture consisting of
  - Over 20 components, 80 ports, several subsystems
- Re-engineered from Java application
  - Over 8 KSLOC
  - See [ACN02] for details



# Aphyds System

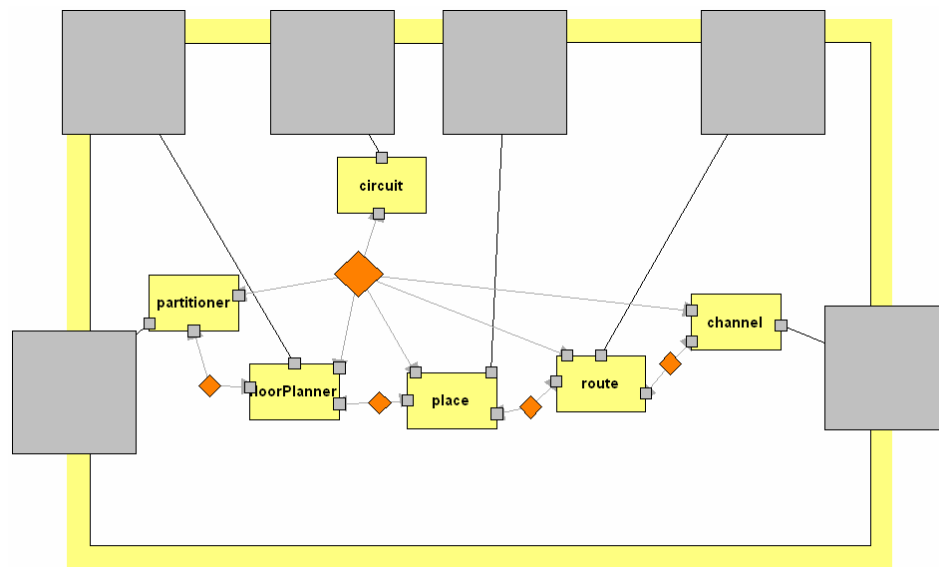


# circuitModel details



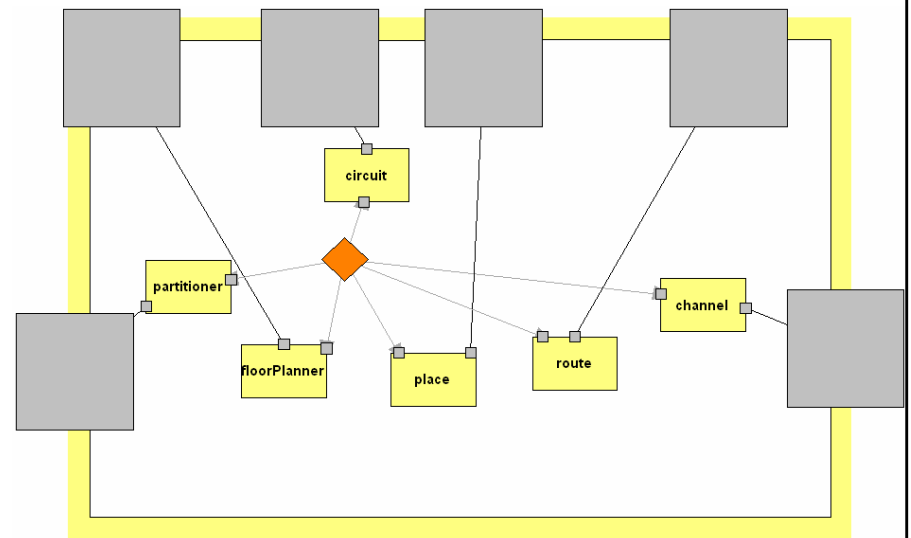
# First Divergence: Extra Connectors!

The “data flow” connectors in the original Architect’s model do *not* exist!



circuitModel  
Representation - repmode

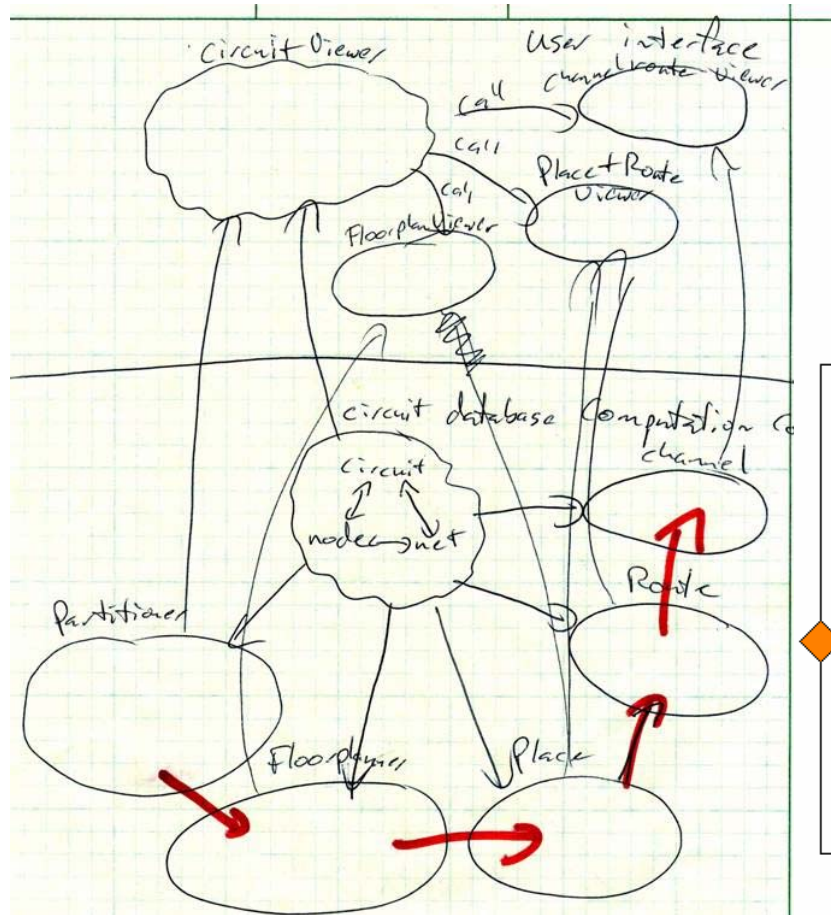
Before



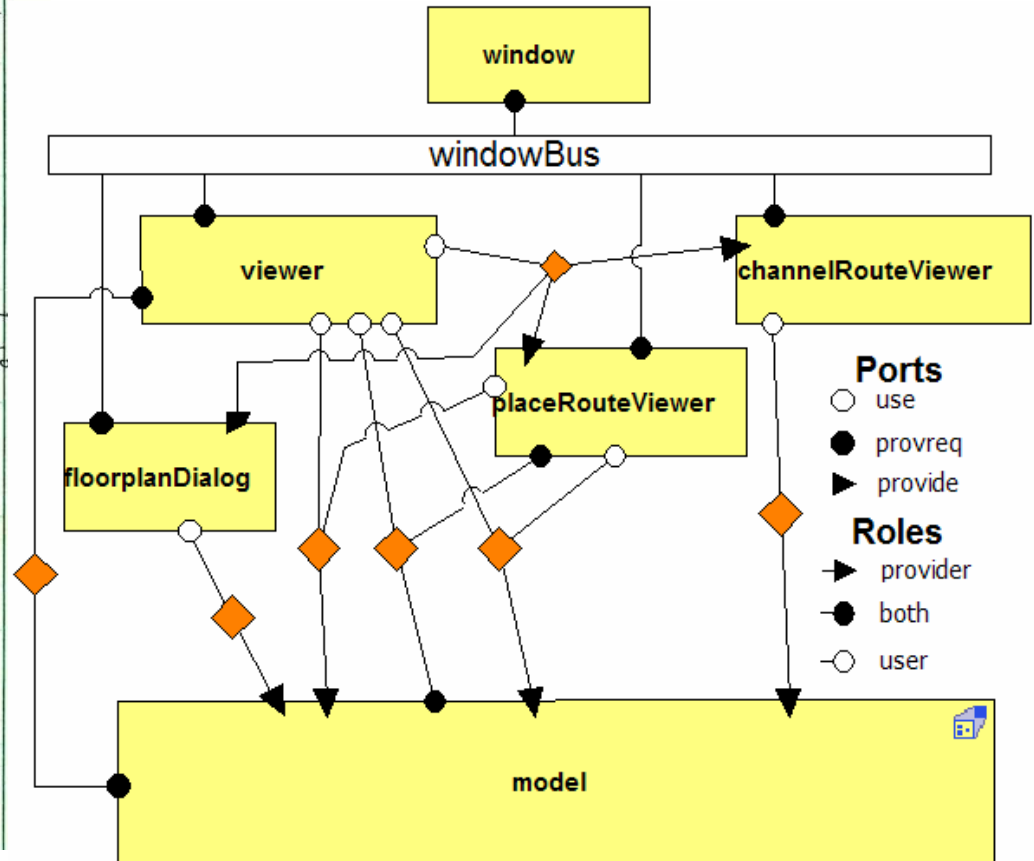
circuitModel  
Representation - repmode

After

# Many Other Divergences



Before



After

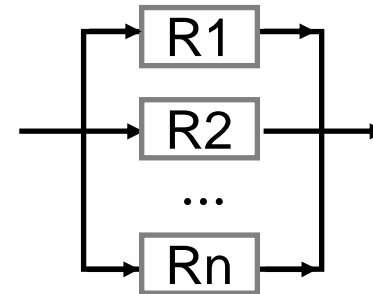
# Reliability Block Diagrams

- Determine aggregate reliability from the parts, for certain styles



$$R_{sys} = \prod_{i=1}^n R_i$$

Serial Composition



$$R_{sys} = 1 - \prod_{i=1}^n (1 - R_i)$$

Parallel Composition

Source: Abd-Allah, Ahmed, "Extending Reliability Block Diagrams to Software Architectures", USC Technical Report USC-CSE-97-501.

# Conclusions

- Our approach encourages continuous use of architectural views and analyses throughout the software life cycle
- Work at appropriate level of abstraction
  - Architectural styles, properties, analyses, ...
- Ensure that design is proper abstraction of implementation

# Questions?

# References

- Acme
  - <http://www.cs.cmu.edu/~acme>
- ArchJava
  - <http://archjava.fluid.cs.cmu.edu/>