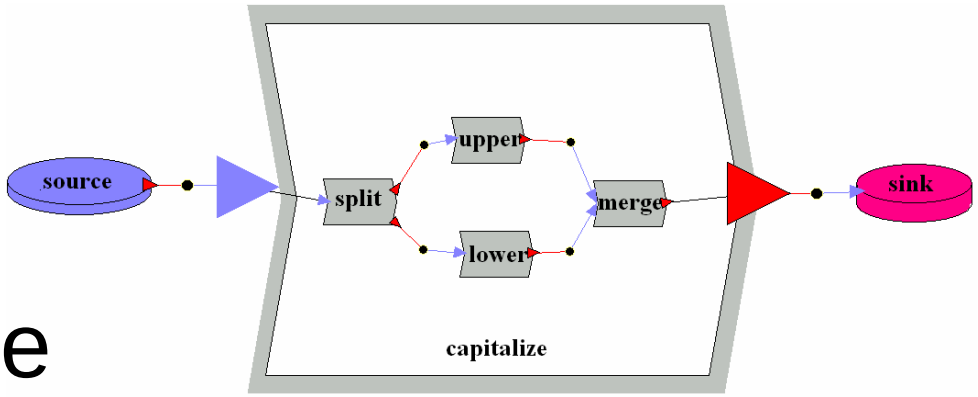
Modeling and Implementing Software Architecture with Acme and ArchJava

Marwan Abi-Antoun Jonathan Aldrich
David Garlan Bradley Schmerl
Nagi Nahas Tony Tseng

Institute for Software Research International
Carnegie Mellon University

Motivation



- Software Architecture
  - High level design of a software system
  - Components, connectors, and constraints on how they interact
- Key benefits
  - Program understanding
  - Software evolution
  - Analysis of quality attributes

Architecture Description Language: Acme

- Express a component & connector (C&C) architectural view
  - Define component types and properties
  - Check constraints of architectural style
  - Perform advanced analyses:
    - Performance analysis based on queuing theory
    - Rate monotonic schedulability analysis
- But, does ***not*** guarantee that implementation conforms to architecture

Architectural Conformance

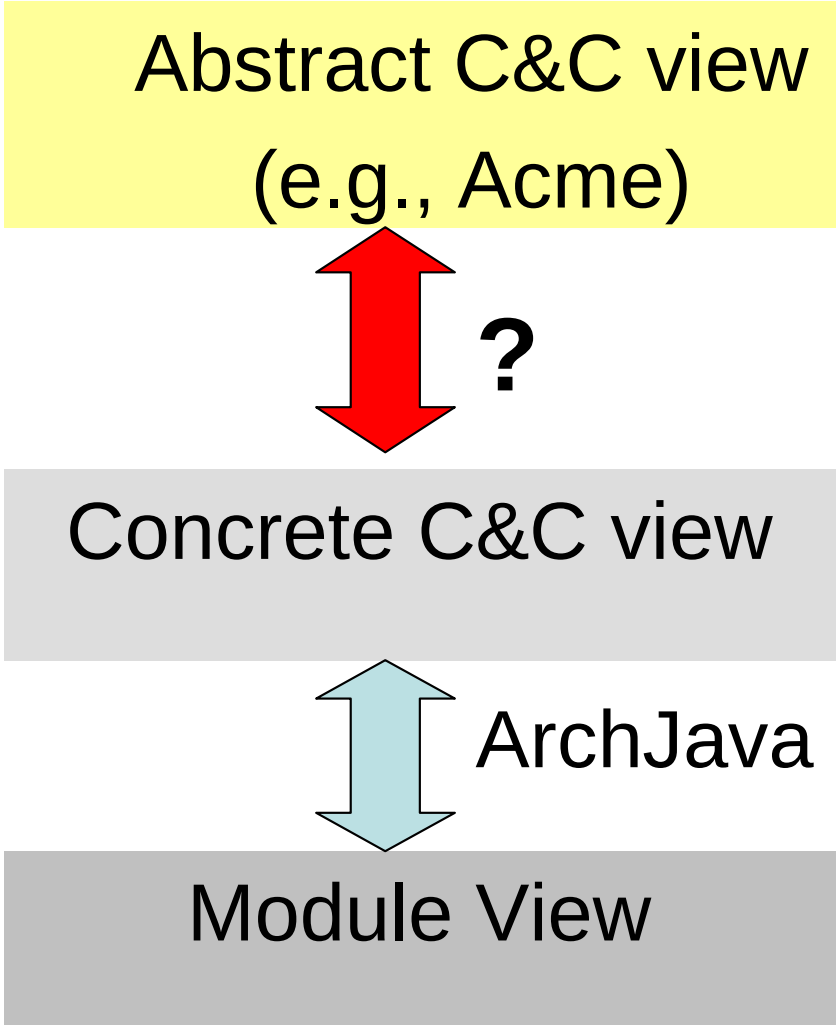
- Benefits of architecture contingent upon correct implementation
  - Program understanding
  - Software evolution
  - Checking architectural constraints
  - Analysis of quality attributes

Conformance with ArchJava

- Extension of Java programming language
- Specify architecture directly within code:
  - Components, Connectors, Ports...
  - Code = architecture specification
- Enforce structural conformance
- Does ***not*** enforce architectural properties
  - Style constraints, analysis of quality attributes…

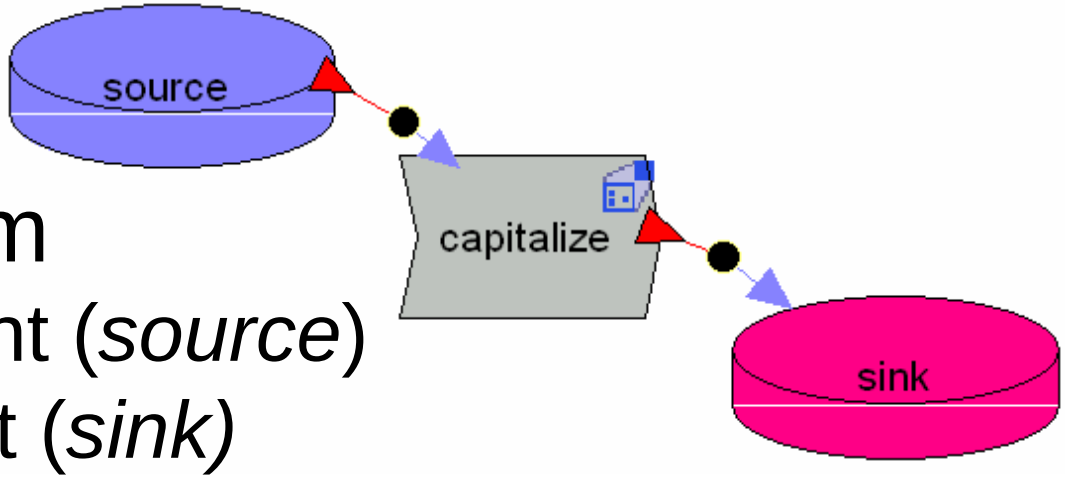
Abstract and Concrete C&C Views

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



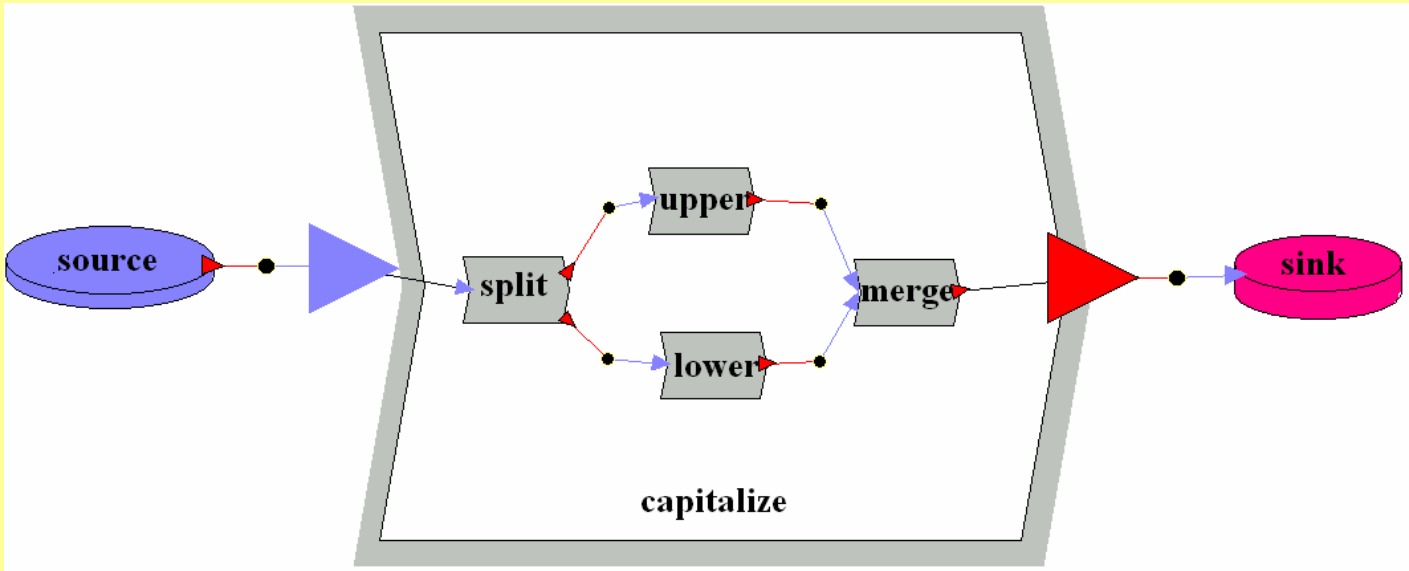
"*Never go to sea with two chronometers; take one or three*"

Running Example: *CaPiTaLiZe*



- Pipe-and-Filter System
  - data source component (*source*)
  - a data sink component (*sink)*
  - a filter component (*capitalize*)
  - two connectors (character pipes)
- Component *capitalize*
  - Receives ASCII characters from *source*
  - Converts characters alternatively to upper or lower case
  - Sends characters on to component *sink*
  - Further decomposed into a sub-architecture consisting of another pipe-and-filter system

Abstract C&C View



Concrete C&C View

```
public component class Capitalize {
    private final Upper upper = new Upper();
    private final Lower lower = new Lower();
    private final Split split = new Split();
    private final Merge merge = new Merge();
    public port portIn { requires char getChar(); }
    public port portOut { provides char getChar(); }
    connect lower.portOut, merge.portIn1;
    connect split.portOut2, lower.portIn;
    connect upper.portIn, split.portOut1;
    connect merge.portIn2, upper.portOut;
    glue portOut to merge.portOut;
    glue portIn to split.portIn;
}
```

ArchJava implementation of component *capitalize*

Our Approach

- Synchronize C&C views
- Incremental
  - Allow both views to evolve simultaneously
  - Do not require complete code re-generation
- Lightweight
  - Fits into one “wizard” dialog
- Semi-automated
  - The computer does the matching

Synchronization: 5 steps

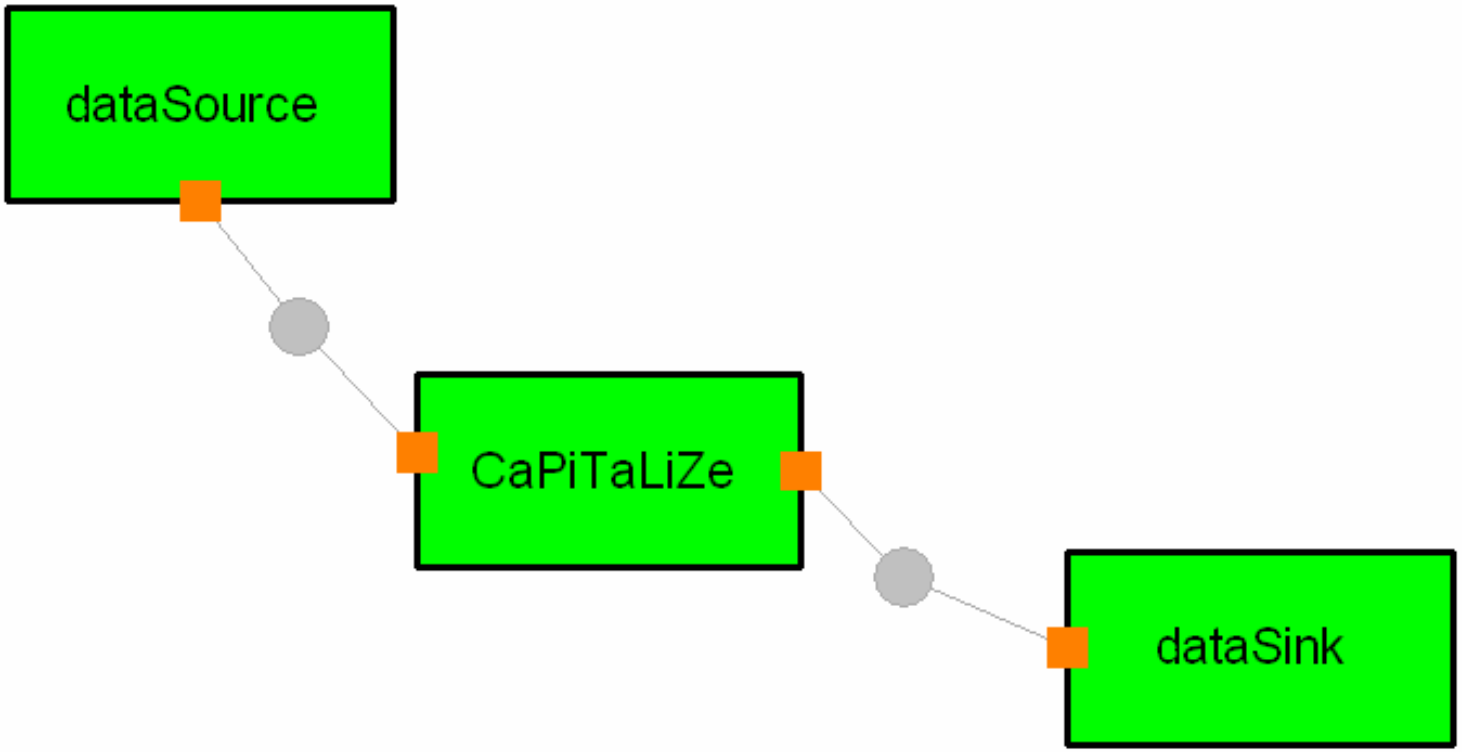
- Step 1: Setup synchronization
- Step 2: View & match types (optional)
- Step 3: View & match instances
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script (optional)

Demonstration Scenario

- Begin with Acme model
  - Partial Structure
  - Without styles or types
- Synchronize structure with implementation
- Assign styles
- Synchronize with architectural types
  - Match type structures
  - Enables Acme constraint checking

Initial Acme Model

- Structure only
- Not styles, no types



Synchronization: 5 steps

- **Step 1: Setup synchronization**
- Step 2: View & match types (skipped)
- Step 3: View & match instances
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script

Synchronization: 5 steps

- Step 1: Setup synchronization
- ➢ **Step 2: View & match types (skipped)**
- Step 3: View & match instances
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script

Synchronization: 5 steps

- Step 1: Setup synchronization
- Step 2: View & match types (skipped)
- ➢ **Step 3: View & match instances**
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script

Step 3: View & match instances

Synchronize Acme and ArchJava

Step 3 of 5: Match Acme and ArchJava Instances

View differences between Acme and ArchJava instances.

Acme Instances:

- System1
  - Components
    - CaPiTaLiZe
      - Ports
        - p
        - p0
    - dataSink
      - Ports
        - p
    - dataSource
      - Ports
        - p
  - Connectors
    - conn
      - Roles
        - r
        - r0
    - conn0
      - Roles
        - r
        - r0

Compare
Clear
Show
Order
Scroll
Report
Match
Reset

ArchJava Instances:

- Main
  - Components
    - capitalize
      - Ports
        - portIn
        - portOut
      - repcapitalize
        - Components
          - b
          - l
          - m
          - s
          - u
        - Connectors
    - sink
      - Ports
        - portIn
    - source
      - Ports
        - portOut
  - Connectors

Properties | Messages | Search

Acme Properties:

Name	Value

ArchJava Properties:

Name	Value
ArchJavaType	StreamSink

< Back | Next > | Finish | Cancel

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

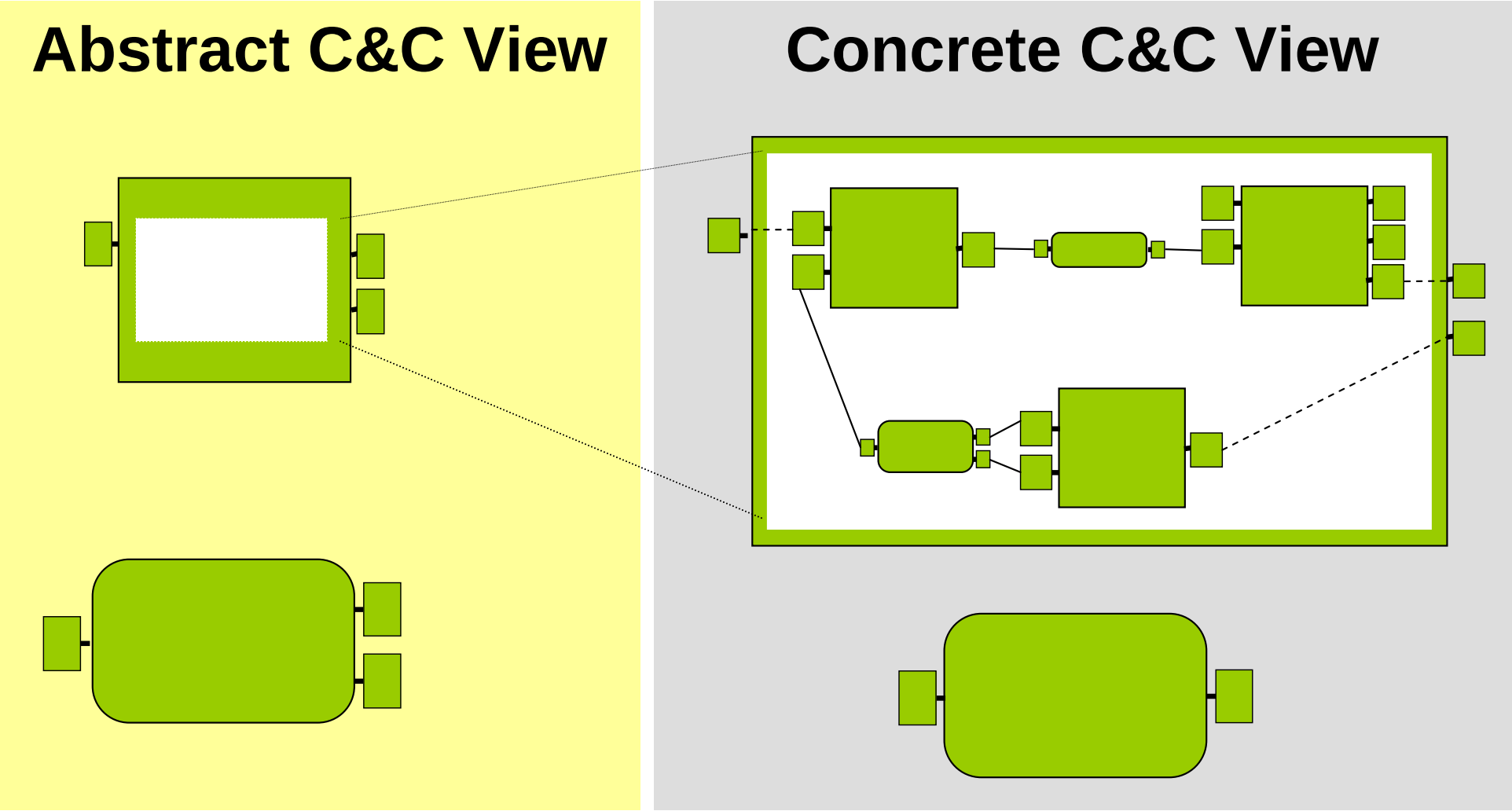
- Types of differences
  - Renames
  - Inserts
  - Deletes
  - Moves
- Detection important for maintaining design properties

Strategy: Automated detection of differences

Structural Comparison

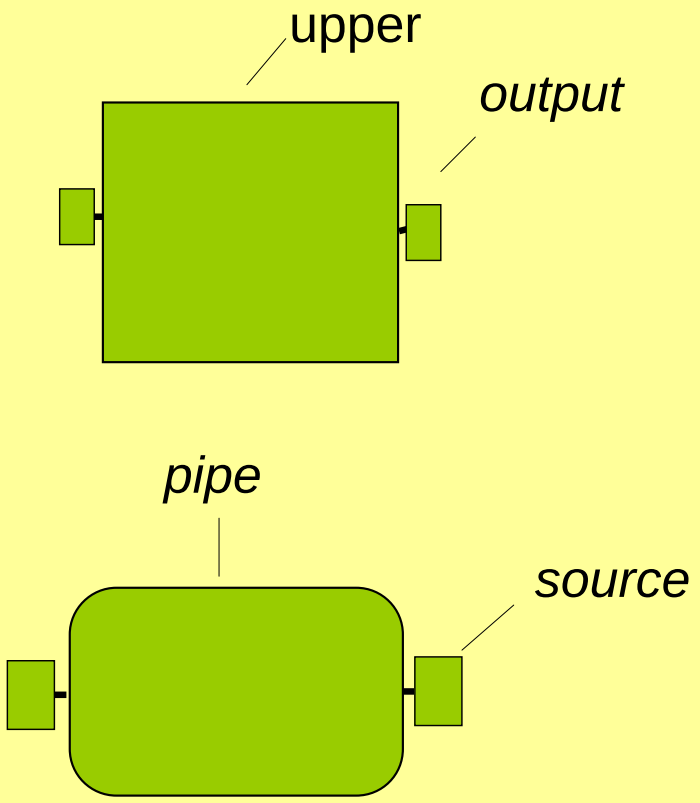
- Reduce to *tree-to-tree correction* problem
- Unordered Labeled Trees
  - General problem proved MAX SNP-HARD
  - Assumptions can produce polynomial time
    - Torsello, A., Hidovic-Rowe, D. and Pelillo, M. Polynomial-Time Metrics for Attributed Trees. *To appear* in IEEE Transaction on Pattern Analysis and Machine Intelligence, 27 (7), 2005.
- We also designed a novel algorithm
  - Initially intended to detect *Moves*
  - Can also detect *Inserts* better!

Insert/Delete Differences

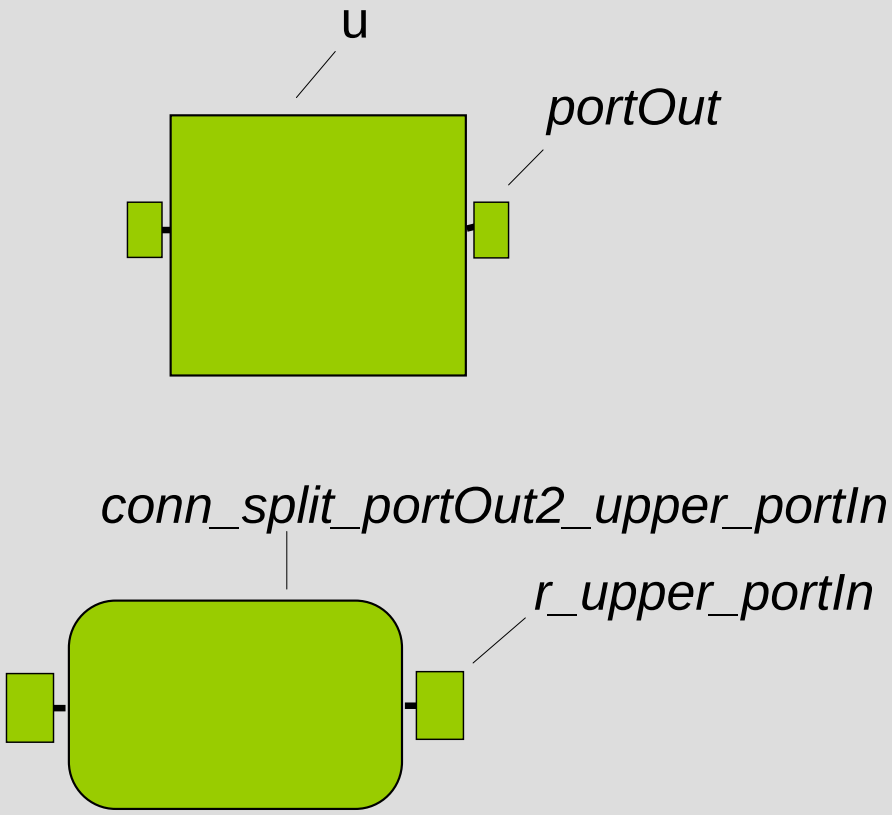


Naming Differences

Abstract C&C View

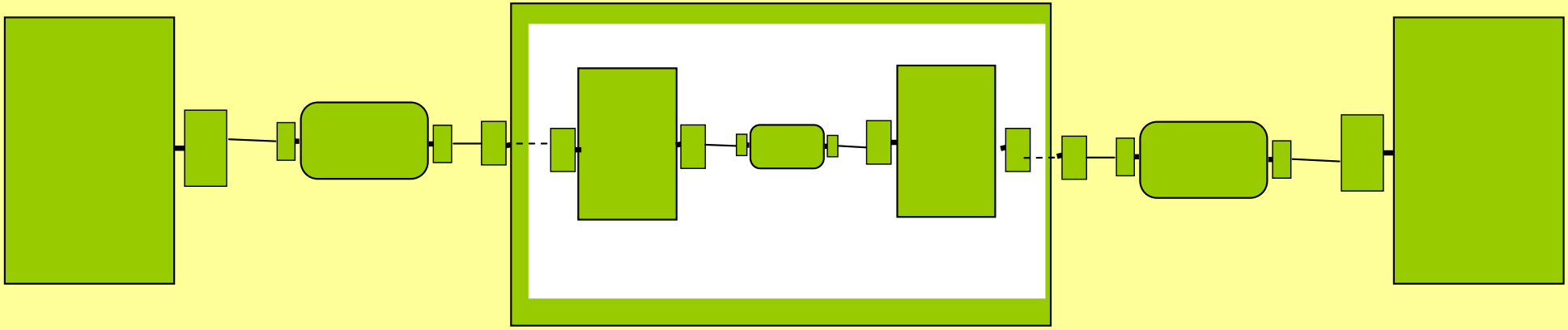


Concrete C&C View

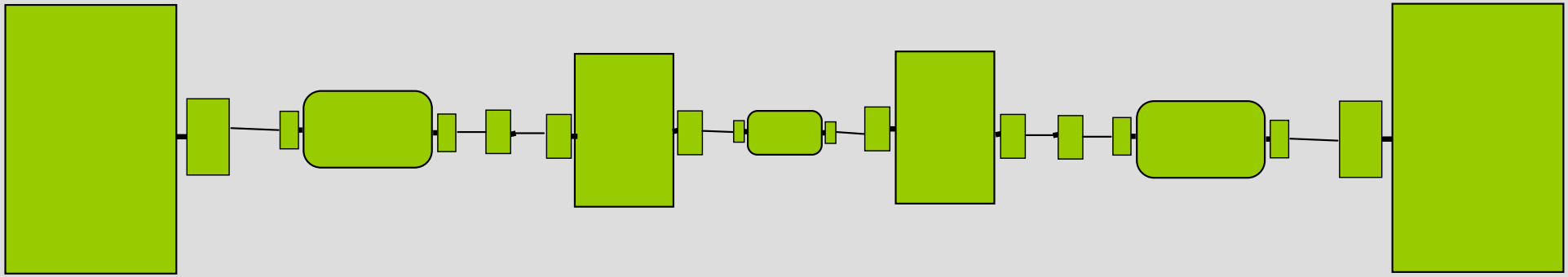


Move Differences

Abstract C&C View

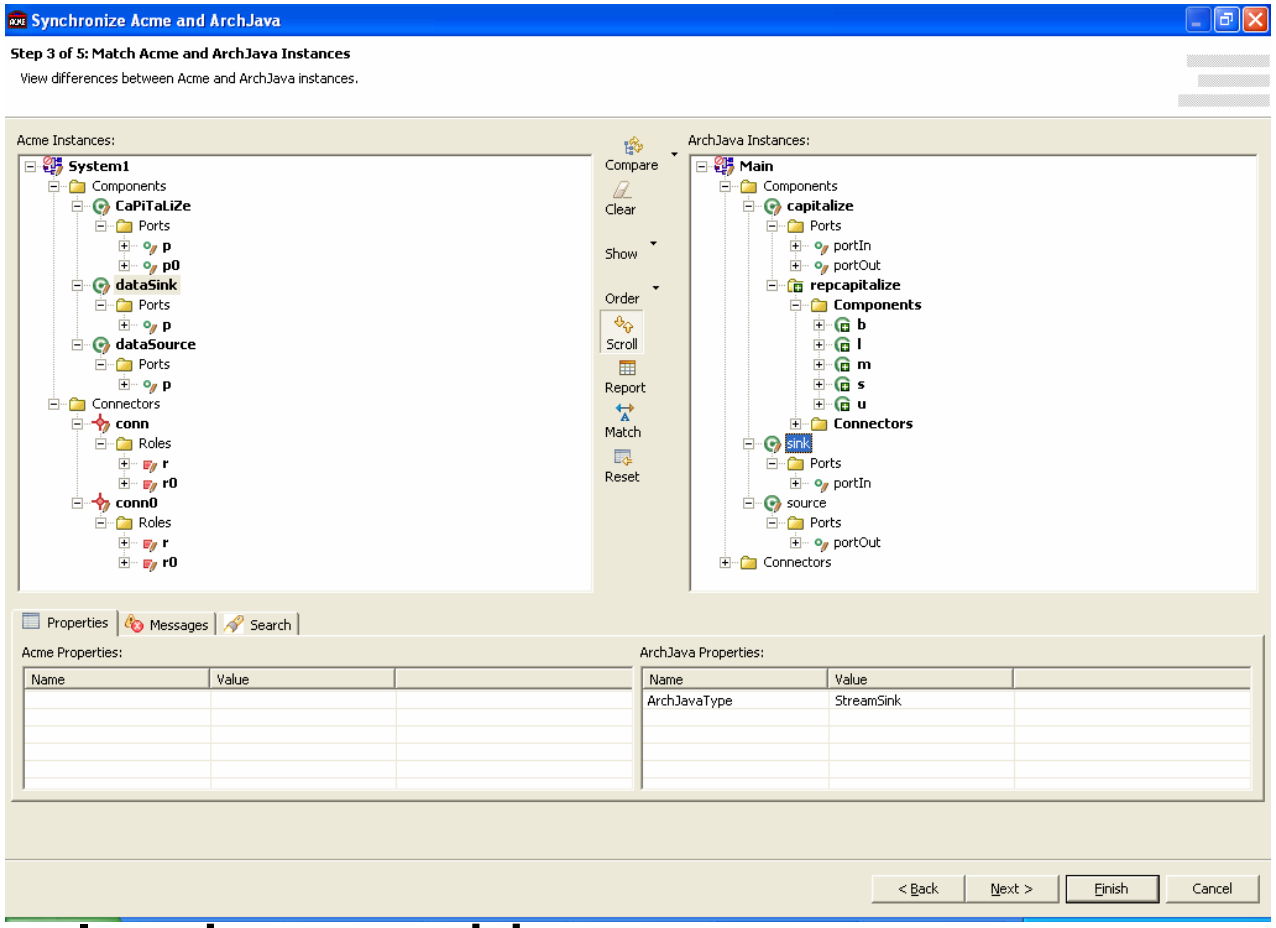


Concrete C&C View



Step 3: Details

- Tree-to-tree correction markers
  - Match
  - Insert
  - Delete
  - Rename
  - Move



- Matching
  - Shown with selection tracking

Step 3: Advanced Features

- Direct manipulation
  - manually insert, delete or rename elements
  - generates corresponding edit actions
- Elision
  - selectively hide (and unhide) elements
  - only exclude from comparison
  - instance- or type-based (e.g. “all connectors”)
- Forced matches
  - force a match between two elements
- Manual overrides
  - override or cancel edit action

Synchronization: 5 steps

- Step 1: Setup synchronization
- Step 2: View & match types (skipped)
- Step 3: View & match instances
- ➢ **Step 4: View & modify edit script**
- Step 5: Confirm & apply edit script

Step 4: View & modify edit script

Synchronize Acme and ArchJava

Step 4 of 5: (Optional) View and Modify the Edit Script

Please check the Messages tab: 0 error(s), 45 warning(s) found.

Acme Instances:

- System
  - Components
    - capitalize (CaPiTaLiZe)
      - Ports
      - repcapitalize
        - Components
          - b
            - Ports
              - portIn
              - portOut
          - l
            - Ports
              - portIn
              - portOut
          - m
          - s
          - u
        - Connectors
    - sink (dataSink)
    - source (dataSource)
  - Connectors
    - conn__capitalize_portOut__sink_portIn (conn0)
    - conn__source_portOut__capitalize_portIn (conn)

Select All
Check
Auto-Correct
Show
Order

Show Preview

Properties | Messages

Element Type	Element Name	Description

< Back | Next > | Finish | Cancel

Step 4: Details

- View merged model in Common supertree
- Modify type assignments
  - Assign/un-assign types
  - Override inferred types
- Cancel unwanted edit actions
- Check edit script for common problems
  - No assigned types
  - Element name using reserved Acme keyword

Synchronization: 5 steps

- Step 1: Setup synchronization
- Step 2: View & match types (optional)
- Step 3: View & match instances
- Step 4: View & modify edit script
- ➢ **Step 5: Confirm & apply edit script**

Step 5: Confirm & apply edit script

Synchronize Acme and ArchJava

Step 5 of 5: Confirm and Apply the Edit Script

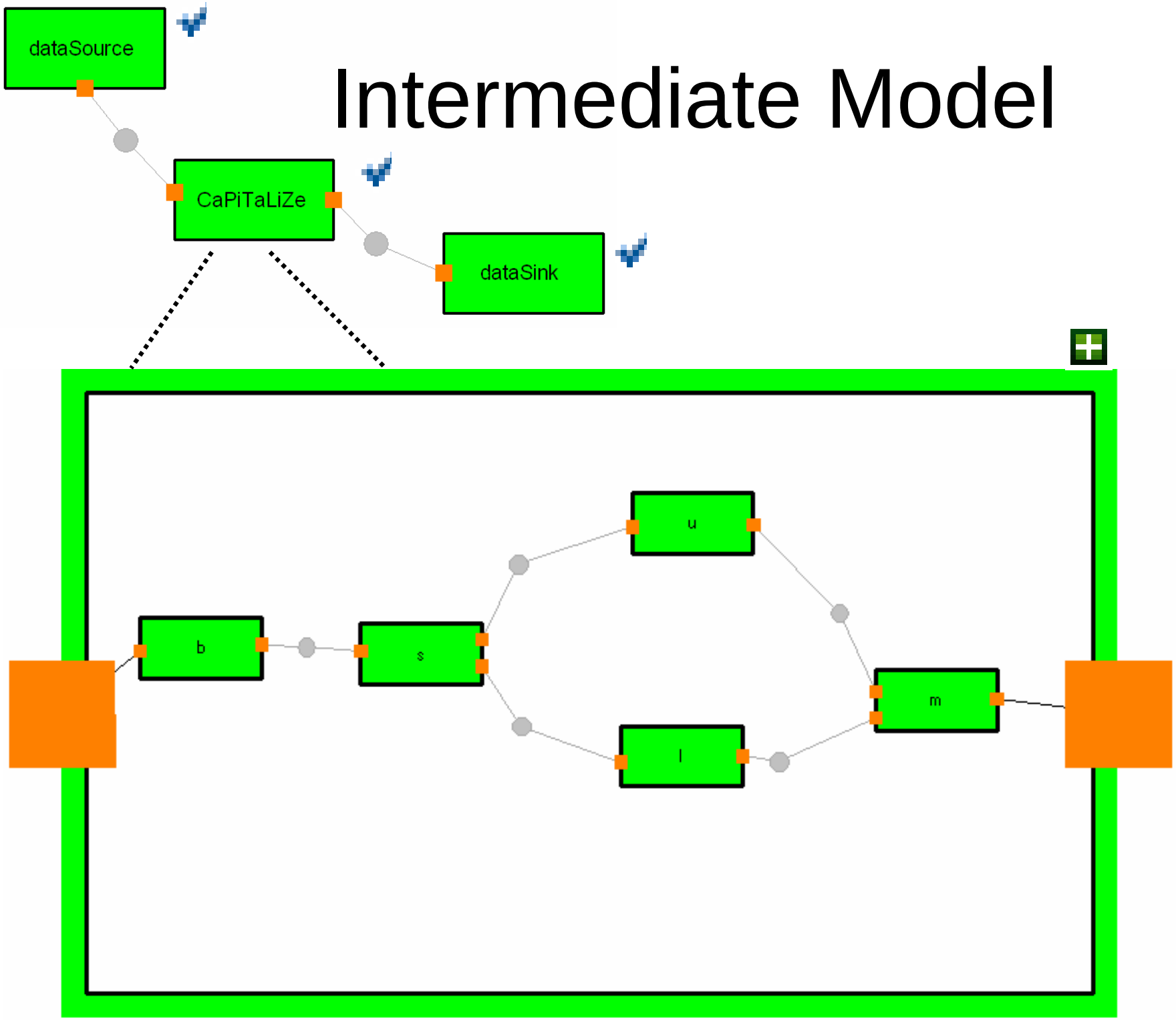
Confirm and apply the edit actions on the Acme system.

Acme Edit Actions:

```
Rename AcmeComponent : Current Name= 'CaPiTaLiZe' -> New Name = 'capitalize'.
Rename AcmeComponent : Current Name= 'dataSink' -> New Name = 'sink'.
Rename AcmePort : Current Name= 'p' -> New Name = 'portIn'.
Rename AcmeComponent : Current Name= 'dataSource' -> New Name = 'source'.
Rename AcmePort : Current Name= 'p' -> New Name = 'portOut'.
Rename AcmeConnector : Current Name= 'conn0' -> New Name = 'conn__capitalize_portOut__sink_portIn'.
Rename AcmeRole : Current Name= 'r' -> New Name = 'role__capitalize_portOut'.
Rename AcmeRole : Current Name= 'r0' -> New Name = 'role__sink_portIn'.
Rename AcmeConnector : Current Name= 'conn' -> New Name = 'conn__source_portOut__capitalize_portIn'.
Rename AcmeRole : Current Name= 'r0' -> New Name = 'role__capitalize_portIn'.
Rename AcmeRole : Current Name= 'r' -> New Name = 'role__source_portOut'.
Create Representation 'repcapitalize' on Component 'capitalize'.
Create Component 'b'  in System [\repcapitalize\] with Types [].
Create Port 'portIn' on Component 'b'  in System [\repcapitalize\] with Types [].
Create Port 'portOut' on Component 'b'  in System [\repcapitalize\] with Types [].
Create Component 'l'  in System [\repcapitalize\] with Types [].
Create Port 'portIn' on Component 'l'  in System [\repcapitalize\] with Types [].
Create Port 'portOut' on Component 'l'  in System [\repcapitalize\] with Types [].
Create Component 'm'  in System [\repcapitalize\] with Types [].
Create Port 'portIn1' on Component 'm'  in System [\repcapitalize\] with Types [].
Create Port 'portIn2' on Component 'm'  in System [\repcapitalize\] with Types [].
Create Port 'portOut' on Component 'm'  in System [\repcapitalize\] with Types [].
Create Component 's'  in System [\repcapitalize\] with Types [].
Create Port 'portIn' on Component 's'  in System [\repcapitalize\] with Types [].
Create Port 'portOut1' on Component 's'  in System [\repcapitalize\] with Types [].
Create Port 'portOut2' on Component 's'  in System [\repcapitalize\] with Types [].
Create Component 'u'  in System [\repcapitalize\] with Types [].
Create Port 'portIn' on Component 'u'  in System [\repcapitalize\] with Types [].
Create Port 'portOut' on Component 'u'  in System [\repcapitalize\] with Types [].
Create Connector 'conn__b_portOut__s_portIn'  in System [\repcapitalize\] with Types [].
Create Role 'role__b_portOut' on Connector 'conn__b_portOut__s_portIn'  in System [\repcapitalize\] with Types [].
Create Role 'role__s_portIn' on Connector 'conn__b_portOut__s_portIn'  in System [\repcapitalize\] with Types [].
Create Connector 'conn__l_portOut__m_portIn2'  in System [\repcapitalize\] with Types [].
Create Role 'role__l_portOut' on Connector 'conn__l_portOut__m_portIn2'  in System [\repcapitalize\] with Types [].
Create Role 'role__m_portIn2' on Connector 'conn__l_portOut__m_portIn2'  in System [\repcapitalize\] with Types [].
Create Connector 'conn__s_portOut1__u_portIn'  in System [\repcapitalize\] with Types [].
Create Role 'role__s_portOut1' on Connector 'conn__s_portOut1__u_portIn'  in System [\repcapitalize\] with Types [].
Create Role 'role__u_portIn' on Connector 'conn__s_portOut1__u_portIn'  in System [\repcapitalize\] with Types [].
Create Connector 'conn__s_portOut2__l_portIn'  in System [\repcapitalize\] with Types [].
Create Role 'role__l_portIn' on Connector 'conn__s_portOut2__l_portIn'  in System [\repcapitalize\] with Types [].
```

< Back | Next > | Finish | Cancel

Intermediate Model



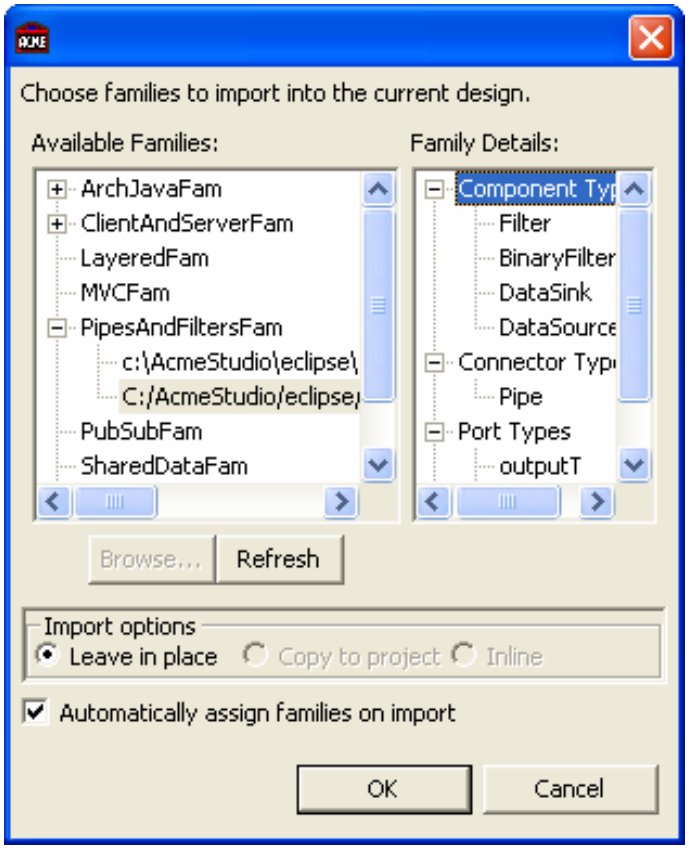
Step 2: View & match types

- Step 1: Setup synchronization
- ➢ **Step 2: View & match types**
- ➢ Step 3: View & match instances
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script

Assign Family or Style to System

- Architectural style:
  - Set of types for components, connectors, roles, ports, and properties
  - Set of rules that govern how elements of those types may be used
- Acme system in particular style:
  - Elements use types defined by that style
  - System satisfies rules of that style

Assign Style



Step 2: View & match types

Synchronize Acme and ArchJava

Step 2 of 5: (Optional) Match Acme and ArchJava Types

Optionally view and match Acme types and ArchJava types

Acme Types:

- PipesAndFiltersFam
  - Component Types
    - BinaryFilter
    - DataSink
    - DataSource
    - Filter
  - Connector Types
    - Pipe
  - Port Types
    - inputT
    - outputT
  - Role Types

Match
Unmatch
Reset
Show
Order
Check

ArchJava Types:

- ArchJava
  - Component Types
    - ANY
    - Capitalize
    - CharBuffer
    - Lower
    - Main
    - Merge
    - Split
    - StreamSink
    - StreamSource
    - Upper
  - Connector Types
    - ANY
      - Roles
  - Port Types
    - ANY
    - PROVIDE_AND_REQUIRE
    - PROVIDE_ONLY
    - REQUIRE_ONLY
  - Role Types

Properties | Messages

Acme Properties:

Name	Value

ArchJava Properties:

Name	Value
AcmeFamilyType	PipesAndFiltersFam.inputT

< Back | Next > | Finish | Cancel

Matching Type Structures between Abstract and Concrete C&C Views

Acme Types

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

ArchJava Types

- Conventional type system
- ArchJava Types
  - Interface of provided and required functionality
- Some types not first-class
  - Port types, role types..

Matching Type Structures

Abstract C&C View

system: ***PipeAndFilterStyle***

split **: *FilterT***

output: ***p_outputT***

charPipe : ***PipeT***

source: ***r_sourceT***

Concrete C&C View

s : **SplitFilter**

out: ***?***

conn_1: ***?***

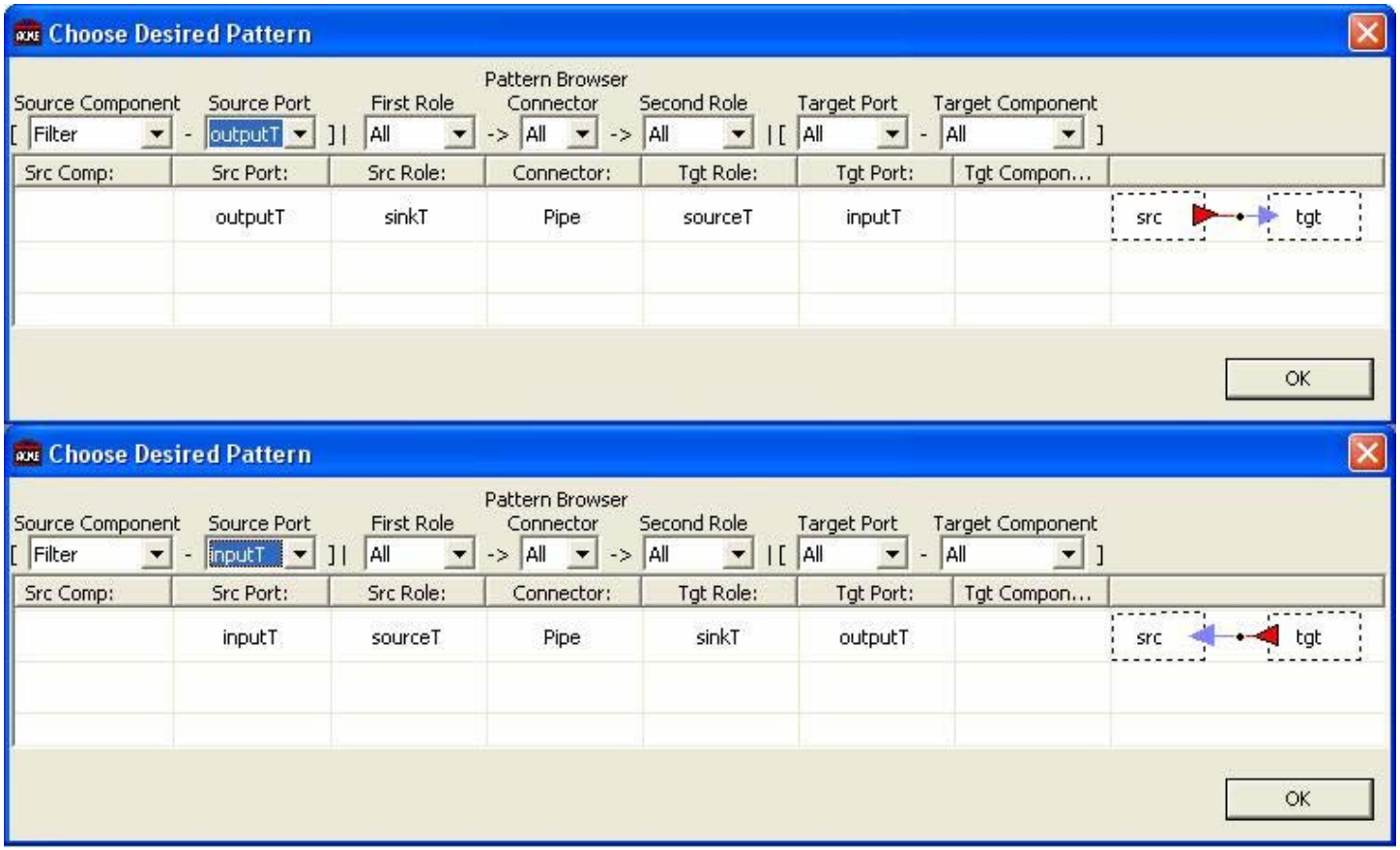
r1: ***?***

Step 2: Details

- Match explicit types when possible
  - Acme Component Type ↔ ArchJava Component Type
- Assign types to instances when no explicit type
  - Acme Port Type ↔ ArchJava Port instances
- Special wildcards (type *ANY*)
  - Acme Connector Type ↔ ArchJava Connector instances
  - Acme Port Type ↔ ArchJava Port instances with only provided methods, only required methods, PROVIDE_ONLY, REQUIRE_ONLY, …
- Infer types when possible
  - Role type based on style specific “connection patterns”

Style “Connection Patterns”

- Infer type of connector role (e.g., *sourceT*) based on source component type (e.g., *Filter*), source port type (e.g., *inputT*), and connector type (e.g., *Pipe*)



Step 3: View & match instances

- Step 1: Setup synchronization
- Step 2: View & match types
- ➢ **Step 3: View & match instances**
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script

Step 3: View & match instances

Synchronize Acme and ArchJava

Step 3 of 5: Match Acme and ArchJava Instances

View differences between Acme and ArchJava instances.

Acme Instances:

- System1
  - Components
    - capitalize
      - Ports
        - p
        - p0
      - repcapitalize
        - Components
          - b
          - l
          - m
          - s
          - u
        - Connectors
          - conn__b_portOut__s_portIn
          - conn__l_portOut__m_portIn2
          - conn__s_portOut1__u_portIn
          - conn__s_portOut2__l_portIn
          - conn__u_portOut__m_portIn1
    - sink
    - source

Compare
Clear
Show
Order
Scroll
Report
Match
Reset

ArchJava Instances:

- Main
  - Components
    - capitalize
      - Ports
        - portIn
        - portOut
      - repcapitalize
        - Components
          - b
          - l
          - m
          - s
          - u
        - Connectors
    - sink
    - source
  - Connectors
    - conn__capitalize_portOut__sink_portIn
      - Roles
        - role__capitalize_portOut
        - role__sink_portIn

Properties | Messages | Search

Acme Properties:

Name	Value

ArchJava Properties:

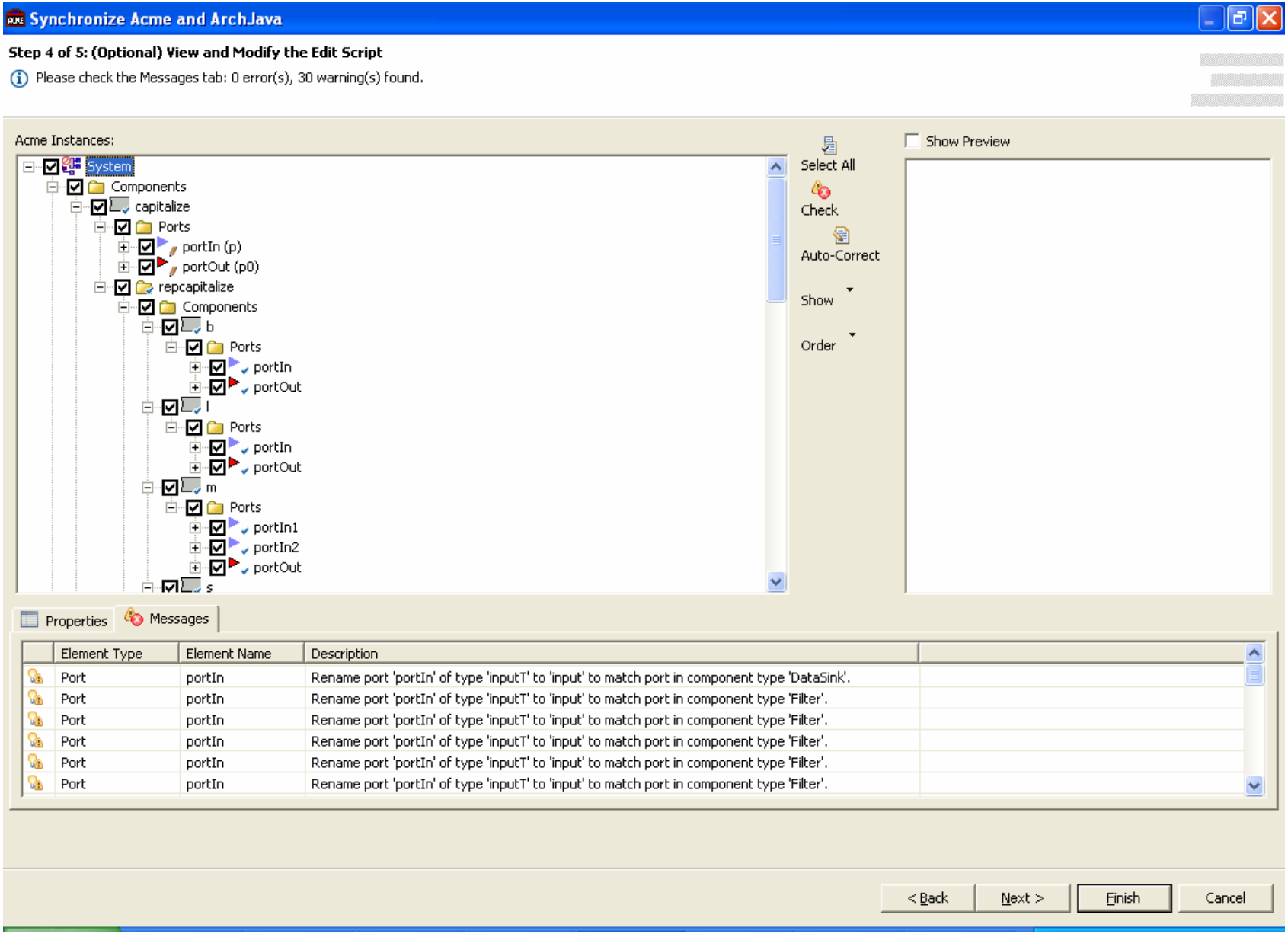
Name	Value
IsDynamic	false
AcmeFamilyType	PipesAndFiltersFam.inputT

< Back | Next > | Finish | Cancel

Step 4: View & modify edit script

- Step 1: Setup synchronization
- Step 2: View & match types
- Step 3: View & match instances
- **Step 4: View & modify edit script**
- Step 5: Confirm & apply edit script

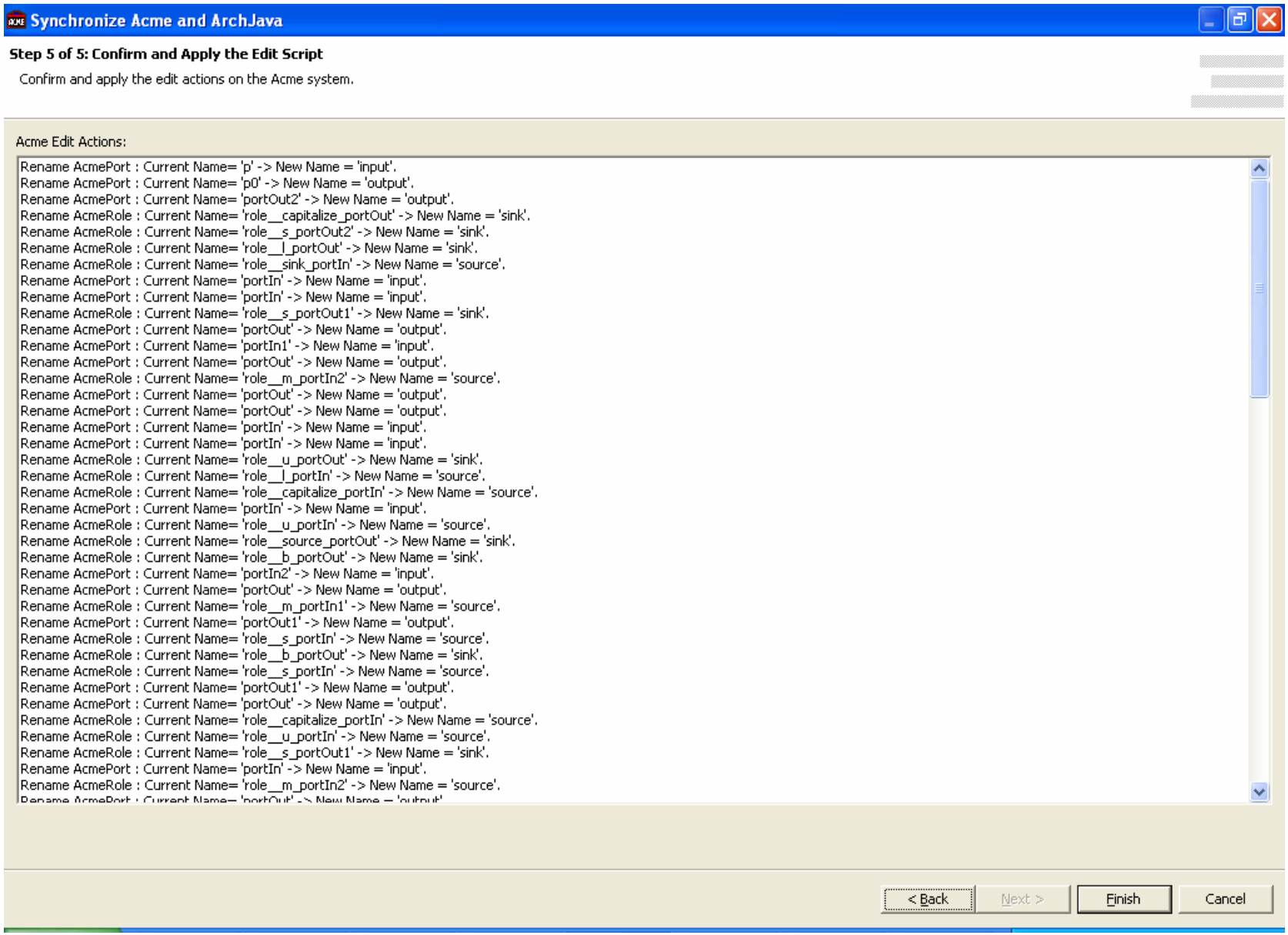
Step 4: View & modify edit script



Step 4: Details

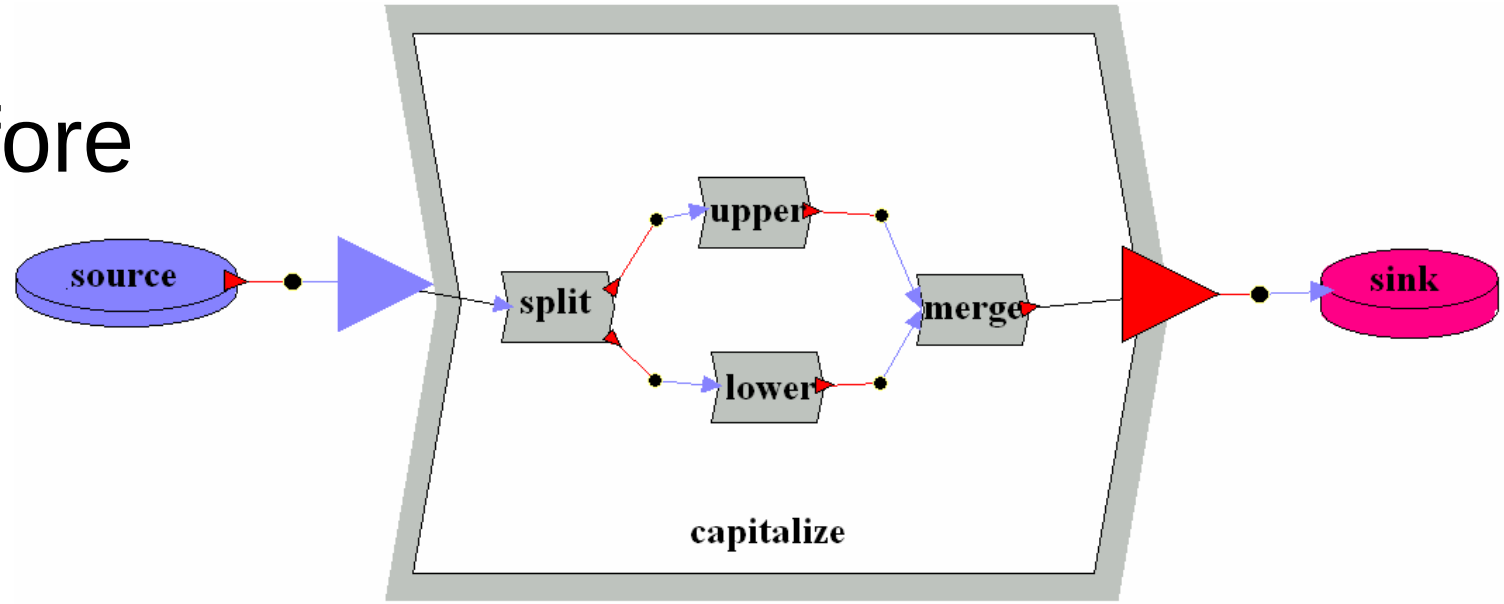
- Types affect processing of the edit script
  - If component instance inherits ports from its assigned type
    - *Filter* has ports *input: inputT* and *output: outputT*
  - May need not create additional ports on the component instance
  - May rename a port to match name declared in architectural type
    - Rename *portIn to input; portOut to output*

Step 5: Confirm & apply edit script

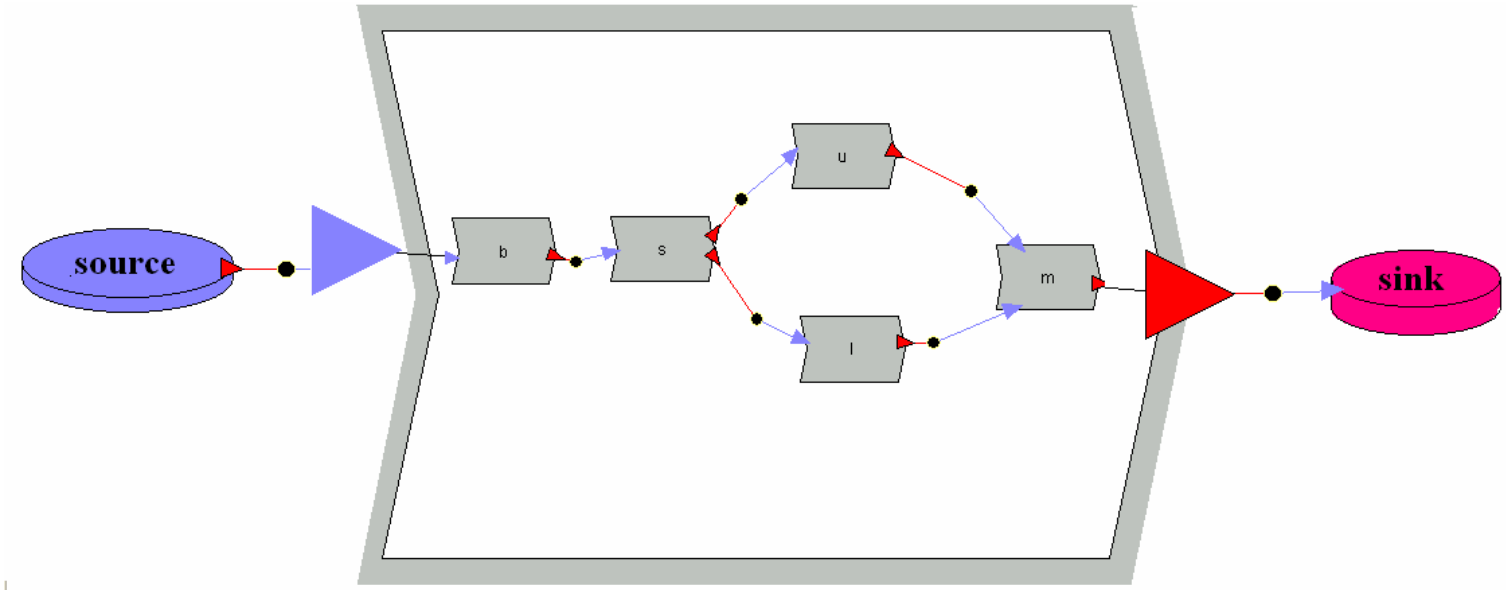


Final Result

Before



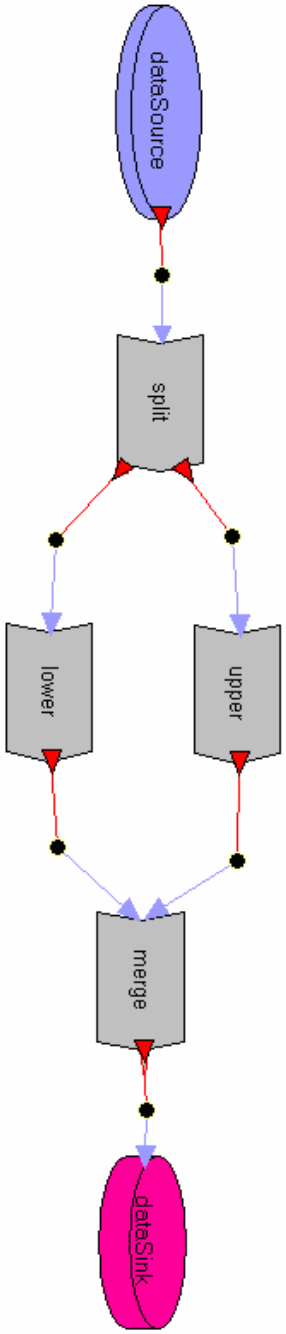
After

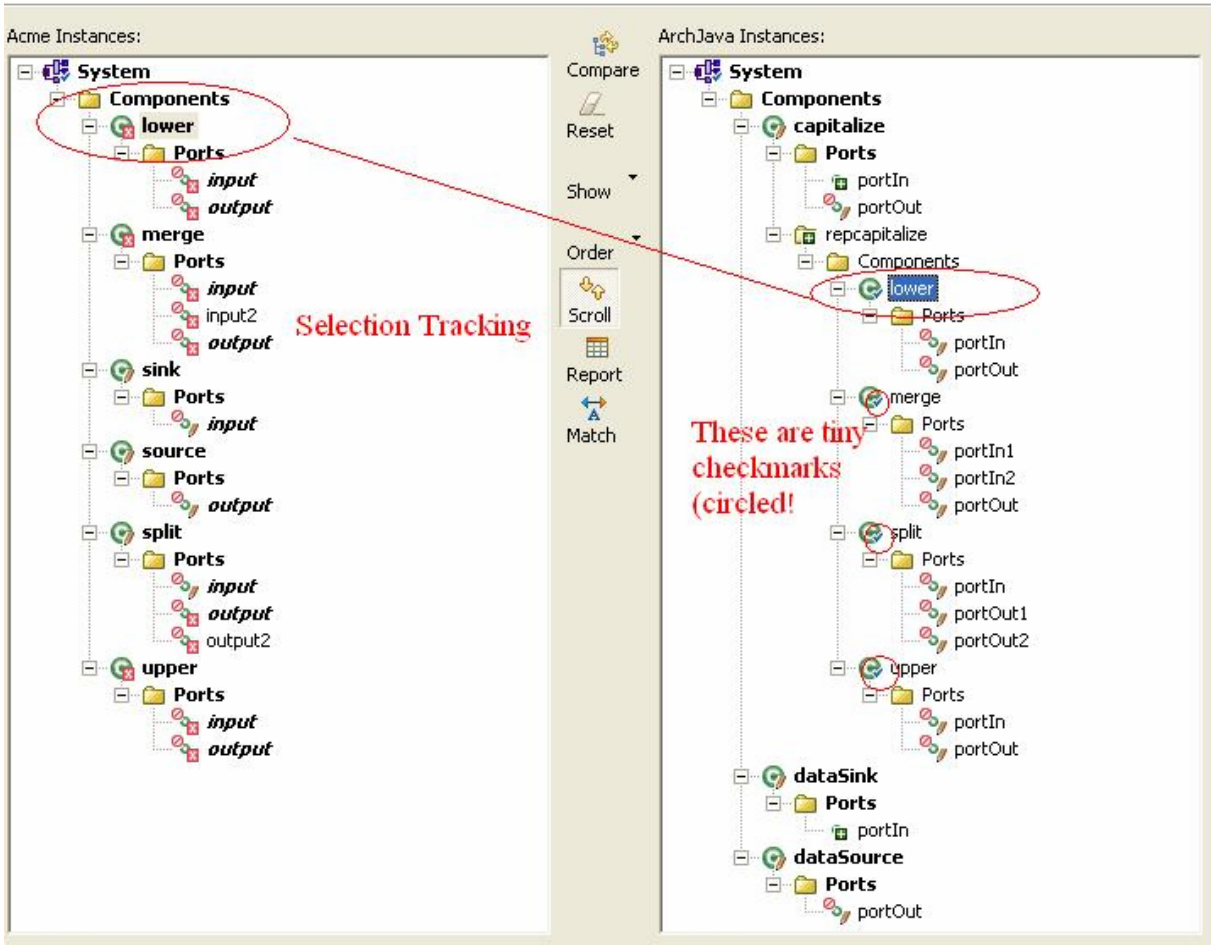


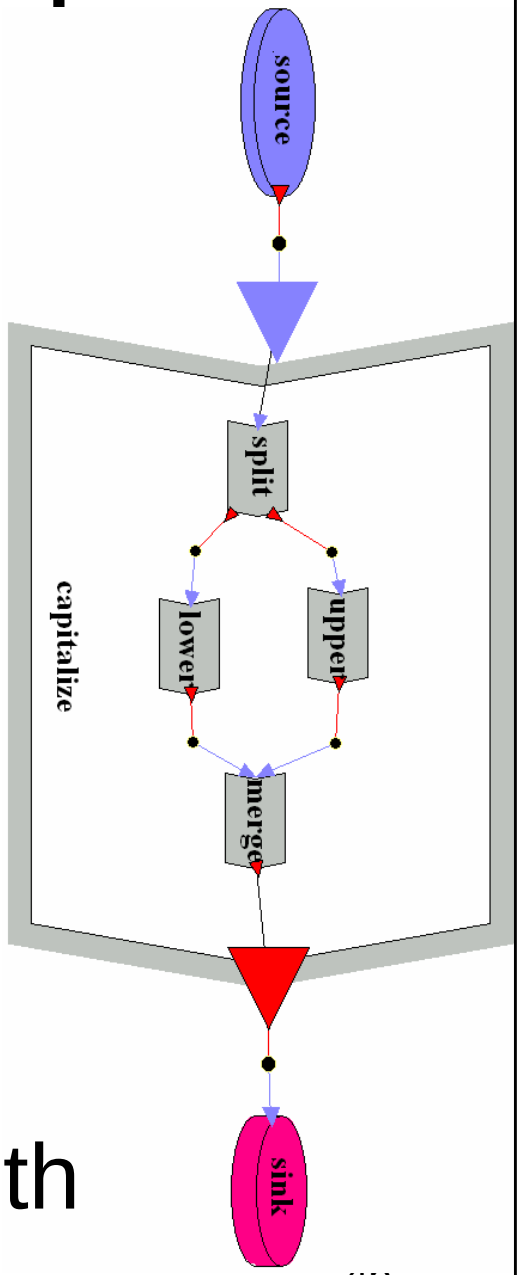
What we didn’t tell you

- Automated refinement of conceptual C&C view into skeleton code
- Automated abstraction of a C&C view from an implementation
- Structurally compare two versions of an implementation
- Structurally compare two versions of an architectural model

Detecting Moves Example



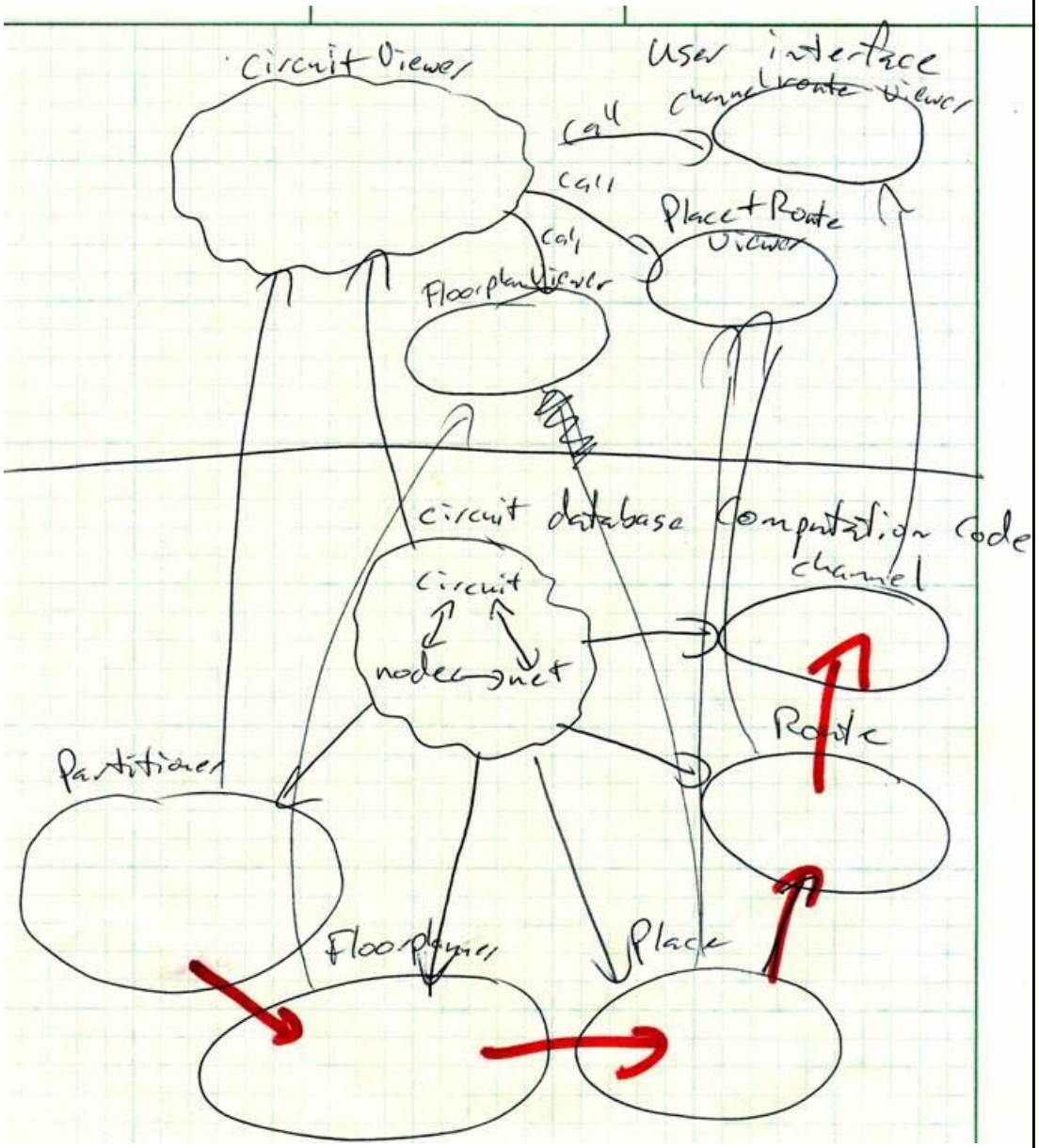




E.g., replace a component with its representation

Stay tuned for more!

- Extended Example on Friday's informal demo session
- ArchJava architecture consisting of
  - Over 20 components, 80 ports, several subsystems



Conclusion

- 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/