Architecture-Driven Software Development using Acme and ArchJava

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

Presentation on research in progress
Software Research Seminar (SSSG)
Jan 12th 2005

Marwan Abi-Antoun

Problem Statement

"How does one ensure that every trifling detail of an architectural specification gets communicated to the implementer, properly understood by him, and accurately incorporated into the product?"

Frederick P. Brooks, Jr., *in* The Mythical Man-Month

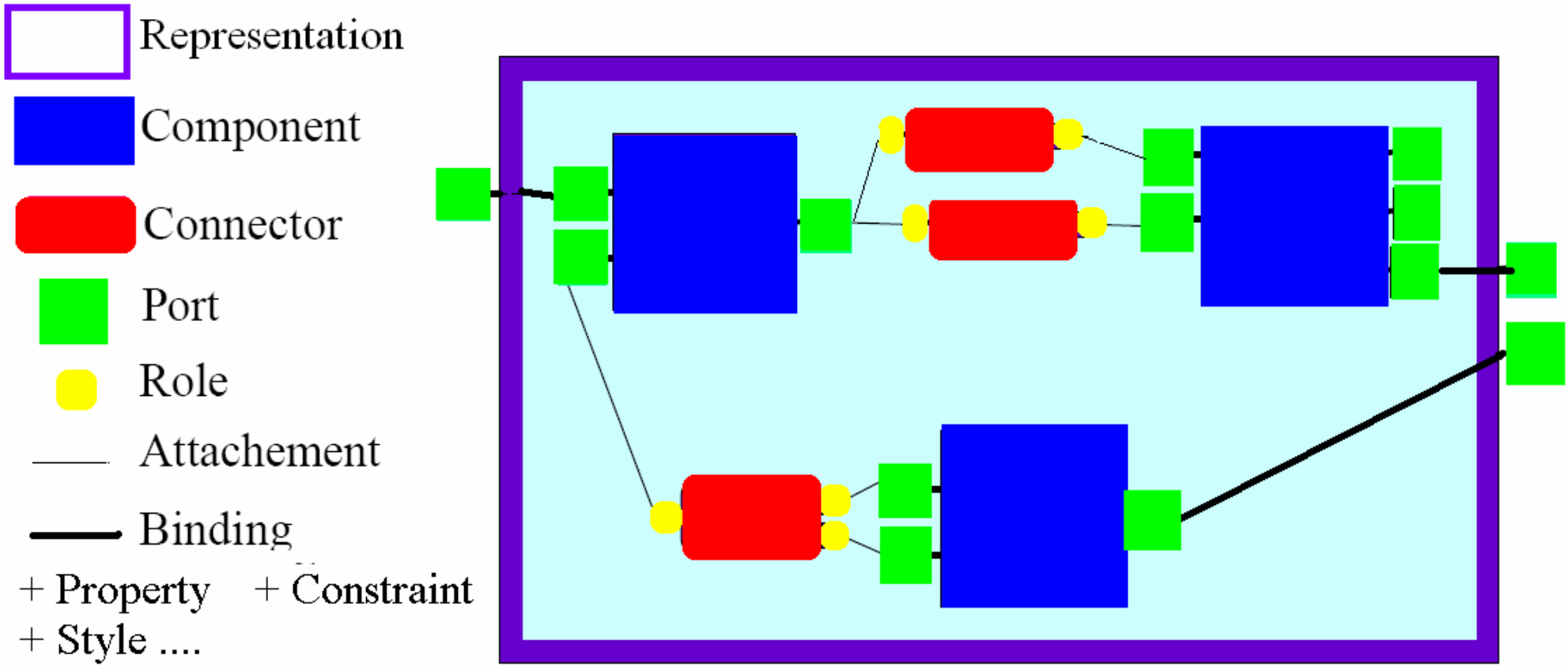
Conceptual Integrity

- Importance of a documented software architecture specification:
  - Outsourcing and/or distributed development
  - Software maintenance and evolution
- Architectural model either missing or obsolete or evolves separately from implementation
- Need to check conformance and keep specification in sync with implementation

ArchJava in a Nutshell

- Extension of Java programming language
- Specify architecture directly within code:
  - Components, Connectors, Ports, Glue...
  - Code serves as documentation of architecture
- Guarantee architectural conformance
  - Type system enforces inter-component communication
  - Component substitutability

Acme Terminology



Acme + ArchJava

- Use Acme to check other properties:
  - Architectural style
  - System behavior
- Use Acme's tool support (AcmeStudio):
  - Visualization
  - Advanced analyses:
    - Performance analysis based on queuing theory
    - Rate monotonic schedulability analysis

Main Contribution

- Continuous/incremental synchronization of an Architecture (Acme) and an Implementation (ArchJava)
  - Initial generation of ArchJava stubs from Acme model
  - Initial creation of Acme model from existing ArchJava implementation
  - Synchronization between existing Acme model and existing ArchJava implementation

Industrial Experience

- Generate architecture diagrams
  - Boxes and arrows
- Generate UML class diagrams from existing code:
  - Pull-in too many low-level details!
  - Spend time weeding output (every time)
- No way to bridge the gap between the two
  - Manual comparison
  - Manual merge

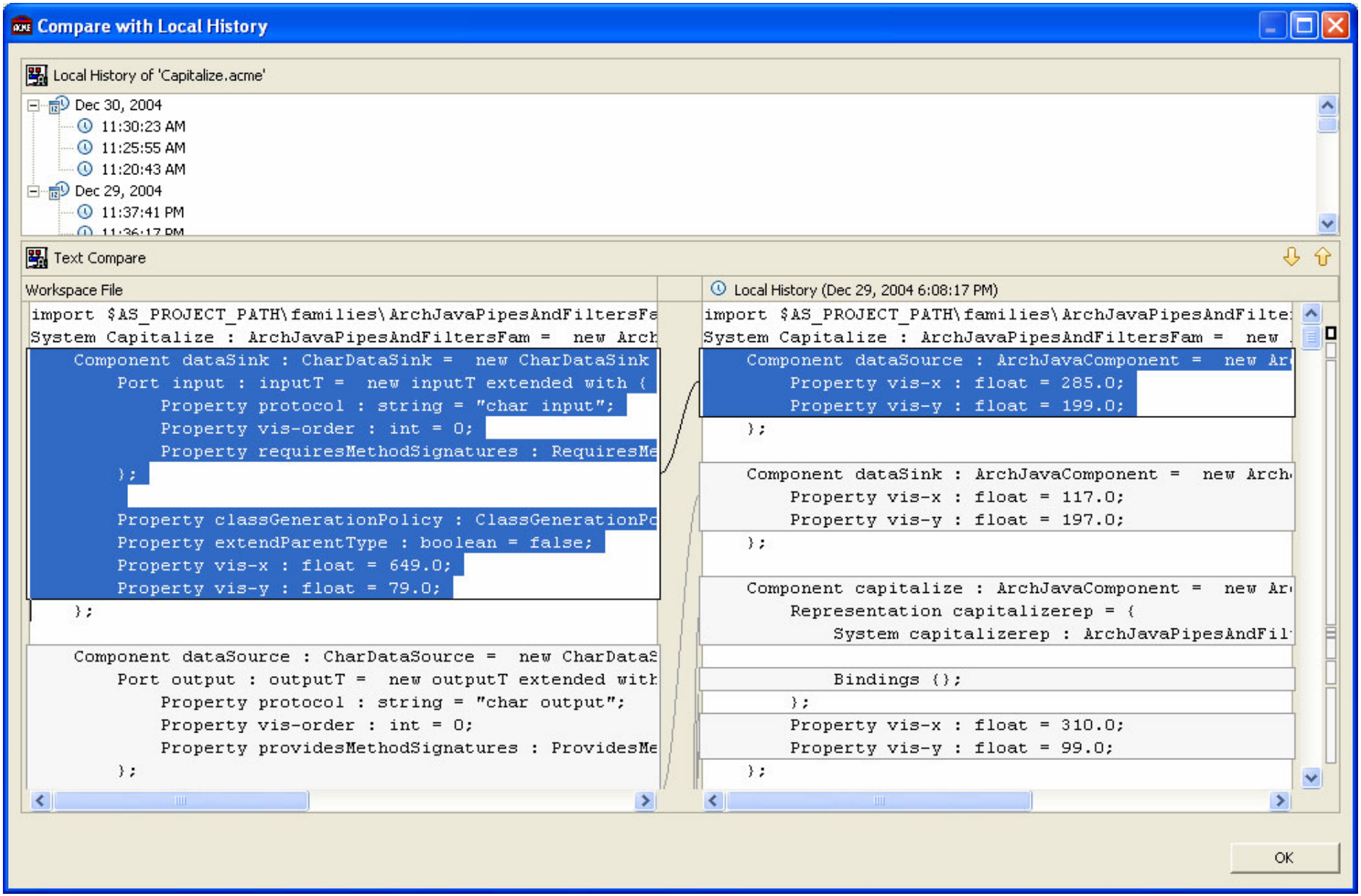
Bridging the Gap

- Perceived difficulties:
  - Reconstruct architecture from implementation
    - Breakthrough: ArchJava
  - Handle overlapping/missing information
    - Partial support from ArchJava
- Detecting differences:
  - First step of synchronization process
  - Adopt structural not textual comparison

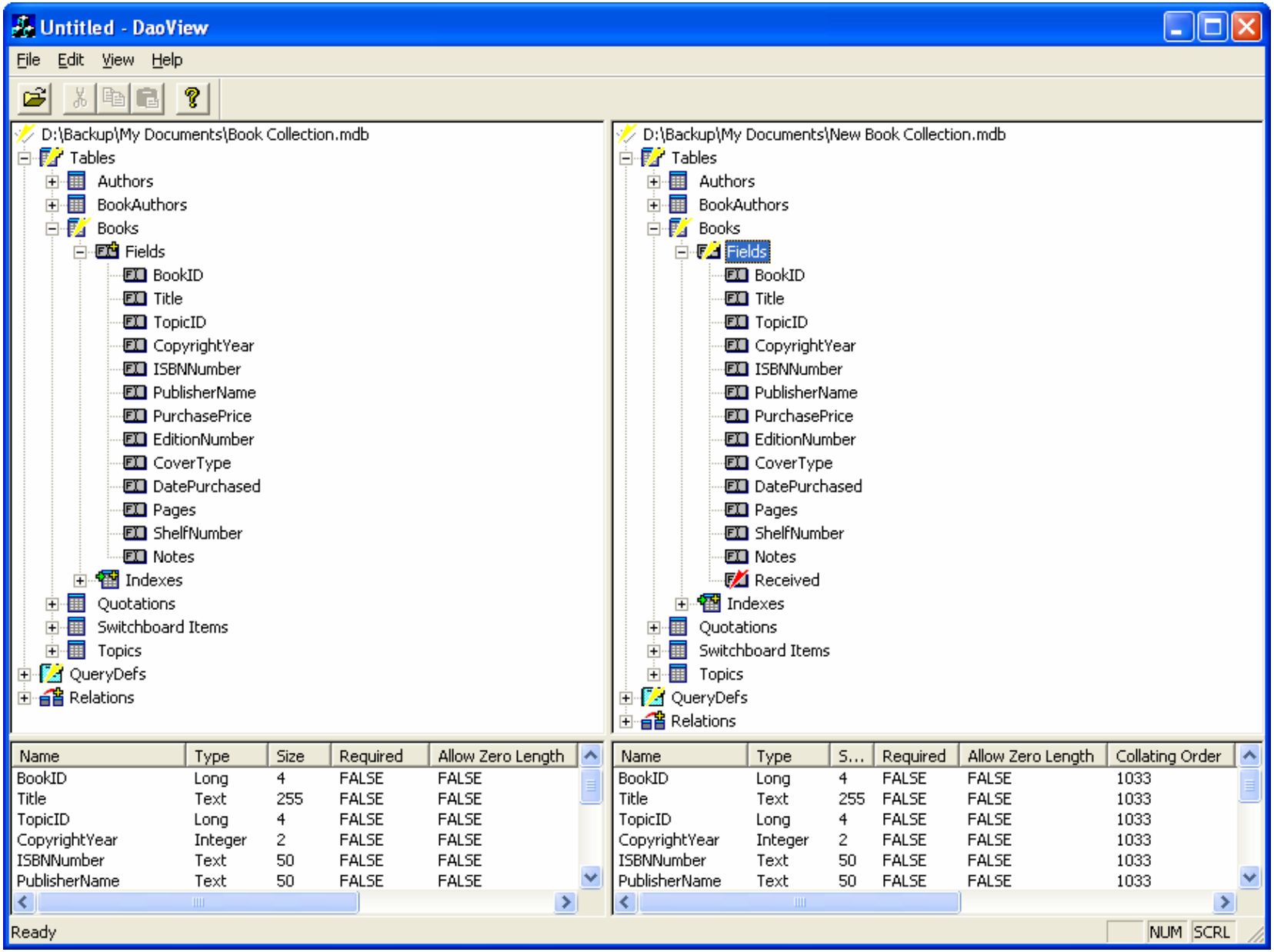
Textual Comparison

- Used by most commonly available tools
- Line-oriented
- Arbitrary ordering on unordered data
- Not suitable for tree/graph structured data:
  - XML files
  - Database schemas
  - Architecture Description Languages (ADLs)
- Focus on string labels at the expense of other semantic attributes

Limitations of Textual Comparison



Example Structural Comparison



Key Ideas

- Use only architecturally relevant elements:
  - Ignore classes/fields not of type *component*
  - Can ignore port methods if not in Acme
- Ignore non-essential details:
  - Names for connectors, roles, attachments, …
- Component instance as organizing entity
- Compare both types and instances
- Type-driven

Structural Comparison

- Build complete intermediate representation for Acme and ArchJava
  - Includes types, instances, attachments, …
- Compare intermediate representations:
  - Exclude parts from comparison as needed
- Do NOT generate or rely on tags/comments in Acme or ArchJava
- Do NOT listen to Acme or ArchJava change events (disconnected operation)

Structural Comparison Goals

- Be able to detect:
  - Matches 
  - Inserts
  - Deletes 
  - Renames  (do NOT treat as Insert + Delete)
- Not (initially) supported:
  - Moves

Your Feedback

- Demonstration of early prototype
  - Unordered comparison completed last week
  - Some features not enabled yet
- We're very interested in your feedback!
  - Approach
  - Usability
  - Suggestions for improvement
- Periodic demonstrations (outside SSSG)

Currently Supported Scenarios

- ArchJava Stub Generation (skip)
  - Generate ArchJava stubs from Acme Model
- Architectural Import
  - Import Acme model from ArchJava implementation
- Architectural/Code Evolution
  - Additional code is added
  - Detect and create representation in Acme
- Architectural Synchronization
  - Rename components in Acme
  - Detect renames when comparing with ArchJava

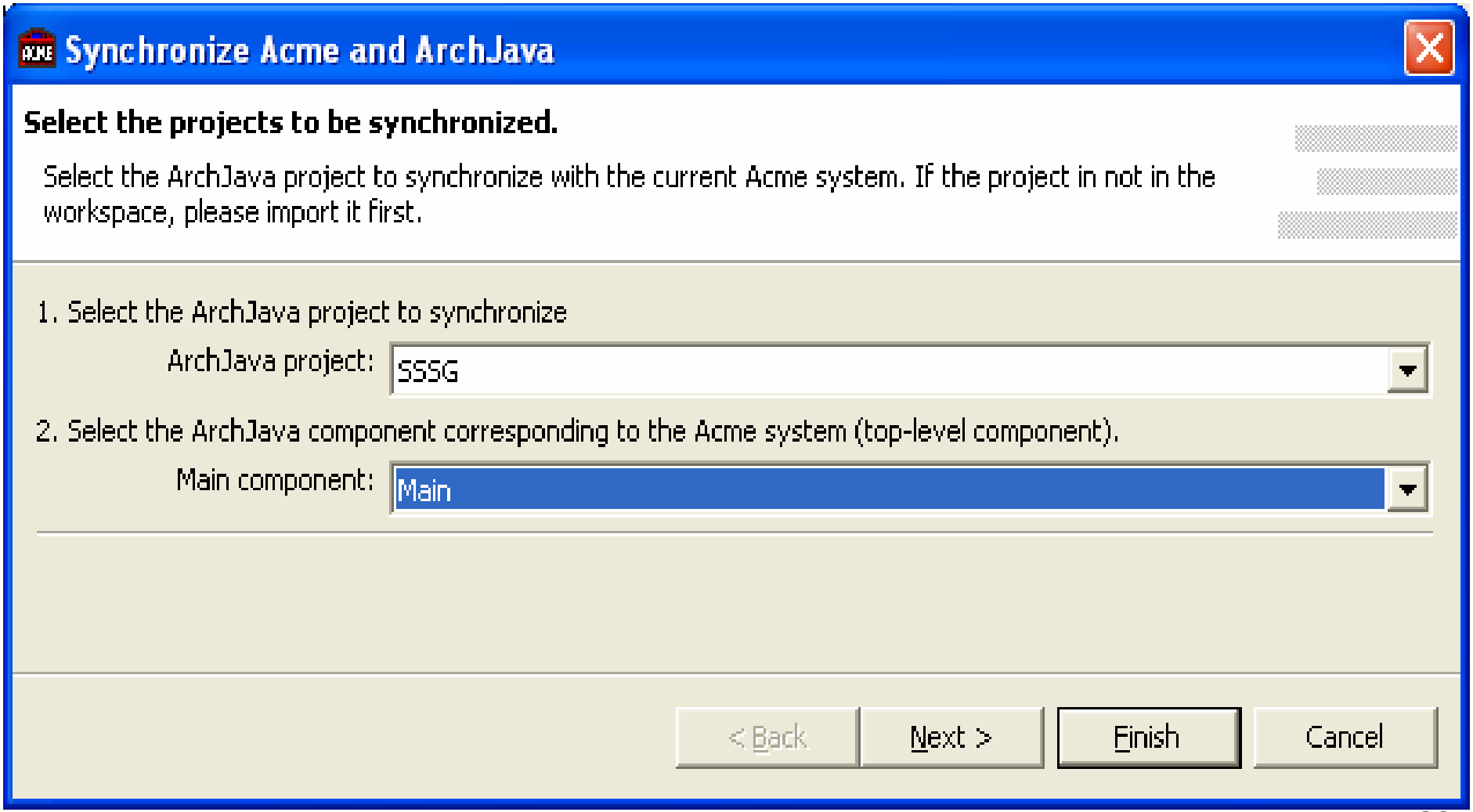
Walkthrough

- Setup Synchronization
- Compare Type Information (skipped)
- Compare Instance Information
- Generate Edit Script
  - Only Acme edits supported for now
- Apply Edit Script
- Visualize Changes in AcmeStudio

Setup Synchronization

- Select/Create Acme model:
  - Current model: select system/representation
  - New model: select or import families
- Select an ArchJava project from the workspace (import if necessary)
- Select the main component class

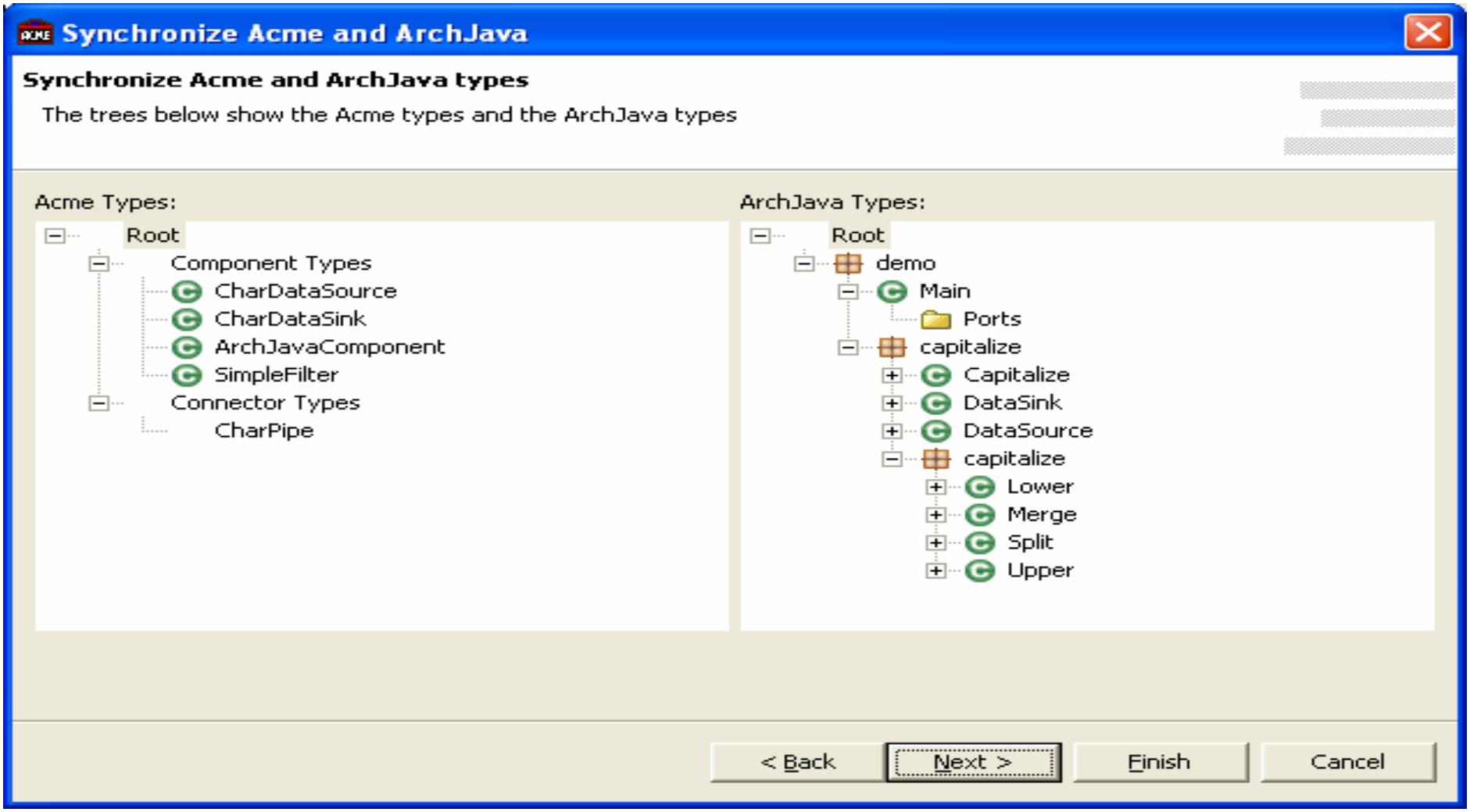
Setup Demo


Synchronize Acme and ArchJava
Select the projects to be synchronized.
Select the ArchJava project to synchronize with the current Acme system. If the project in not in the workspace, please import it first.
1. Select the ArchJava project to synchronize
ArchJava project:
SSSG
2. Select the ArchJava component corresponding to the Acme system (top-level component).
Main component:
Main
< Back
Next >
Finish
Cancel

Compare Type Information

- Retrieve types from Acme and ArchJava
- Not very useful if no ArchJava classes were generated for Acme component types:
  - *InstanceOnly* setting for generating ArchJava stubs
  - Most commonly used setting

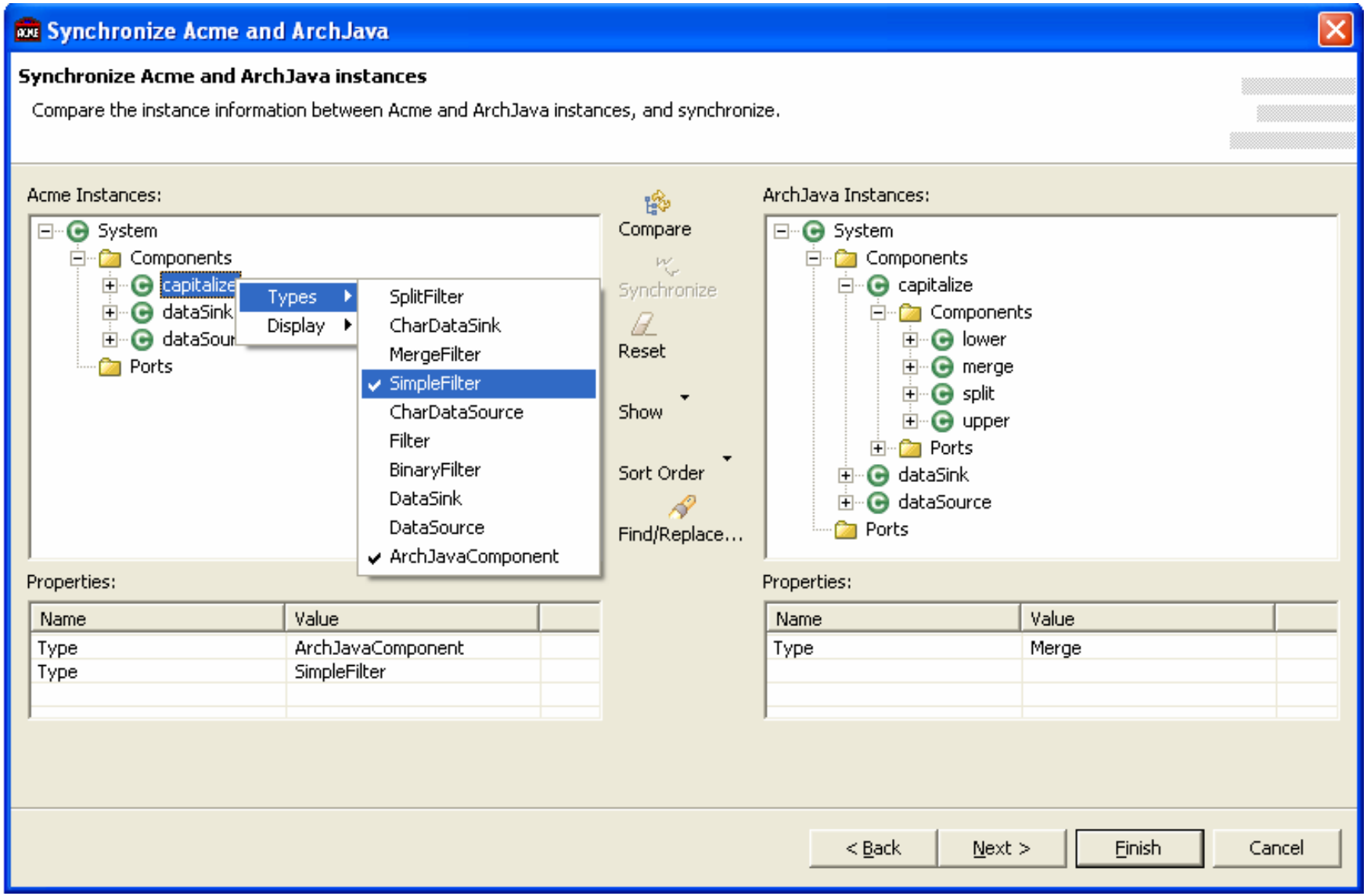
Type Information Demo



View Instance Information

- Components
- Port
  - Port Involvements
  - Provided Methods
  - Required Methods
- Connectors
- Attachments
- Bindings
- Types for instances

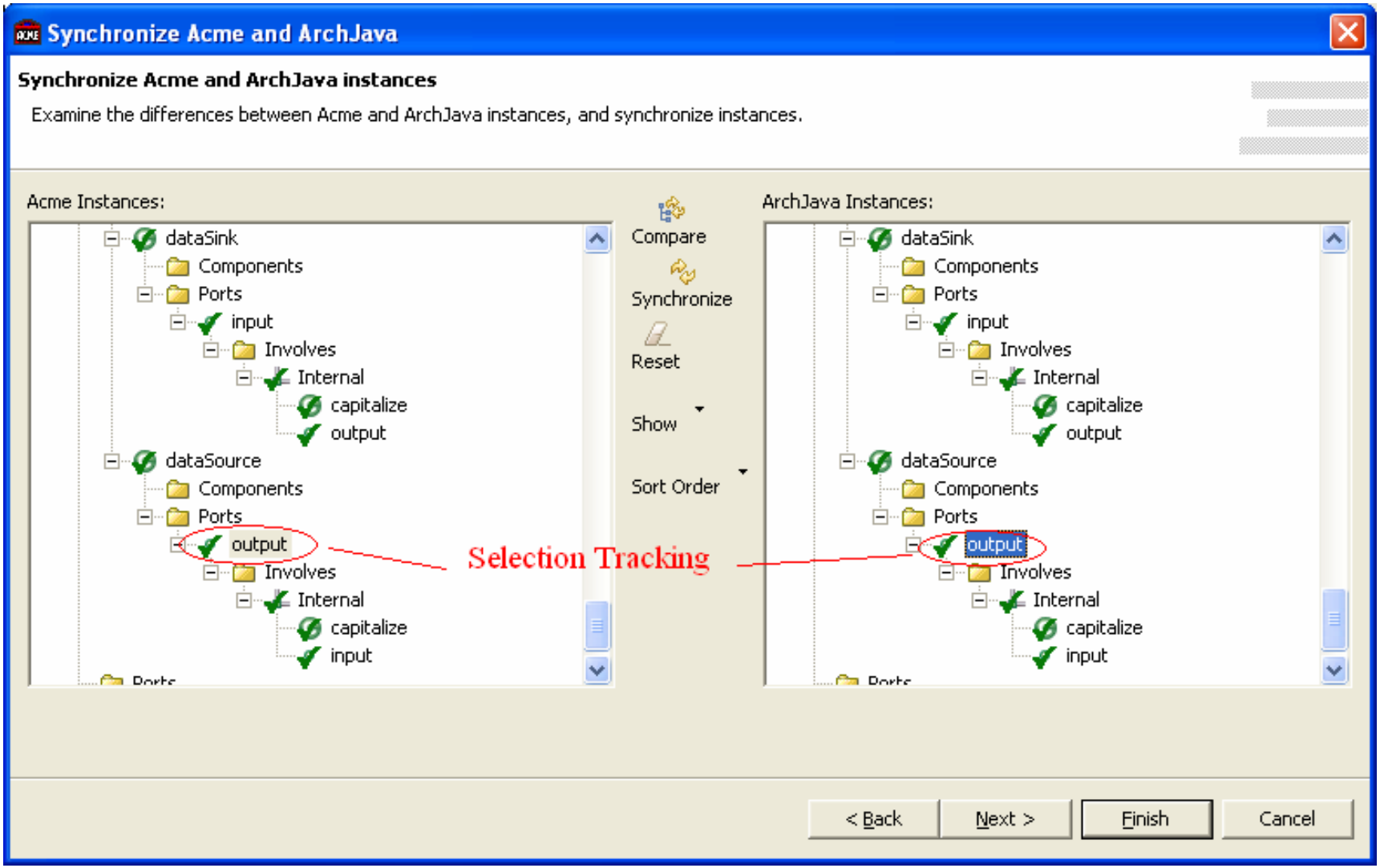
View Instance Information Demo



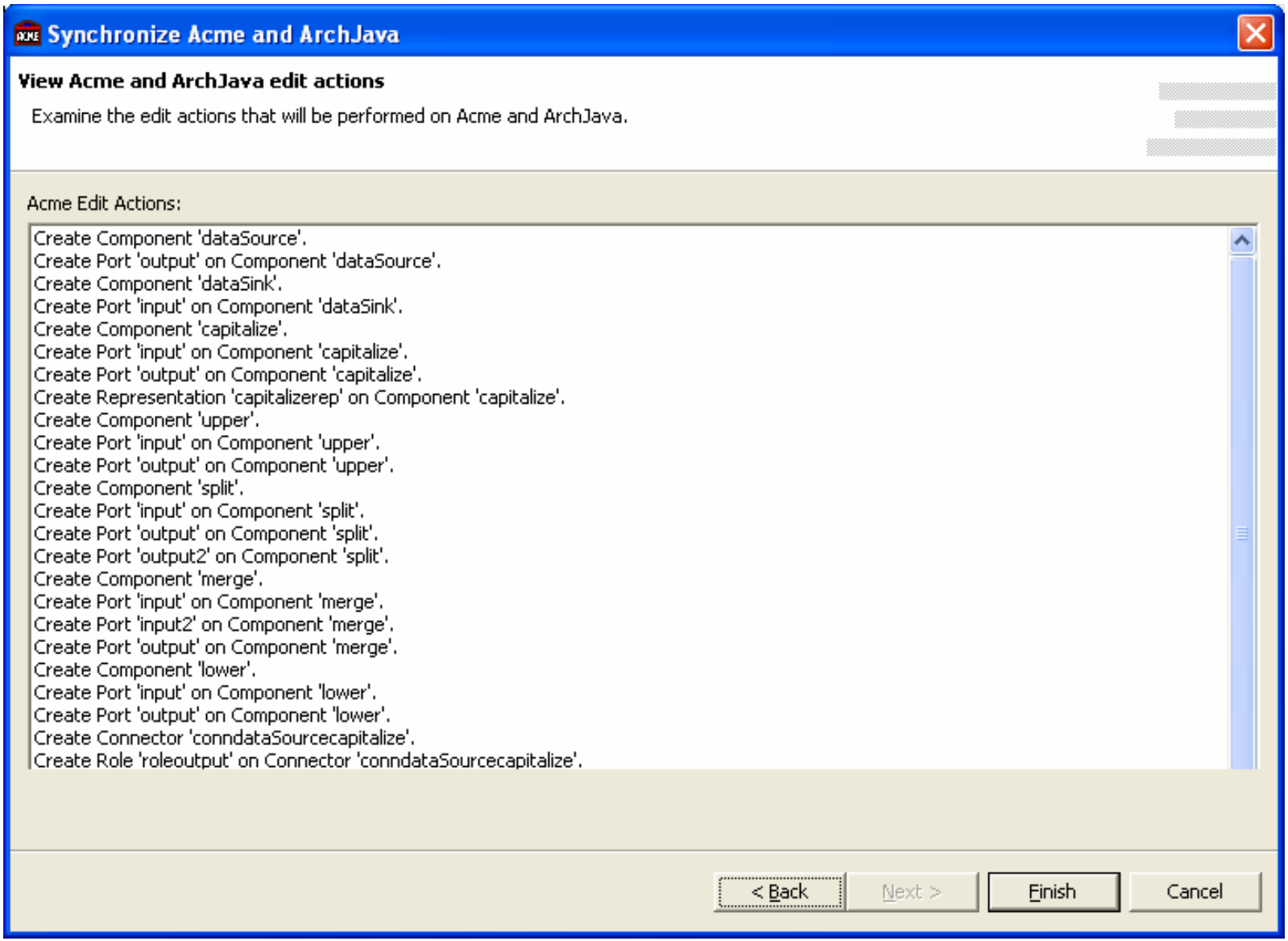
Compare Instance Information

- Assigning Types affects the Matching
  - Assigning a type to a component could mean that the component inherits port from the type
  - E.g., port can be inherited from the type

Compare Instance Information Demo



Edit Script Demo



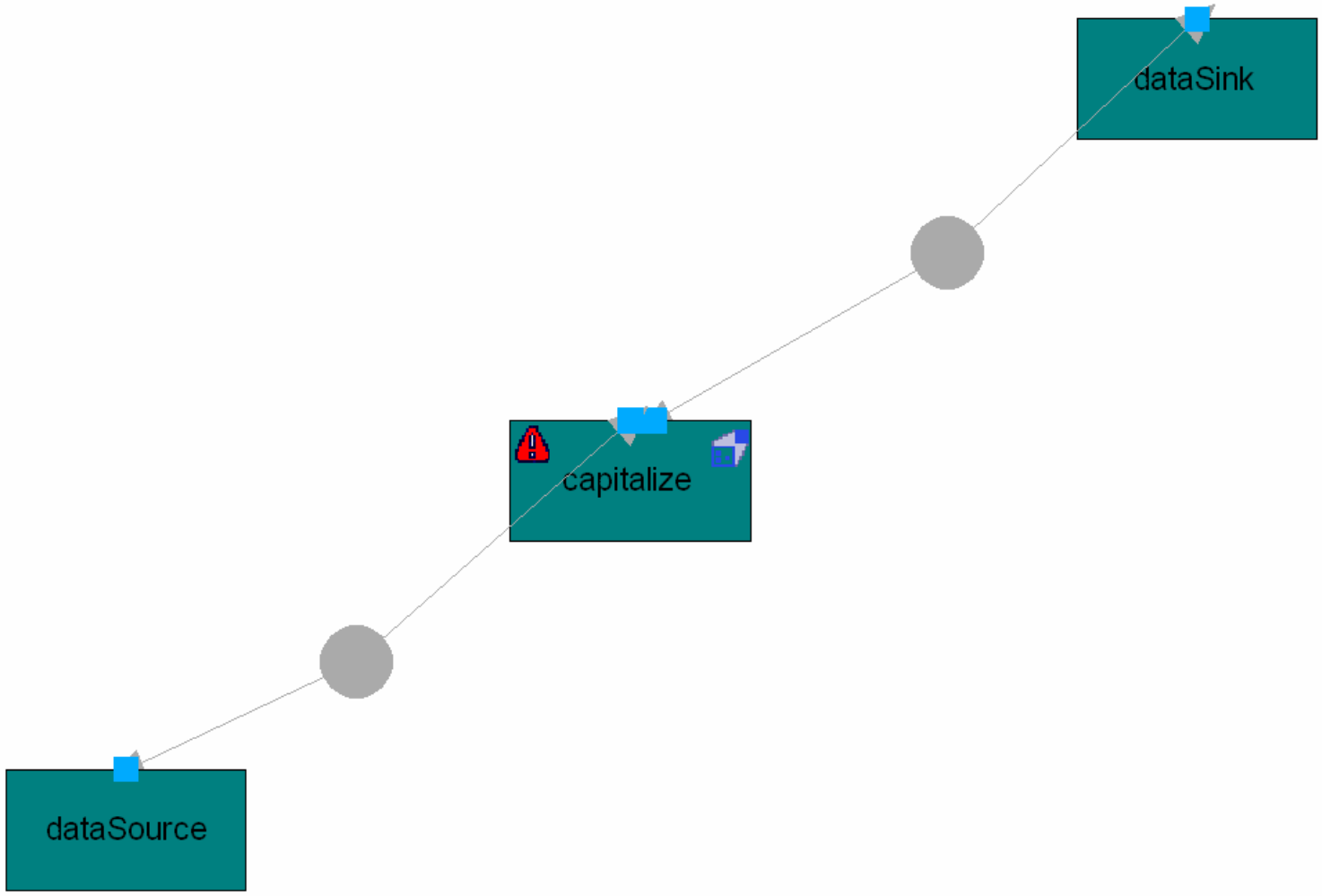
Visualize in AcmeStudio

- Visualize Acme Model:
  - AcmeStudio: no layout engine out of the box
  - Implemented layout using a force-directed layout algorithm (JIGGLE)
- Accidental difficulty:
  - Layout unrelated to synchronization
  - Absolutely essential when generating many Acme elements automatically
  - Bad/Missing coordinates crash AcmeStudio

Apply Edit Script

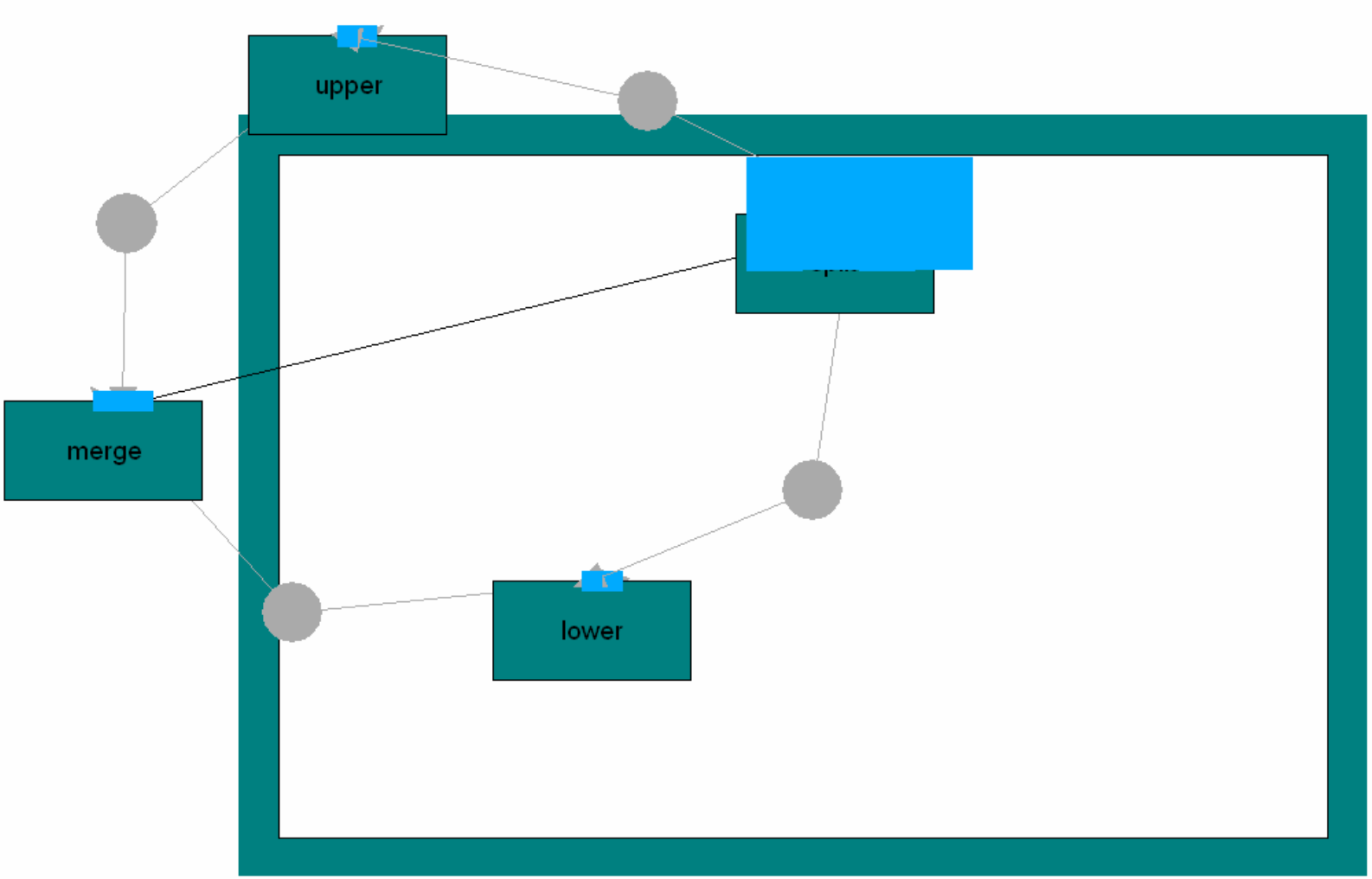
- Layout depends on assigned type:
  - Ports: input ports → left, output ports → right
- Clicking on ‘Cancel’ abandons all changes
- Future Work:
  - Verify types assigned to Acme elements to be created
  - Verify no illegal Acme names (e.g., reserved keywords) will be generated
  - Allow user to modify edit script

Imported Architecture – Part 1



Raw layout currently generated for system

Imported Architecture – Part 2



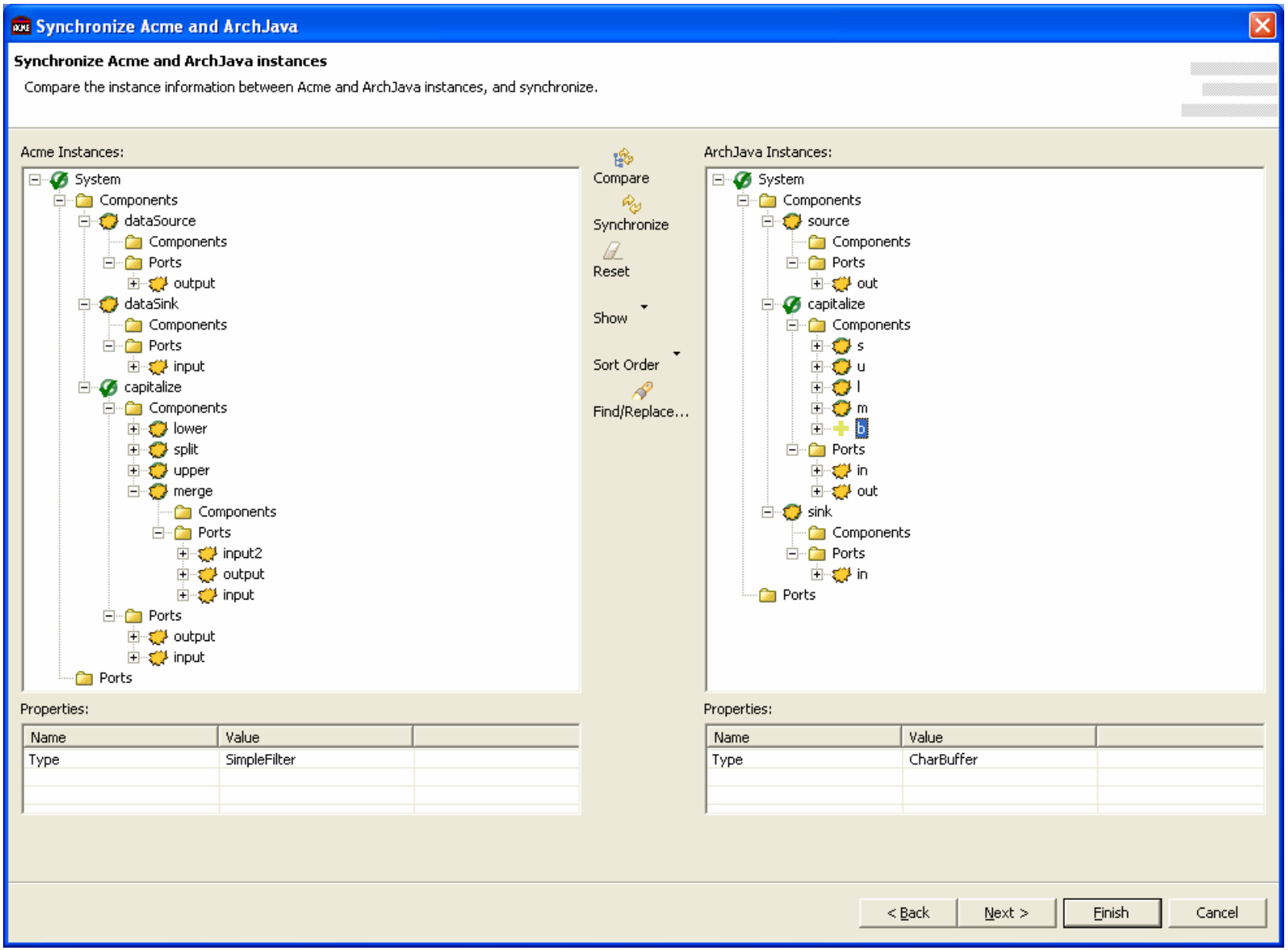
Raw layout currently generated for representation

A hard case

- Early ArchJava implementation
- Predates current Acme model
- Many changes:
  - Everything renamed
  - One insertion (component)
  - Some deletions (ports)
- Subtrees are too similar!
- Acme model may not have required/provided methods
  - Cannot use to distinguish between children

Acme	**Arch Java**
	b
lower	l
merge	m
split	s
upper	u

Demonstration of the hard case



Research Questions - 1

- Missing type information in ArchJava:
  - No declared types for ports, connectors, …
- Missing instance information in ArchJava:
  - No declared names for ports, connectors, attachments, bindings, …
- False positives:
  - “Global” attachments (Acme) v/s attachments spread out across components (ArchJava)
  - “port involvements” minimize structural differences between Acme and ArchJava

Research Questions - 2

- Structural typing instead of nominal typing:
  - Java-like languages use nominal typing
  - Names are the most easily changed
  - Rely on additional semantic information
- Architectural type inference:
  - User-assigned types for Acme model elements to be created
  - Is it possible to infer (some) types?

Research Questions - 3

- Type-check incremental changes/additions to Acme and ArchJava
  - Never push breaking changes (e.g., create dangling ports, break substitutability, …)
- Better handling of first-class connectors:
  - Assigning types to connectors, roles
  - Assigning representations to connectors
  - Using connector instances as organizing entities for structural comparison

Structural Comparison

- Standard *tree-to-tree correction* problem
- Ordered Tree Comparison
- Unordered Tree Comparison
  - May fit problem domain better
  - Less attention than ordered comparison
  - General problem proved MAX SNP-HARD
  - Assumptions can produce polynomial time
- We think we should keep both for now

Ordered Tree Comparison

- Implemented Zhang-Sasha algorithm:
  - Exact (not approximation) algorithm
  - Generates optimal Tree Edit Distance
  - Good performance (with no optimization)
- Our enhancements:
  - Structural instead of alphabetical ordering
  - String-to-string correction instead of matching
  - Forced matches between selected nodes

Unordered Tree Comparison

- Implemented adaptation of algorithm by Torsello, Hidovi and Pelillo
- Chosen assumption:
  - If two nodes match (i.e., either exact match or rename), so do their parents
  - Order does matter within each sub-tree!
  - This assumption can yield bad answers:
    - Create new component which wraps already existing components

Usability Questions - 1

- Using trees for software architecture visualization
  - Asymmetric display of changes
  - Right-click menu to edit/assign types
  - Propagate up to root detected differences
  - Using trees for other than inheritance trees!
- Present user with multiple choices
  - Allow manual override of detected differences

Usability Questions – 2

- Allow the user to modify the edit script
  - The allowed changes not orthogonal
- Support saving edit script as "patch" file
  - Can be applied repeatedly
- Undo support
- Global "Find/Replace" with regular expressions to specify possible (pattern) matches (e.g., 'in*' ←→ 'input', 'in1', …)

Engineering Questions

- Generalize implementation to allow comparing/merging between:
  - Two Acme implementations
  - Two ArchJava implementations
  - Useful for comparing versions stored in a configuration management system
- Layout engine as AcmeStudio plug-in:
  - Mainly packaging/licensing issues
  - Need architectural style-specific rendering

Future Implementation Work

- Refactor ArchJava code generation as special case of synchronization
- Add support for incremental changes in ArchJava
- Improve layout inside Acme representation
- Support modify/save/import Edit Script

Generality of Approach

- Other Architecture Description Languages:
  - Main consideration: availability of tool support
- Other implementation languages as long as architectural information available:
  - Use Java annotation or C# custom attribute to specify a component class
  - Retrieve coclass'es from type-library (TLB) or Interface Definition Language (IDL) in Microsoft COM implementations (Visual Basic, Visual C++)

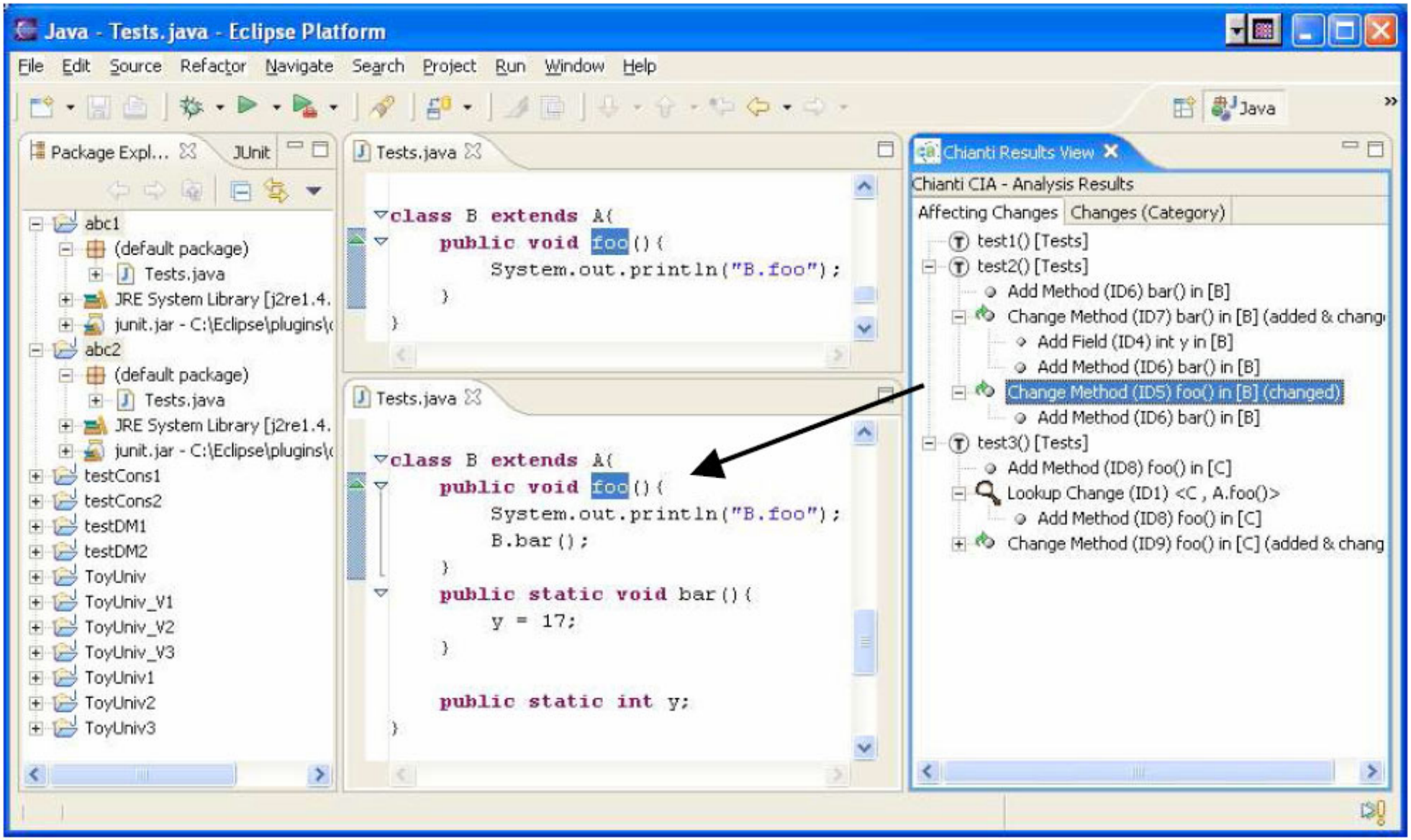
Scalability of Approach

- Some parts more stable than others
- Start synchronization of large systems at:
  - Acme representation
  - ArchJava component class
- Polynomial-time algorithms
- Tree structure amenable to optimization:
  - Lazy population on node expansion
- Edit script used to keep track of changes

Complementary Approaches

- We focus on “upstream” activities
  - Architecture ←→ Implementation
- Others focus on “downstream” activities
  - Change Impact Analysis
    - E.g., CHIANTI (fine-grained structural comparison)
  - Refactoring
    - Eclipse support (textual comparison)
- NOT implying a “waterfall” process

Chianti: Change Impact Analysis of Java Programs



Related Work

- Comparing and Synchronizing Tree-Structured Data (e.g., XML, HTML, SGML)
  - X-Diff, X-Tree Diff, XML Diff (Microsoft), XML TreeDiff (IBM)
- Graph eXchange Language (GXL):
  - Description language used in reverse engineering community to communicate data between reverse engineering tools

Acknowledgements

- Nagi Nahas
  - Implementation of ordered and unordered comparison algorithms
- Bradley Schmerl
  - Help with AcmeLib and AcmeStudio
- Daniel Tunkelang (CS Ph.D.'98)
  - Author of *Java Interactive Graph Layout Environment* (*JIGGLE*)
  - http://www-2.cs.cmu.edu/~quixote/

References

[1] Dennis Shasha, Kaizhong Zhang. *Approximate Tree Pattern Matching*, in *Pattern Matching Algorithms*, A. Apostolico and Z. Galil, Eds., chapter 14. Oxford University Press, 1997.

[2] Kaizhong Zhang and Tao Jiang. *Some MAX SNP-hard results concerning unordered labeled trees*. Information Processing Letters, 49:249–254, 1994.

[3] Andrea Torsello, Džena Hidovi , Marcello Pelillo. *Polynomial-Time Metrics for Attributed Trees*. Dipartimento di Informatica, Università Ca' Foscari di Venezia. TR CS-2003-19.