

Semi-Automated Incremental Synchronization between Conceptual and Implementation Level Architectures

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

*Institute for Software Research Intl
Carnegie Mellon University*



Architectural Conformance



- Runtime software architecture views
 - Components, connectors, and constraints on how they interact
- Benefits of architecture contingent upon correct implementation
 - Program understanding
 - Software evolution
 - Checking architectural constraints
 - Analysis of quality attributes



Conceptual-Level Architecture

- Expressed in an Architecture Description Language (ADL)
- Architectural styles
 - Sets of related architectures
 - Types of components, connectors, ...
 - Topological constraints
- But, does ***not*** guarantee that implementation conforms to architecture

Implementation-Level Architecture

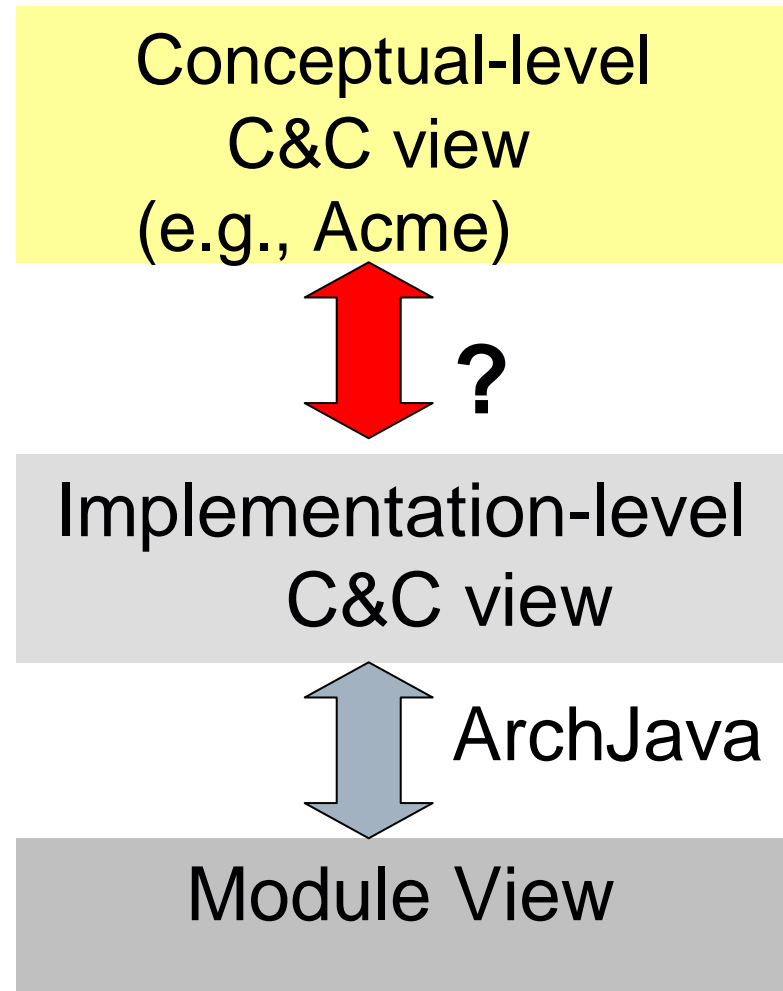


- ArchJava
 - Extension of Java programming language
 - Code = architecture specification
- Specify architecture directly within code
 - Components, Ports, connections
- Enforce communication integrity
 - Two components in the implementation may communicate only if they are connected in the architecture.
- Does **not** enforce architectural properties
 - Style constraints, analysis of quality attributes...

Relating Conceptual- and Implementation-Level views



- Conceptual-level C&C view
 - Architect's design view
 - Problem-specific
 - May elide information
- Implementation-level C&C view
 - Actual communication between implementation components





Our first observation

- We need to carefully reason about differences between
 - Design languages (e.g., ADLs) vs. implementation-oriented languages
 - Conceptual-level C&C views vs. implementation-level C&C views
- Some dimensions we identified
 - Matching type structures
 - Matching hierarchies
 - Incidental differences

Matching Type Structures



Acme C&C View

- Predicate-based type system
- Types = logical predicates
- Interfaces optional
 - Properties on ports
- Architectural style constraints

ArchJava C&C View

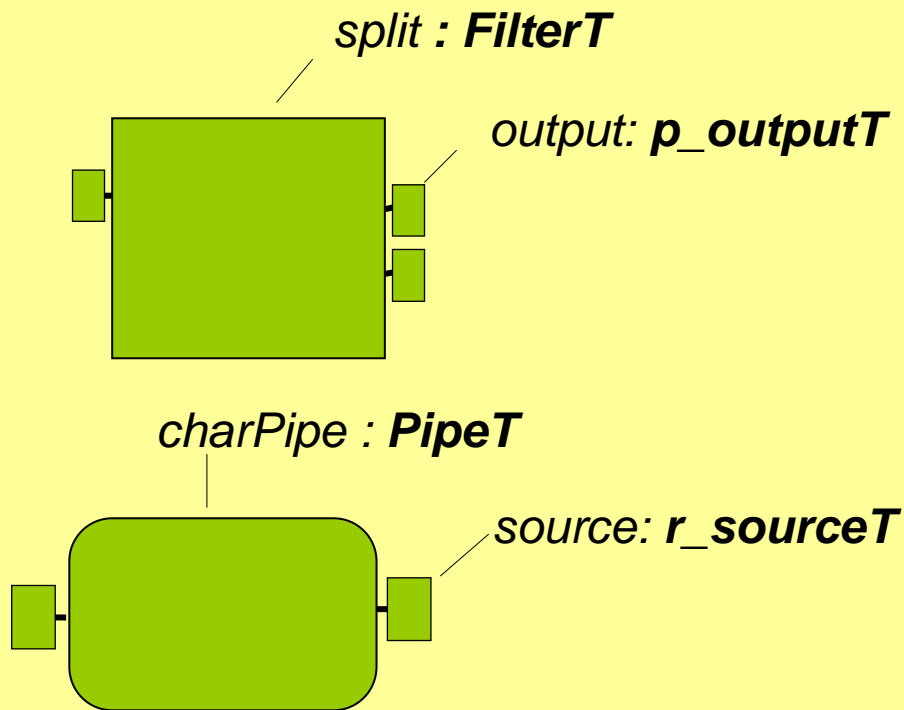
- Conventional type system
- ArchJava types
 - Interface of provided and required functionality
- No first-class types for ports, roles

Matching Type Structures

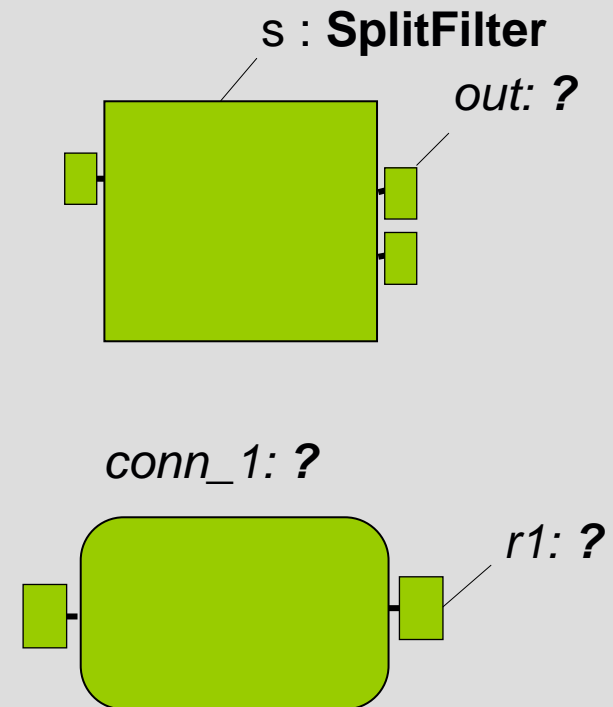


Conceptual View

system: *PipeAndFilterStyle*



Implementation View



- First-class types missing in ArchJava for connectors, ports, roles
- Conceptual C&C view types at higher level of abstraction

Matching Hierarchies



Acme C&C View

- Acme Hierarchy
 - Design-time composition
 - Element = \sum (parts)
 - No notion of visibility
- Multiple representations
 - Multiple decompositions
 - Only one actually implemented!

ArchJava C&C View

- ArchJava Composition
 - Element < \sum (parts)
 - Glue
 - Private ports
- Hierarchy implications
 - Component lifetime
 - Data sharing

Incidental Differences



Acme C&C View

- Top-level element
 - Acme System
 - Cannot have ports
- Attachment vs. Binding
 - Binding only from *outer* port/role to *inner port* (or role) respectively

ArchJava C&C View

- Top-level element
 - Component
 - Can have ports
- Missing elements
 - Connector roles
- Unnamed elements
 - Connectors, ...



Our second observation

- Synchronize C&C views incrementally
 - Allow both views to evolve simultaneously
 - Enable architects to work at appropriate level of abstraction
 - Do not require complete code re-generation or complete architectural recovery
- Lightweight and semi-automated
 - Fits into one “wizard” dialog
 - The computer does most of the matching
 - Some manual overrides may be needed

Detection of Structural Differences



- Strategy: automated comparison
- Types of differences
 - Renames
 - Inserts
 - Deletes
 - Moves
- Detection important for maintaining design properties
 - Rename != Delete + Insert
 - Move != Delete + Insert

Synchronization Requirements



- No unique identifiers/labels
- No ordering between view elements
- Support disconnected operation
 - No monitoring of structural edits
- Detect hierarchical moves
- Allow manual overrides
- Type information for optimization only
 - Different levels of abstraction with different type systems

Automated structural comparison

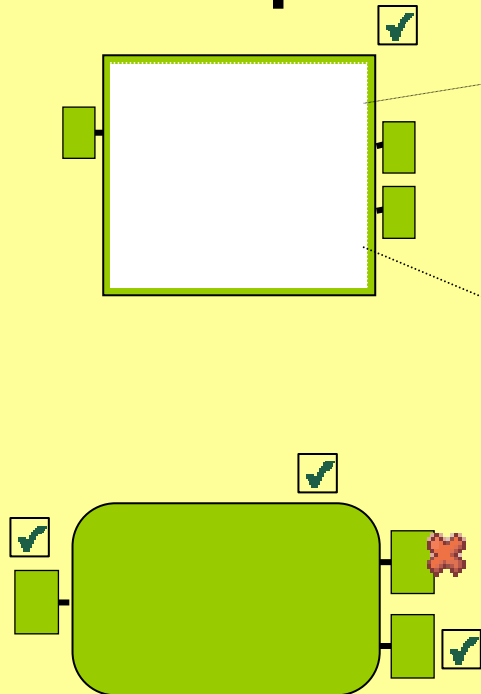


- View represented as a graph
- C&C views as hierarchical views
 - General graph matching – NP complete
 - Take advantage of tree hierarchy and use unordered labeled trees – also NP-Complete
- Assumptions produce polynomial time
 - If two nodes match, so do their parents
 - Changed to be able to detect some “moves”
 - Nodes moved not too far from original position
 - Novel algorithm detects sequences of deletions/insertions in middle of tree

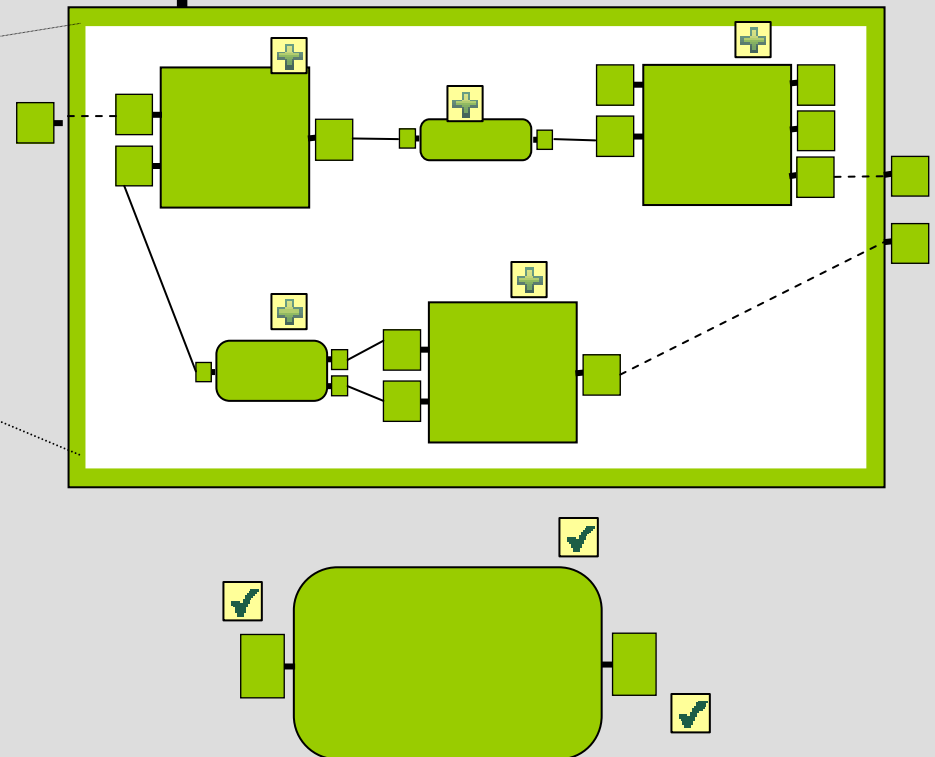
Insert/Delete Differences



Conceptual View



Implementation View

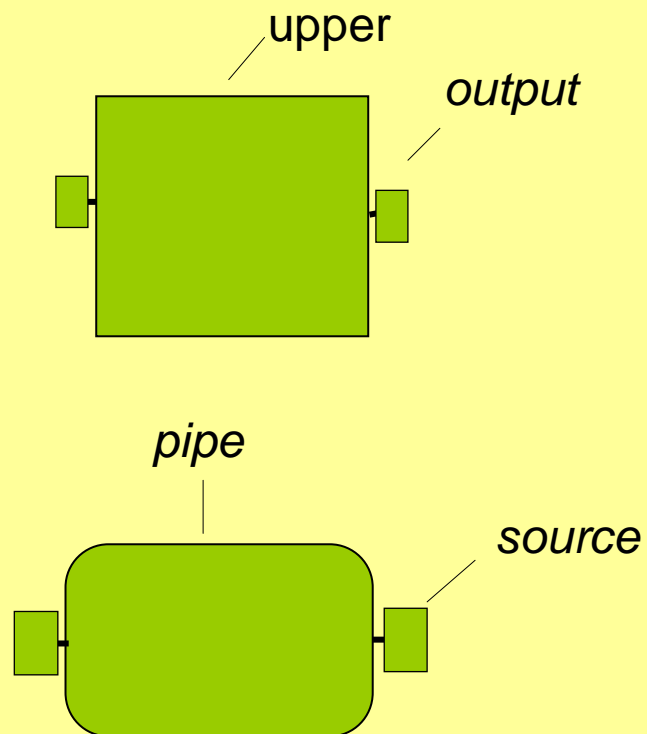


- Insert element in one view (including sub-architecture)
- Delete elements in one view

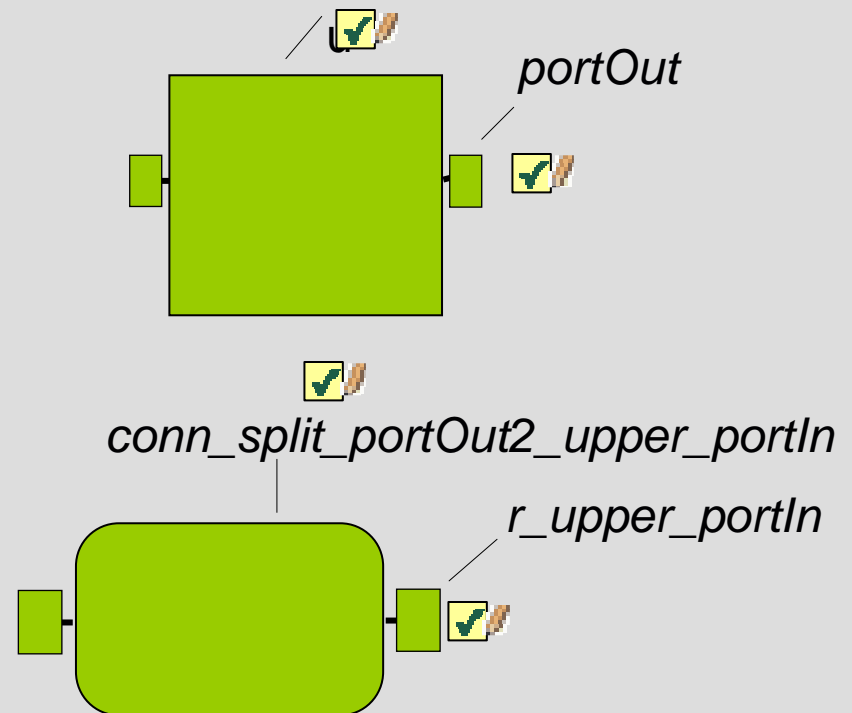
Naming Differences



Conceptual View



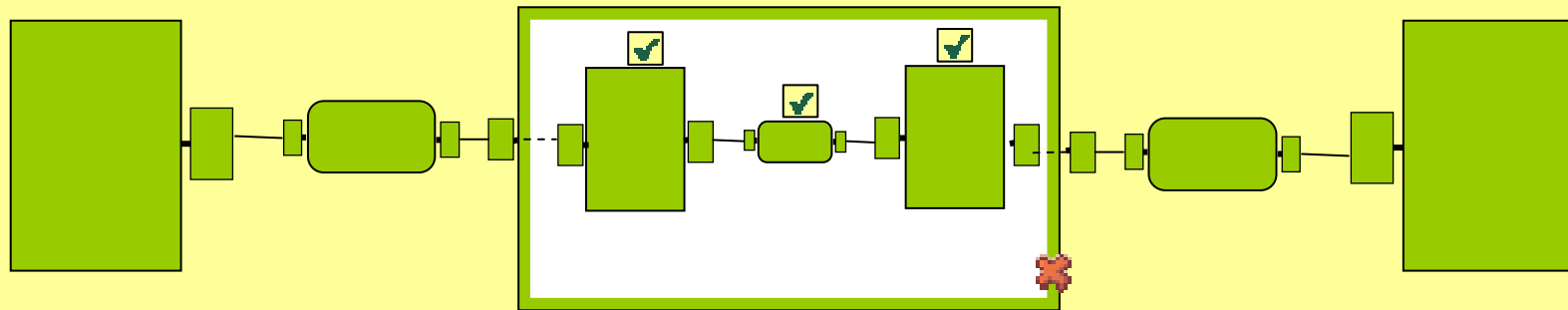
Implementation View



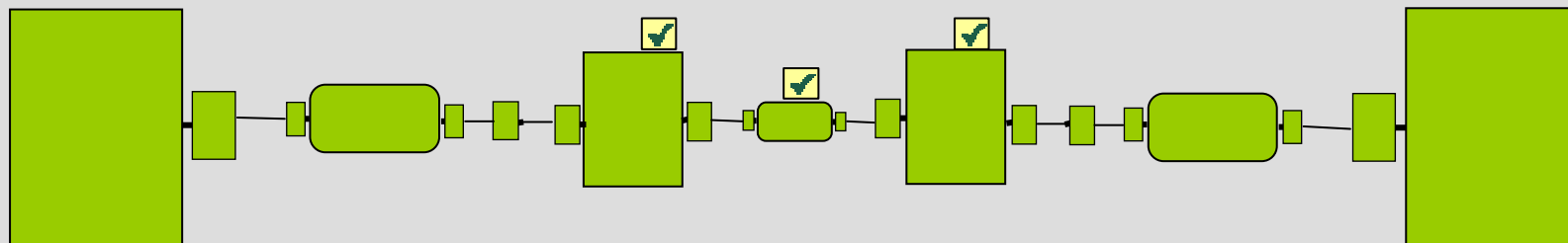
- Incidental renames: no names for connectors, roles, in ArchJava
- Independent evolution: may forget to update other view

Move Differences

Conceptual View



Implementation View



- Restricted move: replace element with its representation
- Other possible moves (not all currently supported)

Synchronization Tool



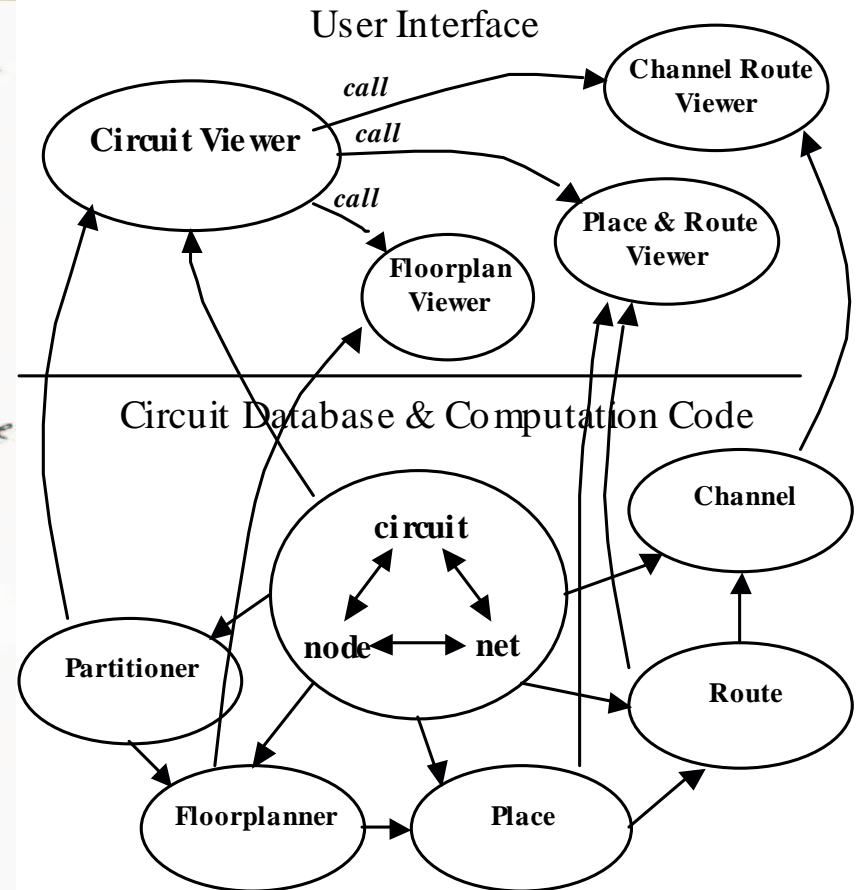
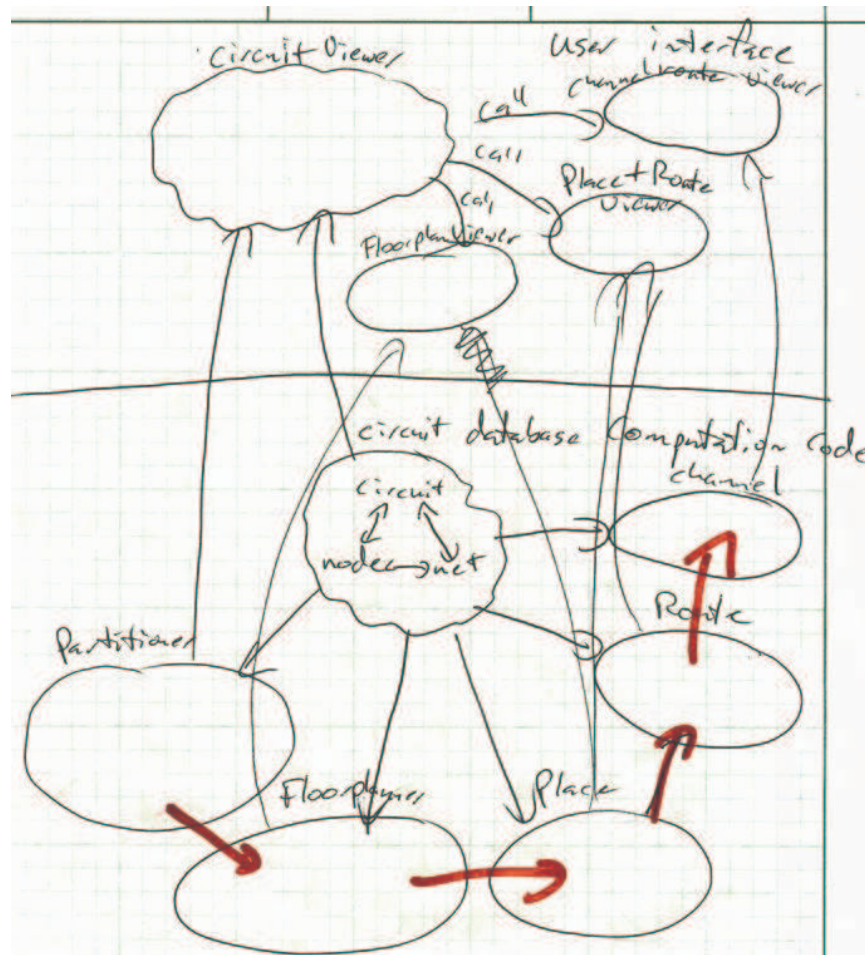
- Step 1: Setup synchronization
- Step 2: View & match types (optional)
- Step 3: View & match instances
- Step 4: View & modify edit script
- Step 5: Confirm & apply edit script (optional)



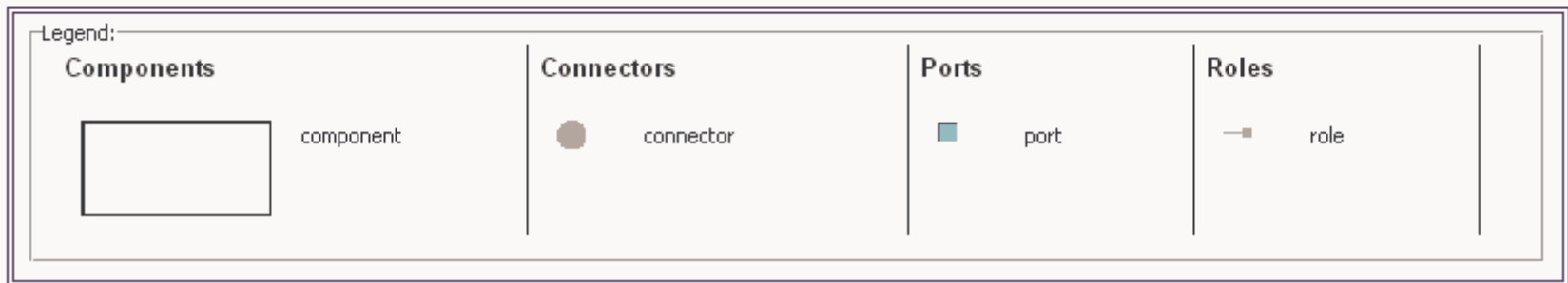
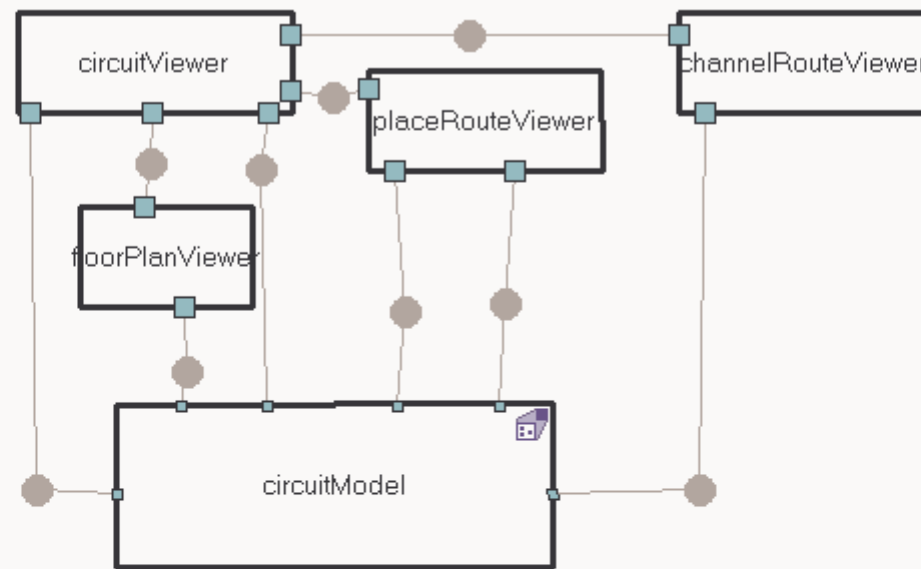
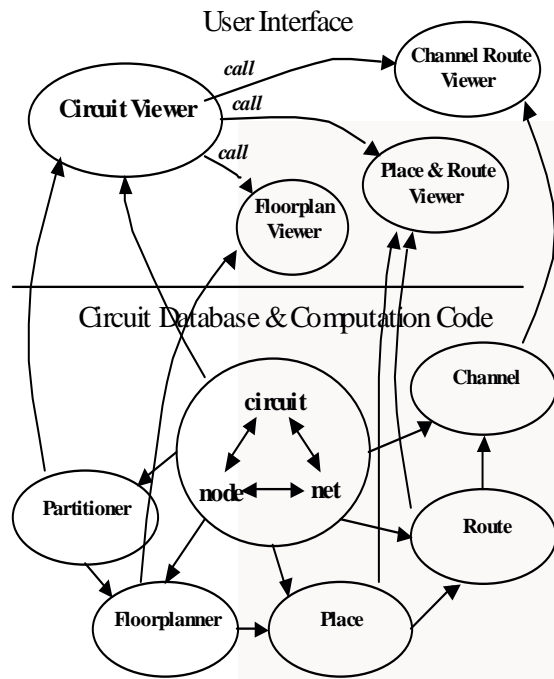
Extended Example: Aphyds

- Pedagogical circuit layout application
- Re-engineered from Java application
 - Over 8 KSLOC
- ArchJava architecture
 - Over 20 components
 - Over 80 ports, several subsystems

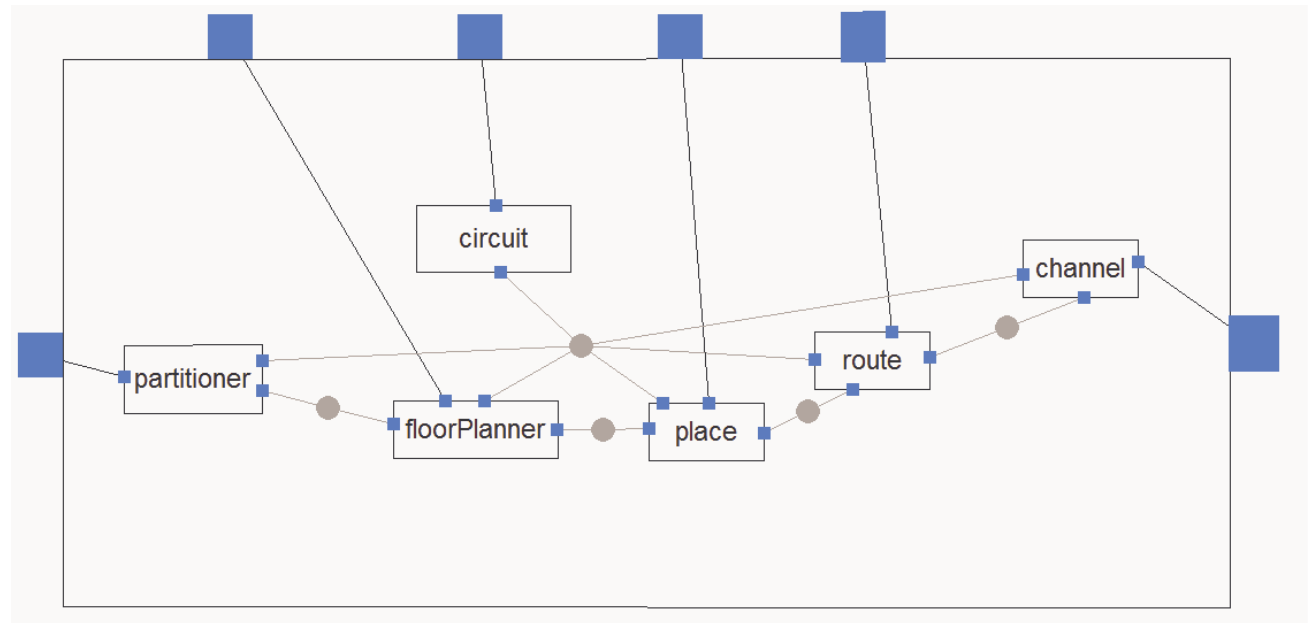
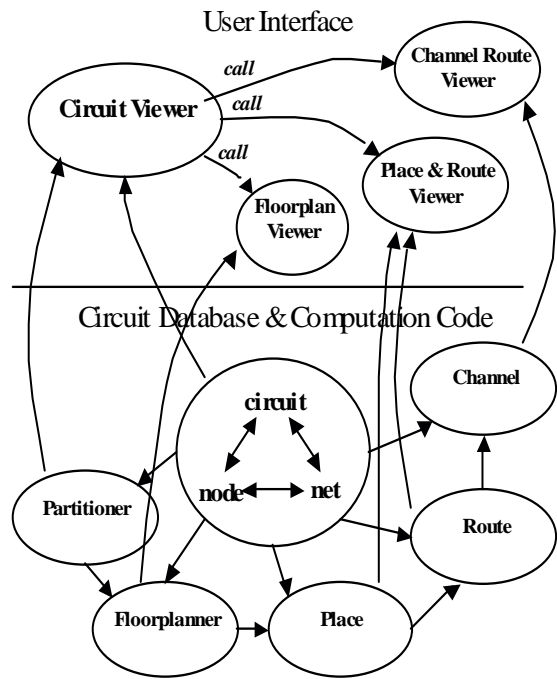
Conceptual-Architecture



Conceptual Architecture in Acme



Conceptual Architecture in Acme



ArchJava: top-level component



```
public component class Aphyds {
  // user interface components
  final owned FloorplanViewer floorplan = ...;
  final owned ChannelRouteViewer channelRoute = ...;
  final owned PlaceRouteViewer placeRoute = ...;
  final owned CircuitViewer viewer = ...;
  // window event communication
  private port window { ... };
  connect window, channelRoute.window, viewer.window, placeRoute.window,
    floorplan.window;
  // command protocol
  connect viewer.command, placeRoute.command, channelRoute.command,
    floorplan.command;
  // model components
  final AphydsModel model = ...;
  // protocols for communication with the model
  connect viewer.circuit, placeRoute.circuit, model.circuit;
  connect viewer.partition, model.partition;
  connect floorplan.floorplan, model.floorplan;
  connect placeRoute.place, viewer.place, model.place;
  connect placeRoute.router, viewer.place, model.router;
  connect channelRoute.channel, model.channels;
  // the program's starting point
  public static void main(String args[]) {
    new Aphyds().run();
  }
  public void run() { viewer.setVisible(true); }
}
```

ArchJava: AphydsModel component

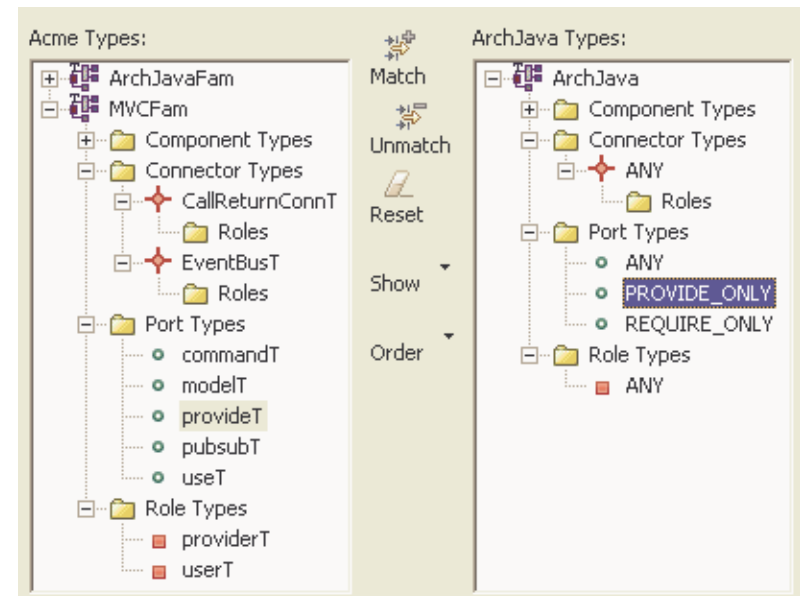


```
public component class AphydsModel {
  final owned Circuit circuitData = ...;
  final owned Partitioner partitioner = ...;
  final owned Floorplanner floorplanner = ...;
  final owned Placer placer = ...;
  final owned GlobalRouter globalRouter = ...;
  final owned ChannelRouter channelRouter = ...;
  public port place { ... }
  public port partition { ... }
  public port floorplan { ... }
  public port circuit { ... }
  public port router { ... }
  public port channels { ... }
  connect circuit, partitioner.circuit, floorplanner.circuit,
    placer.circuit,
      globalRouter.circuit, circuitData.main, channelRouter.circuit;
  connect place, globalRouter.place, placer.place;
  connect partition, partitioner.partition;
  connect floorplan, floorplanner.floorplan;
  connect router, globalRouter.router;
  connect channels, channelRouter.channels;
}
```

Matching Types



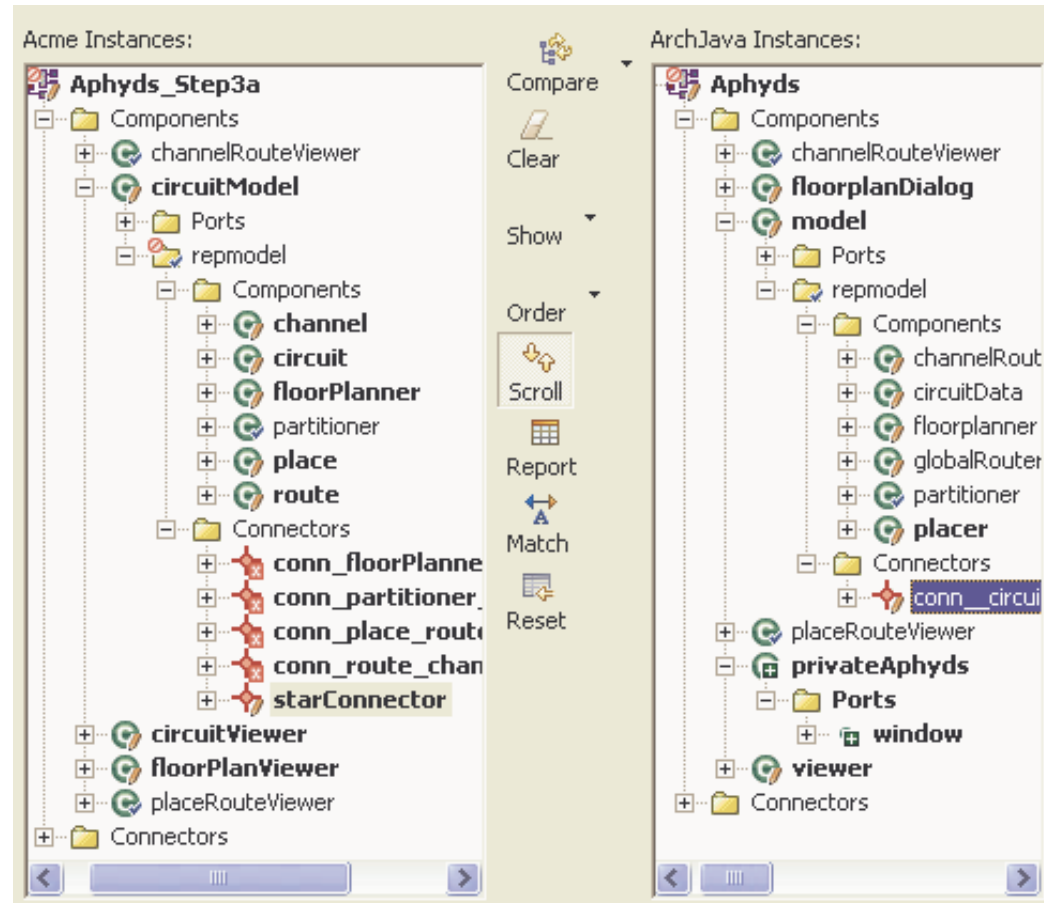
- Different scenarios:
 - Match explicit types if available
 - Assign types to instances when no explicit type
 - Special wildcards
 - Infer types when possible, using style information



Matching Instances



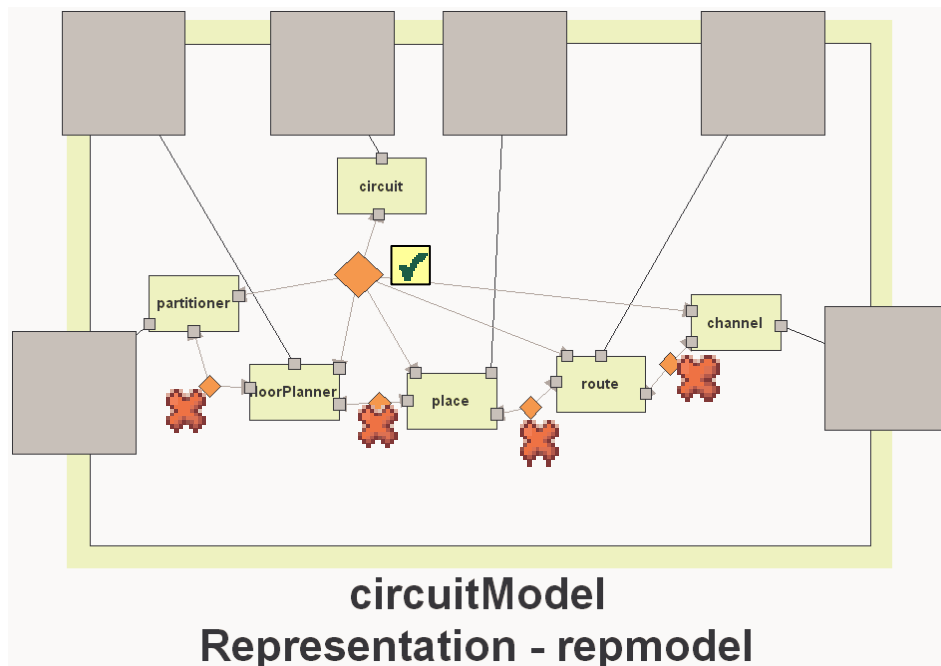
- Detect
 - Match
 - Insert
 - Delete
 - Rename
 - Move



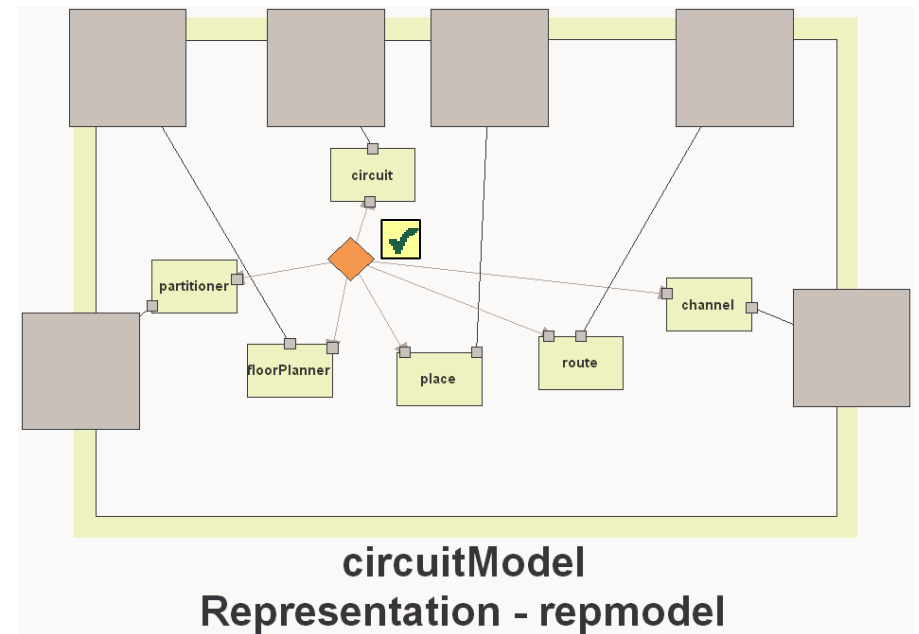
Divergence1: extra connectors



The “data flow” connectors in the original architect’s informal diagram do not exist!

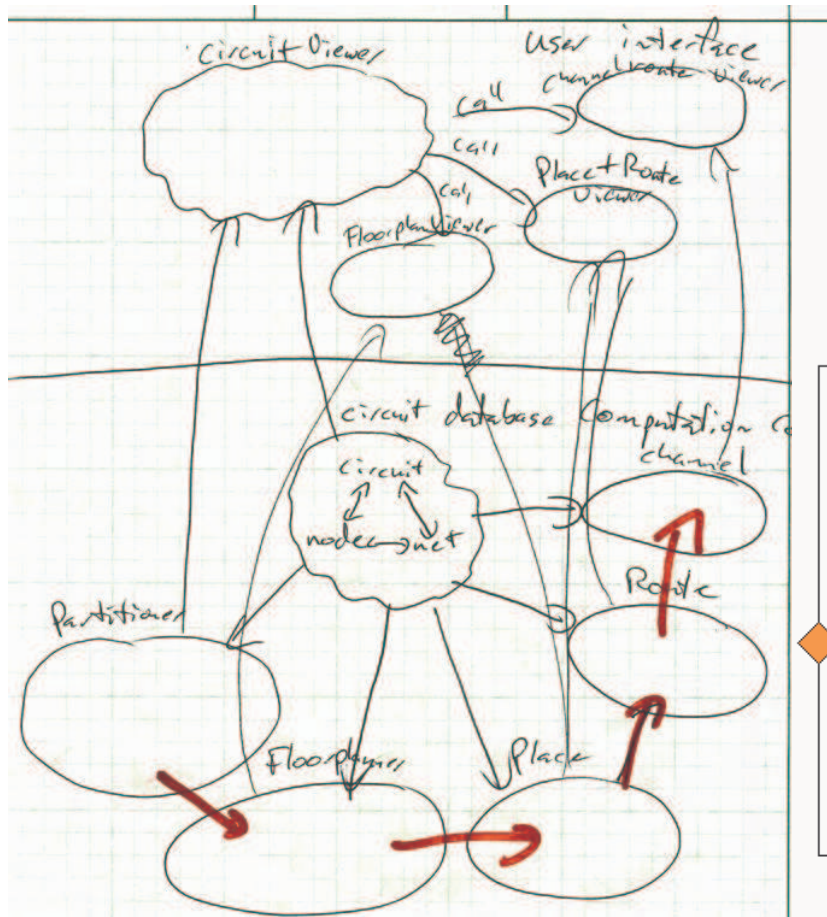


Before

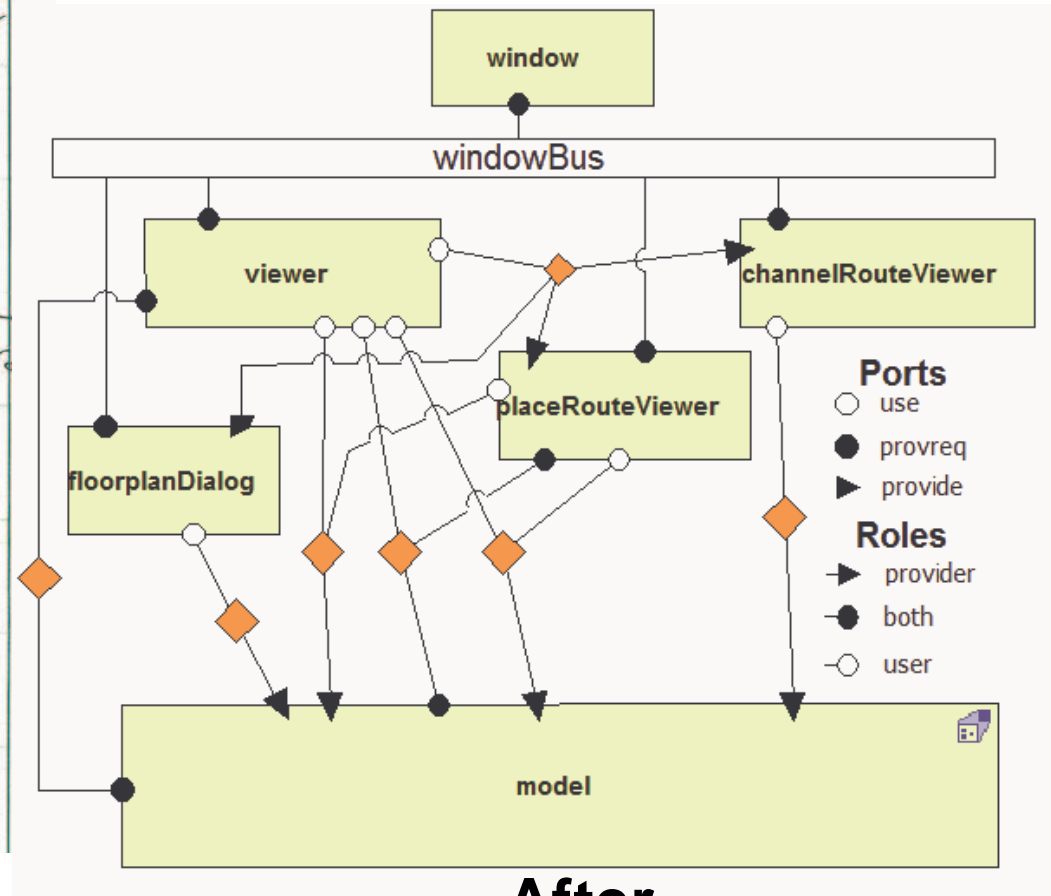


After

Divergence2: missing sub-system



Before

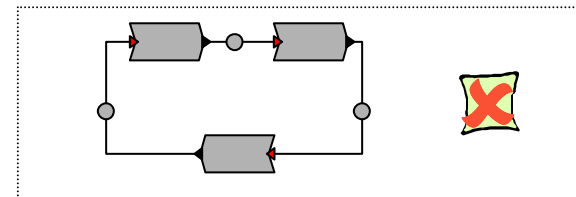
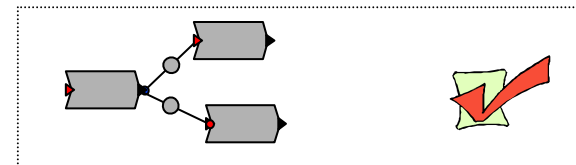
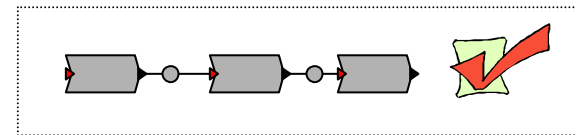
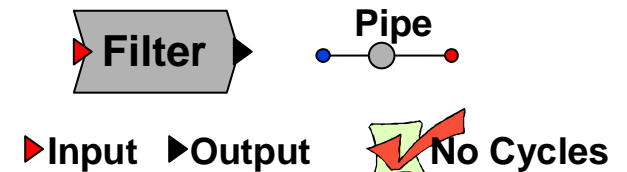


After

Detecting style violations



- Setting architectural types and styles on up-to-date conceptual-level architecture
- Checking conformance to style
 - Constraints
 - Example: no cycles in a pipe-and-filter system





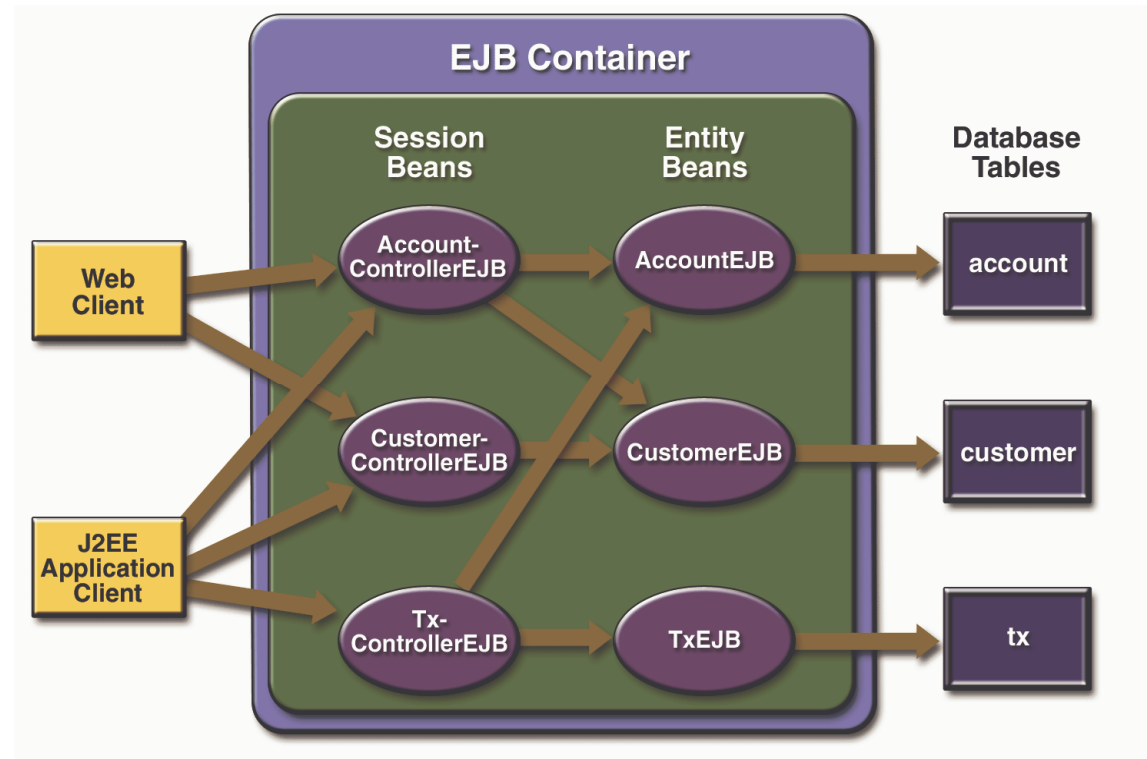
Other uses

- Generalized to differencing and merging any two C&C Views
- Implementation-level view could be recovered using a variety of architectural recovery techniques
 - E.g., instrumenting running system

Example: Duke's Bank Application

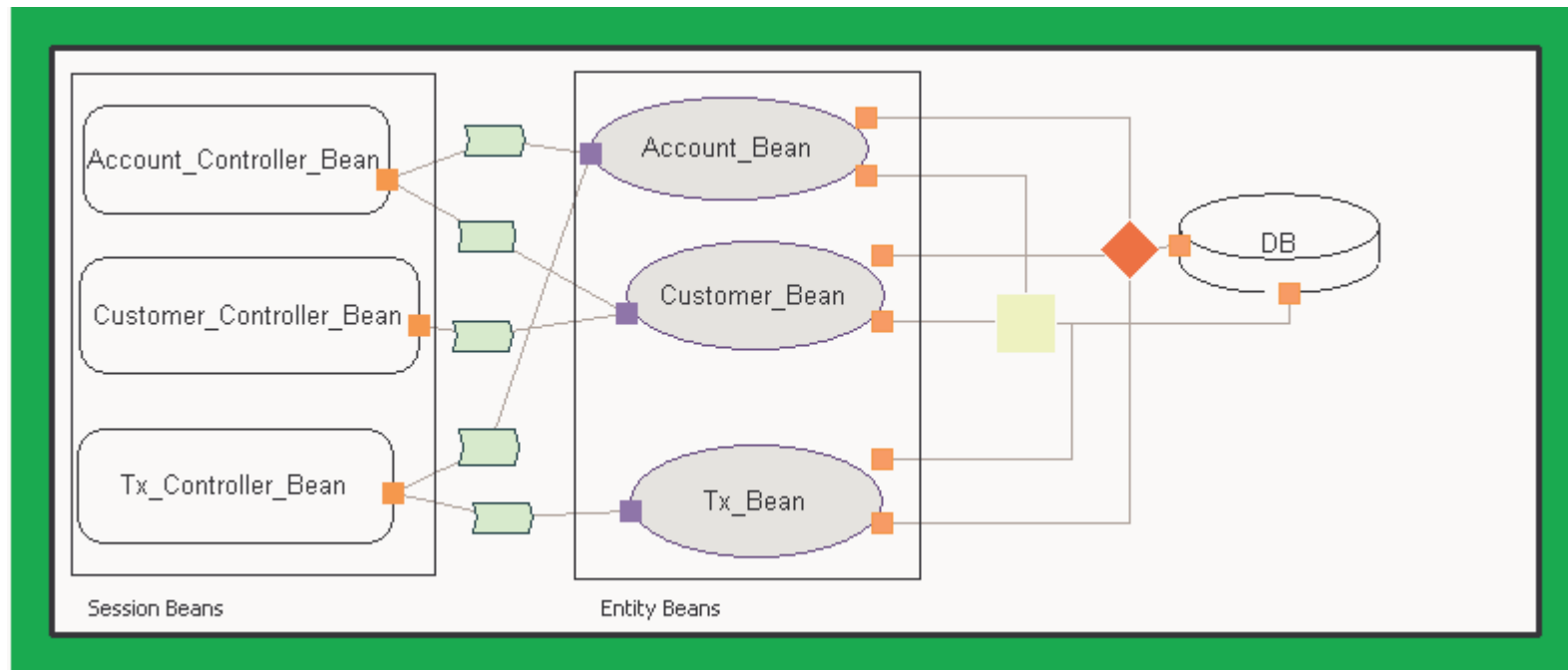


- Simple (EJB) banking application
- Documented architecture



Duke's Bank: documented architecture

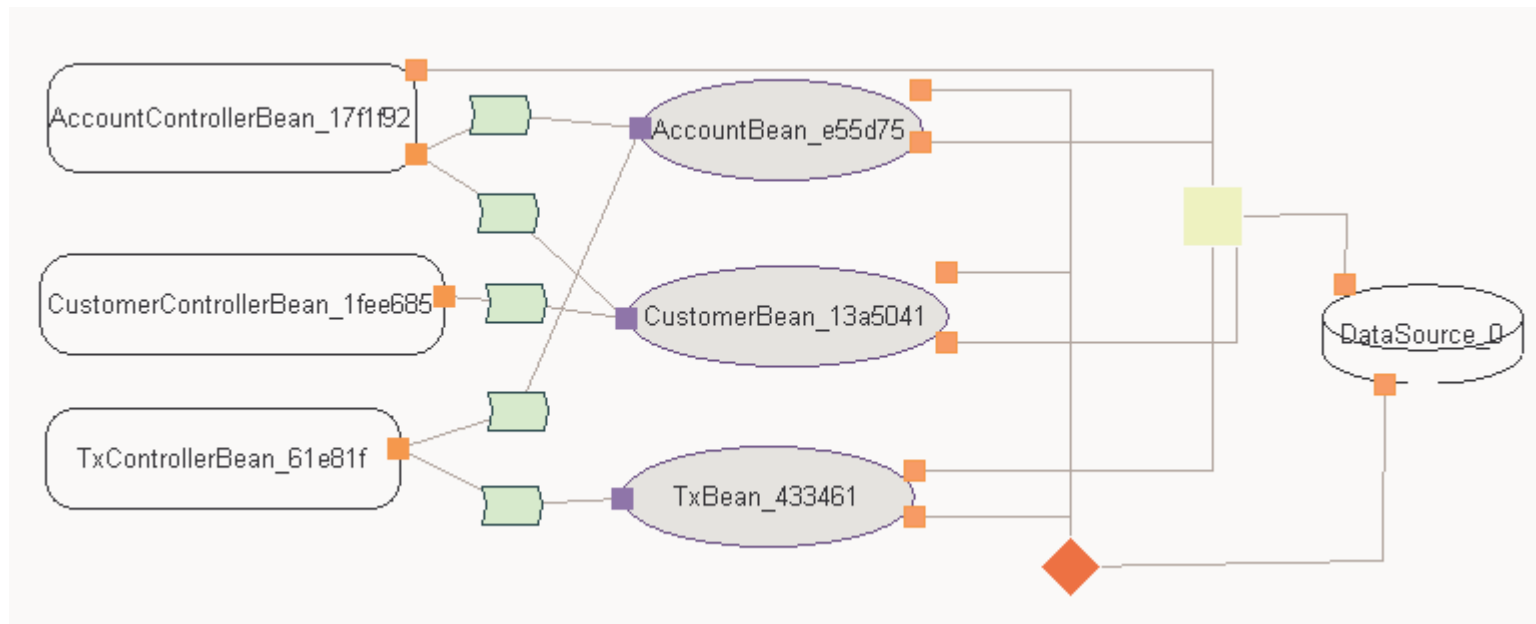
- Defined Acme family (or style) and types based on the EJB specification.



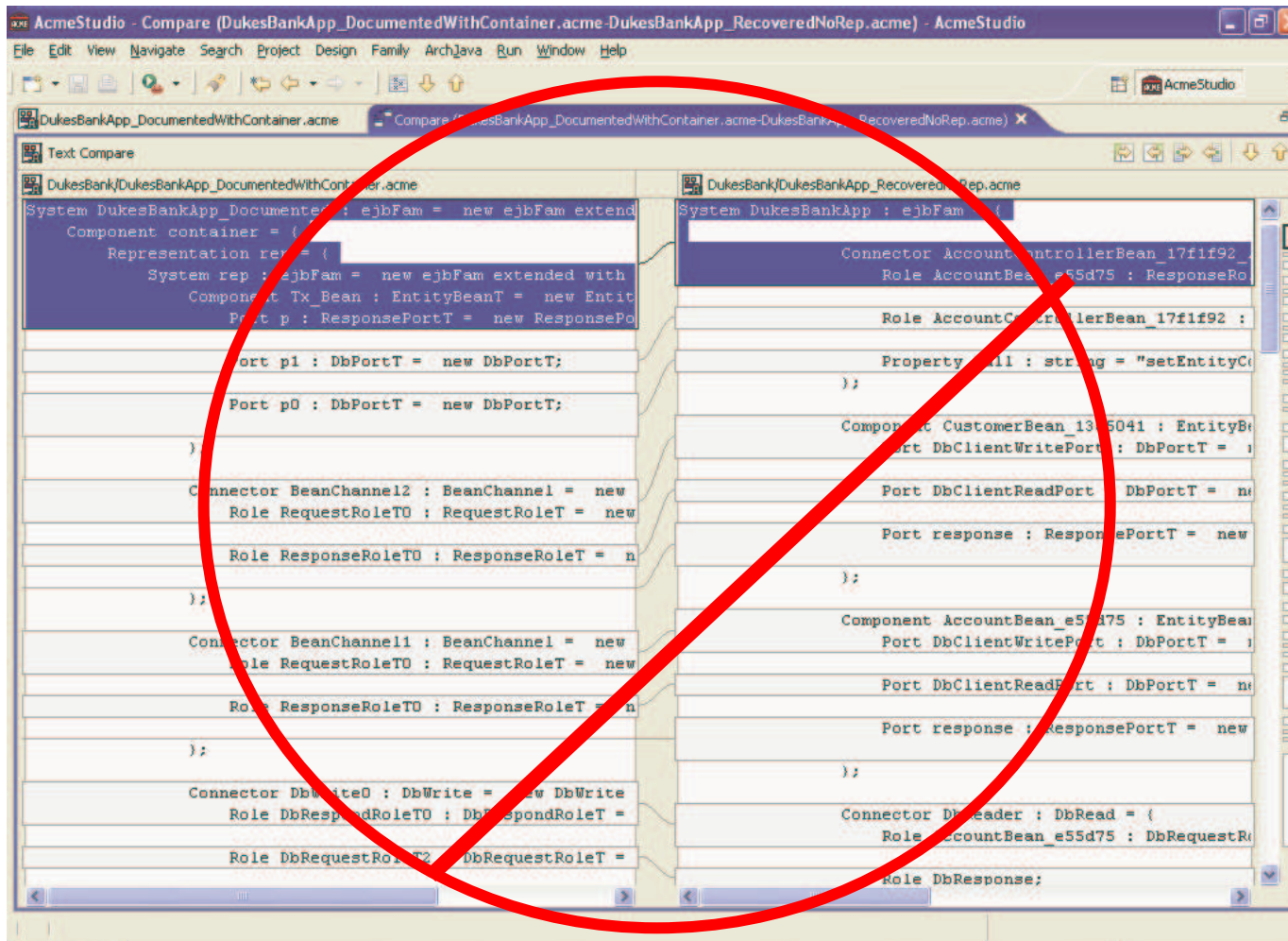
Duke's Bank: recovered architecture



- Recovered by instrumenting running system (using DiscoTect)
- Post-processed to eliminate duplicates



Textual comparison does not work!



Detecting renames



Synchronize two Acme models

Step 3 of 5: Match instances

Please check the Messages tab: 0 error(s), 17 warning(s) found.

Instances:

- DukesBankApp_Documented
 - Components
 - container
 - Ports
 - rep
 - Components
 - Account_Bean
 - Account_Controller_Bean
 - Customer_Bean
 - Customer_Controller_Bean
 - DB
 - Tx_Bean
 - Tx_Controller_Bean
 - Connectors
 - BeanChannel0
 - BeanChannel1
 - BeanChannel2
 - BeanChannel4
 - BeanChannel5
 - DbRead0
 - DbWrite0

Instances:

- DukesBankApp
 - Components
 - AccountBean_e55d75
 - AccountControllerBean_17f1f92
 - CustomerBean_13a5041
 - CustomerControllerBean_1fee685
 - DataSource_0
 - TxBean_433461
 - TxControllerBean_61e81f
 - Connectors
 - AccountControllerBean_17ca8c8_CustomerBean_5e0651
 - AccountControllerBean_17f1f92_AccountBean_e55d75
 - CustomerControllerBean_1fee685_CustomerBean_13a5041
 - DbReader
 - DbWriter
 - TxControllerBean_61e81f_AccountBean_e55d75
 - TxControllerBean_61e81f_TxBean_433461

Properties:

Name	Value
AcmeFamilyType	ejbFam.EntityBeanT

Properties:

Name	Value
AcmeFamilyType	ejbFam.EntityBeanT

Detects Architectural Violations



The screenshot shows a software window titled "Synchronize two Acme models". The window is divided into several sections. At the top, it says "Step 3 of 5: Match instances" and includes a message: "Please check the Messages tab: 0 error(s), 17 warning(s) found." Below this, there are two side-by-side tree views labeled "Instances:". The left tree view shows a hierarchical structure starting with "Account_Controller_Bean", which contains "Ports", "Attaches", and "Binds". Under "Attaches", there are "Account_Bean_p", "Account_Bean", "BeanChannel1", "RequestRoleT0", "Customer_Bean_p", "Customer_Bean", and "BeanChannel4". Under "Binds", there are "Customer_Bean", "Customer_Controller_Bean", "DB", "Tx_Bean", and "Tx_Controller_Bean". The right tree view shows a similar structure for "AccountControllerBean_17f1f92", with "Ports" containing "p", "Attaches" containing "AccountBean_e55d75_DbCli", "CustomerBean_13a5041_Db", "DataSource_0_DbServer", and "TxBean_433461_DbClier", and "Binds" containing "request", "Attaches" (with "AccountBean_e55d75_r", "CustomerBean_13a5041"), and "Binds" (with "CustomerBean_13a5041", "CustomerControllerBean_1fee685", "DataSource_0", "TxBean_433461", and "TxControllerBean_61e81f"). A central toolbar contains icons for "Compare", "Clear", "Show", "Order", "Scroll", "Report", "Match", and "Reset". At the bottom, there are tabs for "Properties", "Messages", and "Search".



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



Open Questions

- How can we reason more carefully about differences between these views?
- How to streamline the two representations to make full-round-trip synchronization a reality
- What are other structural differences that would be valuable to detect?
 - Splitting and merging?
 - Others?
- Can we apply the same approach to other hierarchical architectural views?
- Other ways of enforcing conceptual-level architecture directly in the source code?



References

- [ACN02] Aldrich, J., Chambers, C. and Notkin, D. ArchJava: Connecting Software Architecture to Implementation. In Proc. ICSE, 2002.
- [GMW00] Garlan, D., Monroe, R., and Wile, D. Acme: Architectural Description of Component-Based Systems. In Foundations of Component-Based Systems, Cambridge University Press, 2000.
- [THP05] Torsello, A., Hidovic-Rowe, D. and Pelillo, M. Polynomial-Time Metrics for Attributed Trees. IEEE Trans. on Pattern Analysis and Machine Intelligence, 27 (7), 2005.
- [WH02] van der Westhuizen, C. and van der Hoek, A. Understanding and Propagating Architectural Changes. In Proc. WICSA 3, 2002.