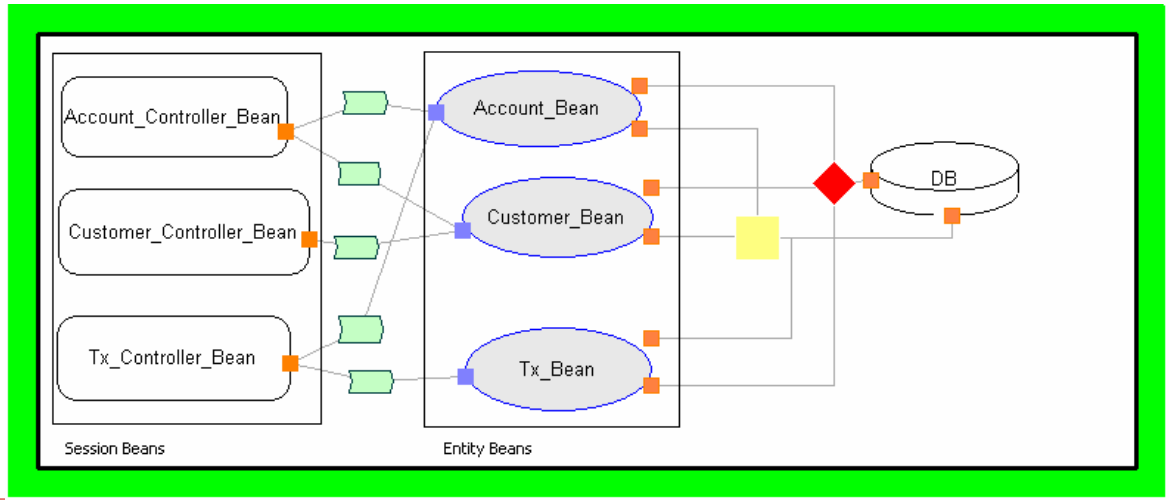
Differencing and Merging of Architectural Views

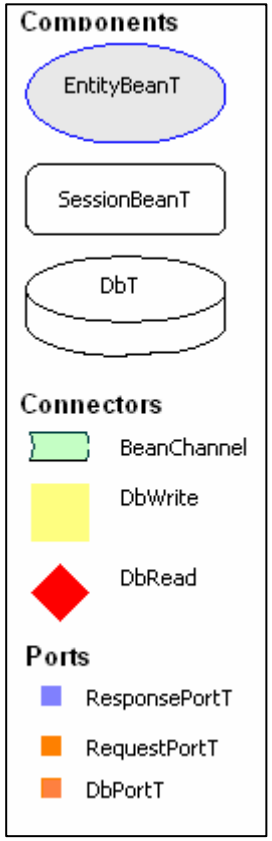
Marwan Abi-Antoun Jonathan Aldrich
Nagi Nahas
David Garlan Bradley Schmerl

Institute for Software Research
Carnegie Mellon University

Software Architectures

- Help reason about software at abstract level
- Runtime organization of system
  - Components (e.g., DataBase)
  - Connectors (e.g., DbWrite)
  - Properties (e.g., IsSynchronized = true)

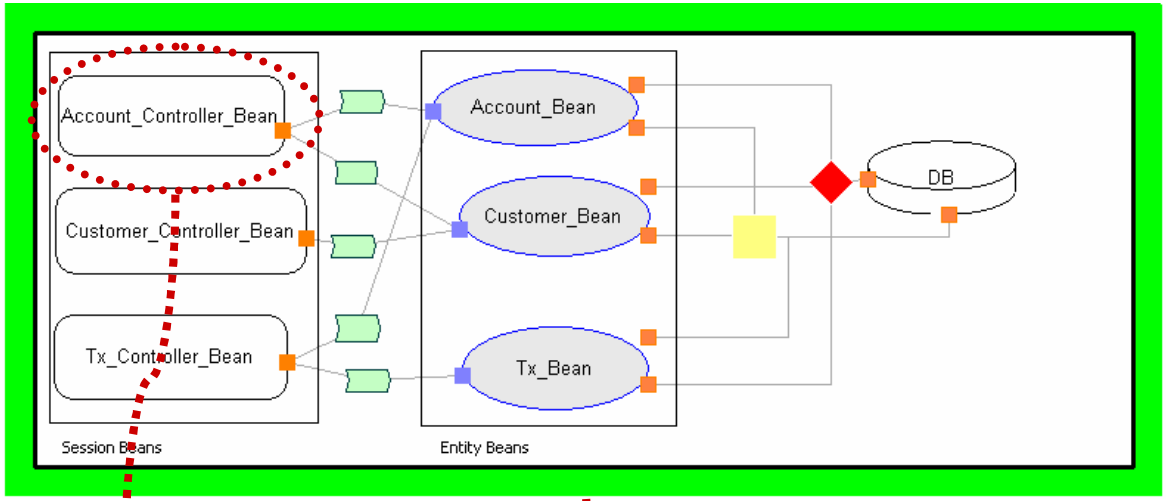


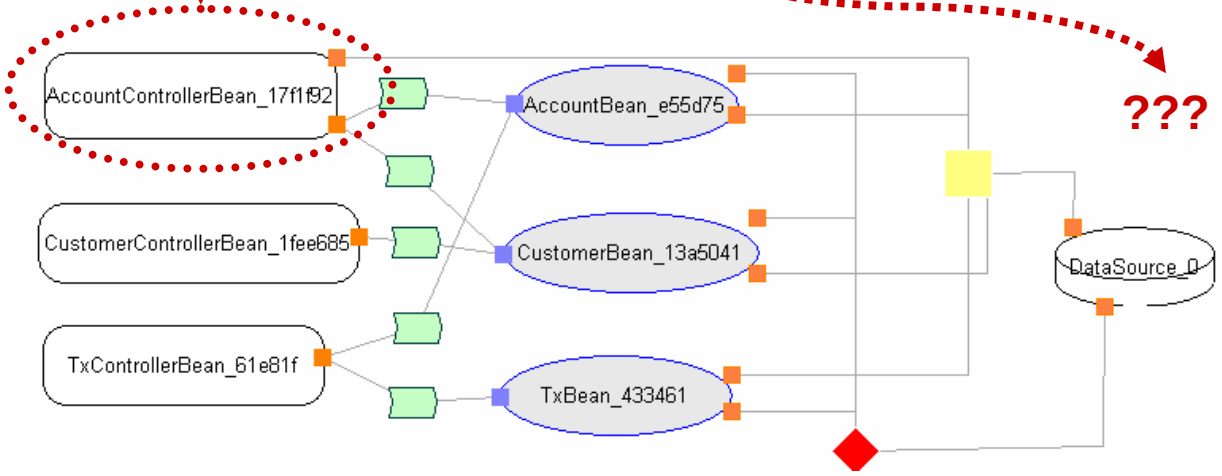


Are these two views the same?

- Renames
- Extra level of hierarchy
  - “Move”
- Insertions
- Deletions

As-designed system





As-built system

Need for View Comparison

- Views evolve independently
- Synchronize two versions
- Compare two variants in product line
- Compare as-designed with as-built view
  - Look for architectural violations
  - Perform change impact analysis

View Comparison Problem

- General graph matching
  - NP-complete problem
- View comparison tradeoffs
  - Assumptions (post-hoc vs. not)
  - Efficiency (exponential vs. polynomial)
  - Accuracy
- Ideal: detect as many changes as possible
  - Rename, insert, delete, move, merge, split…
  - ArchDiff: insertions and deletions, no renames

Possible Assumptions

- Monitoring of structural edits
  - Does not handle legacy models
  - Requires built-in tool support
- Assume unique identifiers or labels
  - Makes problem simpler
  - IDs or Persistent Names may not exist!
- Heuristic-based approaches
  - Assume majority of nodes exactly match
  - Cannot recover information from structure

Efficiency using tree algorithms

- Many architectural views hierarchical
- Hierarchy enables using tree algorithms
- Tree algorithms also NP-complete
- Assumptions produce polynomial time
- THP: If two nodes match, so do their parents

Torsello, A., Hidovic-Rowe, D. and Pelillo, M. Polynomial-Time Metrics for Attributed Trees. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27 (7), 2005.

Detecting renames and moves

- Treating rename or move as insert/delete
  - Produce structurally equivalent views
  - But lose properties associated with elements
- MDIR: Detect hierarchical moves
  - Replacing an abstraction with its contents
  - Move = inserts/deletes in middle of the tree
  - Constraint: nodes not moved too far from their original positions in a hierarchy

Our Contributions

- Use structural information for hierarchical view comparison
- Designed novel algorithm (MDIR)
  - Extends published algorithm (THP)
  - Detects moves
- Incorporated algorithms in a set of tools
- Evaluated tools in case studies
- Found the tools to be useful

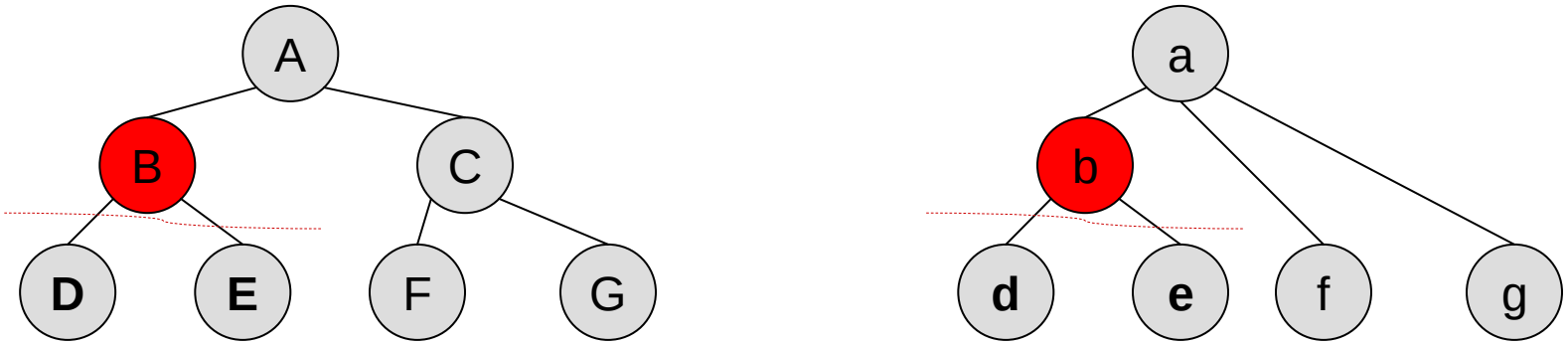
Outline

- Ø MDIR algorithm
- Tools
- Case studies

MDIR Features

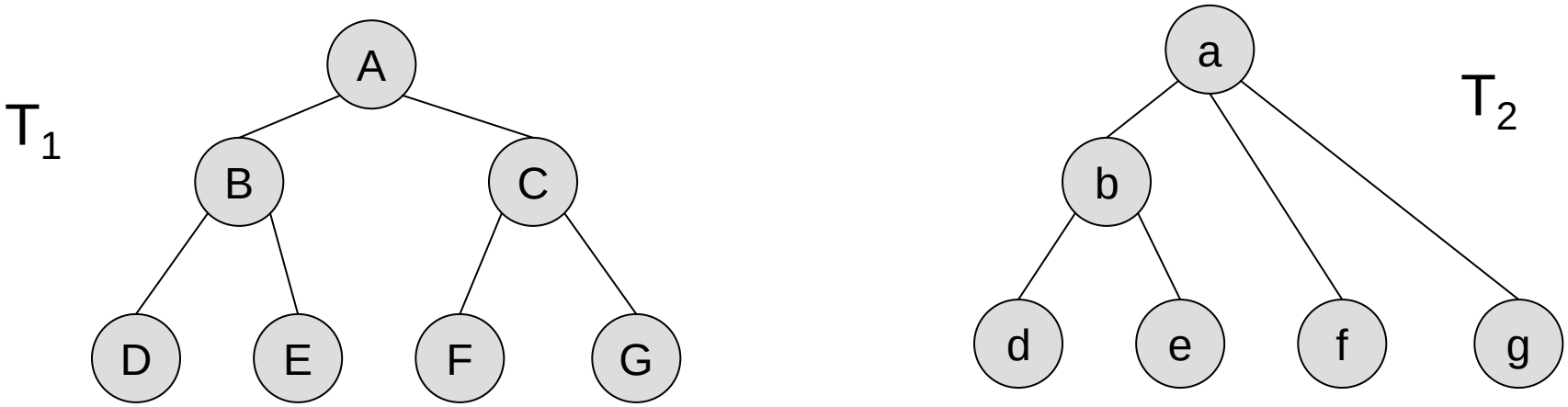
- Detect inserts and deletes
- Detect renames and moves
  - Not treating as insert + delete
  - Preserve architectural properties
- Allow optional manual overrides
  - Force matches between two nodes
  - Prevent matches between two nodes
- Type information optional

Definition: Successor Set of (B, b)



- Take all node pairs where first item descendent of B and second item descendent of b:
  - All pairs for (B,b) = { (D,d), (D,e), (E,d), (E,e) }
- Successor set is subset that obeys conditions:
  1. If (x, y) in set, then ascendants and descendents of x and y cannot occur in any other pair in successor set
  2. If (x, y) in set, neither x nor y can re-appear in pair in set
- Successor set of **(B,b) = { (D,d), (E,e) }**
  - (D,e) and (E, d) excluded because D and E in pairs in set

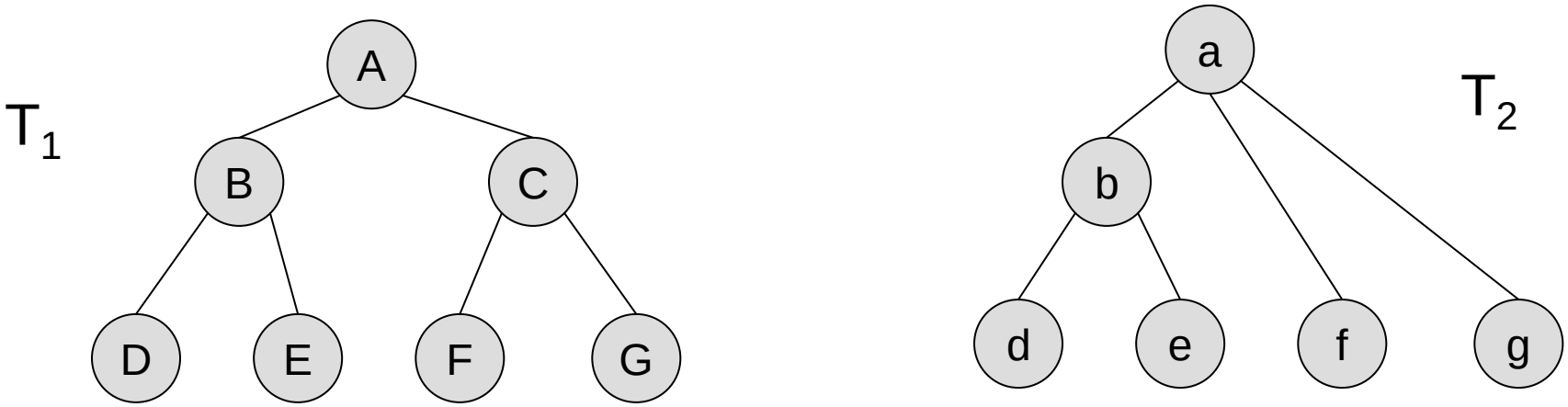
MDIR: high-level intuition



(D, d) =…	(D, e) =…	(D, f) =…	(D, g) =…	(D, b) =…	(D, a) =…
(E, d) = …	(E, e) = …	(E, f) = …	(E, g) = …	(E, b) =…	(E, a) =…
…	…	…	…	…	…

- Post-order nodes in trees T_1 and T_2
- Exhaustively search from bottom to top
- Cost of mapping each node in T1 to every other node in T2

MDIR: cost of matching D to d

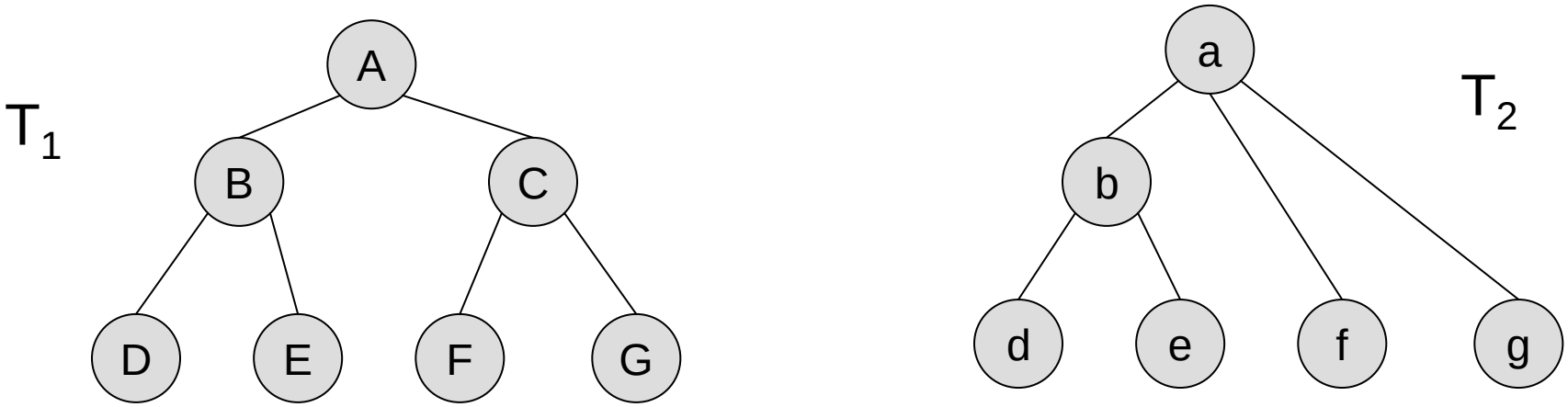


(D, d) = 0	(D, e) = 1	(D, f) = 2	(D, g) =3	(D, b) =…	(D, a) =…
(E, d) = 1	(E, e) = 0	(E, f) = 1	(E, g) = 2	(E, b) =…	(E, a) =…
…	…	…	…	…	…

Cost of editing label = measure of similarity between labels

(D, d) = cost(editing label of D to d)
= 0

MDIR: cost of matching B to d



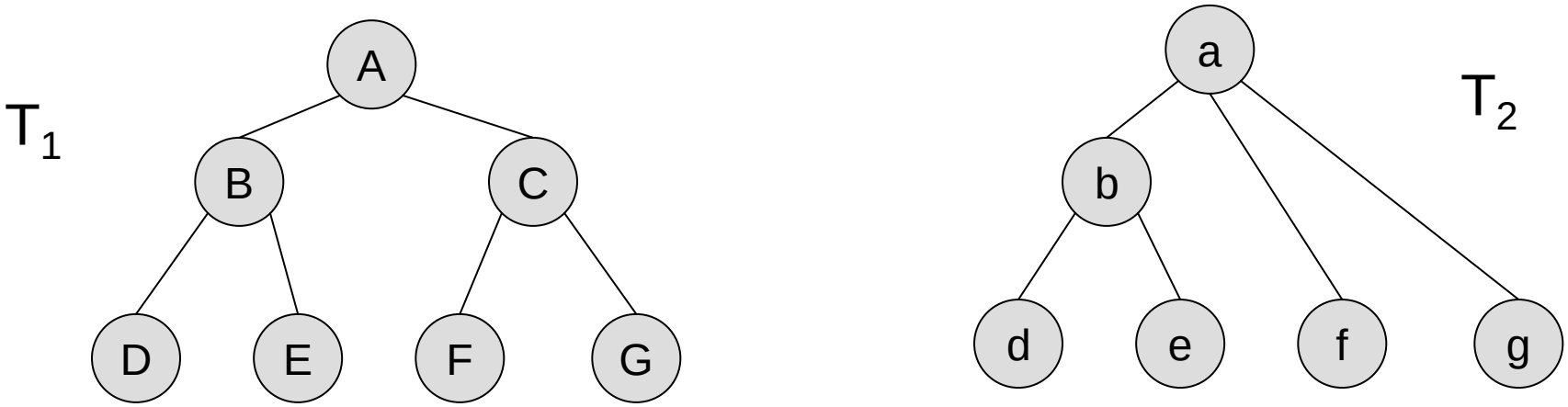
(D, d) = 0	(D, e) = 0	(D, f) = 2	(D, g) =3	(D, b) =…	(D, a) =…
(E, d) = 1	(E, e) = 0	(E, f) = 1	(E, g) = 2	(E, b) =…	(E, a) =…
…	…	…	…	…	…
(B, d) = 12	(B, e) = …	(B, f) = …	(B, g) = …	(B, b) =…	(B, a) =…

(B, d) = cost(deleting children of B) + cost(editing label of B)

= cost(deleting D) + cost(deleting E) + cost(editing label of B)

= 5 + 5 + 2

MDIR: cost of matching B to b



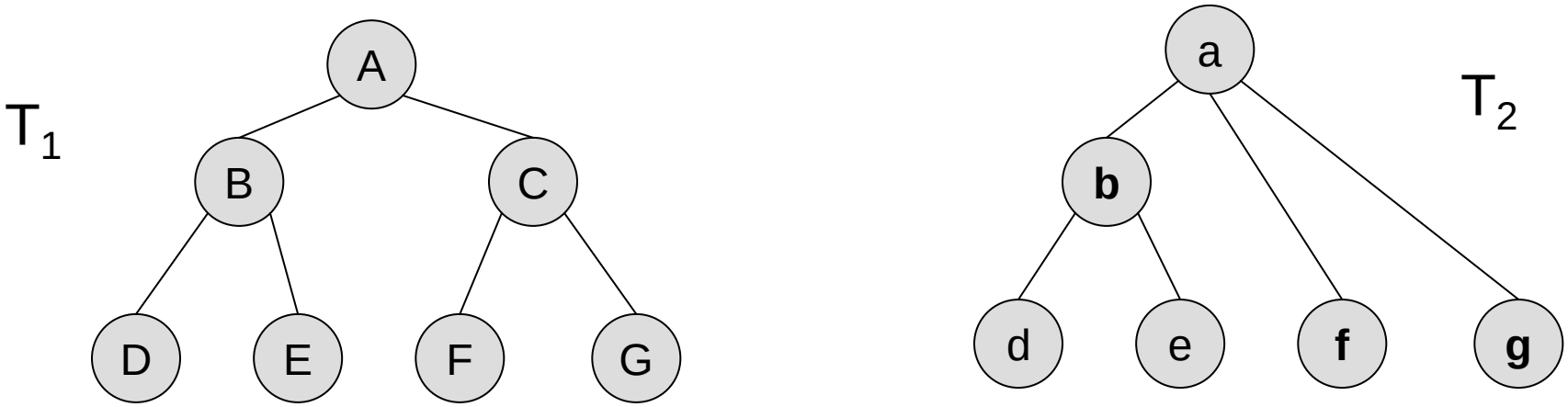
(D, d) = 0	(D, e) = 1	(D, f) = 2	(D, g) =3	(D, b) =…	(D, a) =…
(E, d) = 1	**(E, e) = 0**	(E, f) = 1	(E, g) = 2	(E, b) =…	(E, a) =…
…	…	…	…	…	…
(B, d) = 12	(B, e) = …	(B, f) = …	(B, g) = …	(B, b) = 0	(B, a) =…

Use successor set of (B, b) = { (D, d), (E, e) }

(B, b) = cost(successor set of (B,b)) + cost(editing label of B to b)

= **cost(D,d)** + **cost(E,e)** + 0 = 0

MDIR: cost of matching B to a

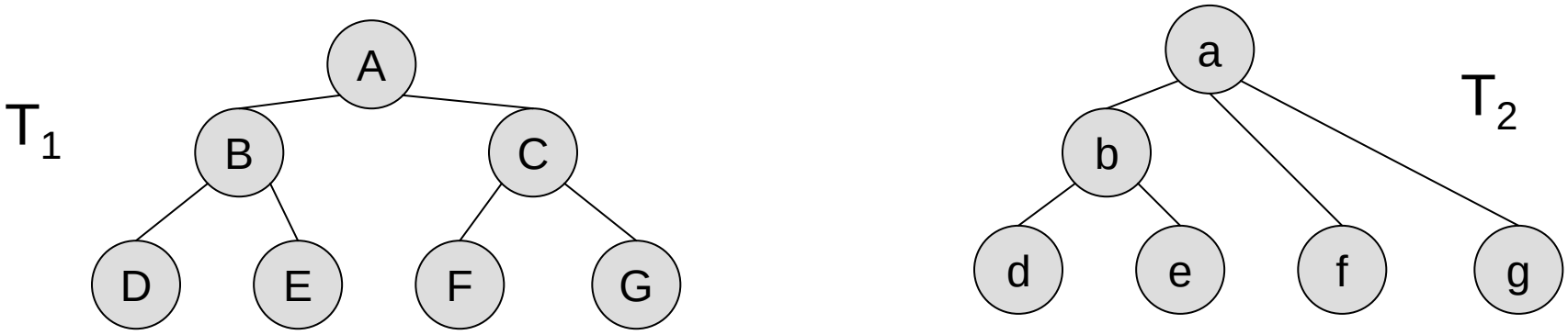


(D, d) = 0	(D, e) = 1	(D, f) = 2	(D, g) =3	(D, b) =…	(D, a) =…
(E, d) = 1	**(E, e) = 0**	(E, f) = 1	(E, g) = 2	(E, b) =…	(E, a) =…
…	…	…	…	…	…
(B, d) = 12	(B, e) =…	(B, f) =…	(B, g) =…	(B, b) = 0	(B, a) = ?

Use successor set of (B, a) = { (D, d), (E, e) }

(B, a) = cost(successor set of (B, a)) + cost(editing label of B to a)
+ cost(deleting **b**, **f** and **g**)

MDIR: finding “best” successor sets



(D, d) = 0	(D, e) = 1	(D, f) = 2	(D, g) =3	(D, b) = ?	(D, a) = ?
(E, d) = 1	(E, e) = 0	(E, f) = 1	(E, g) = 2	(E, b) = ?	(E, a) = ?
…	…	…	…	…	…
(B, d) = 12	(B, e) = …	(B, f) = …	(B, g) = …	(B, b) = 0	(B, a) = ?
(C, d) = ?	(C, e) = ?	(C, f) = ?	(C, g) = ?	(C, b) = ?	(C, a) = ?
(A, d) = ?	(A, e) = ?	(A, f) = ?	(A, g) = ?	(A, b) = ?	(A, a) = ?

- Compute cost of each successor set for pair of nodes
- Determine the “best” successor set
- Store it for next phase (to retrieve the best matches)

MDIR: Summary

- 1st Phase: Compute costs of successor sets
  - Dynamic programming: results of comparing “lower” nodes used to compare “higher” nodes
  - Branch-and-bound: exhaustive search made faster using sorting (early pruning of branches) and using hierarchical constraints as early as possible
- 2nd Phase: Retrieve best matching
- Pseudo-code in paper
- Additional details in technical report

Outline

- MDIR algorithm
- Ø Tools
- Case studies

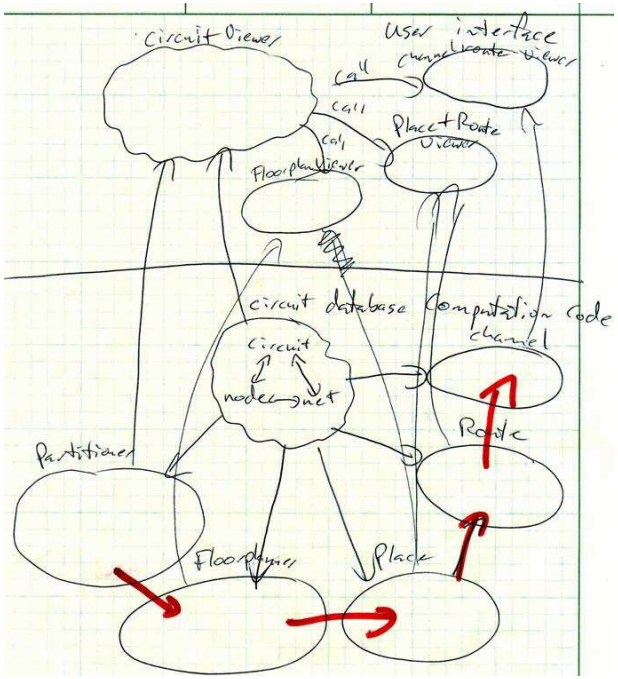
View Differencing and Merging Tools

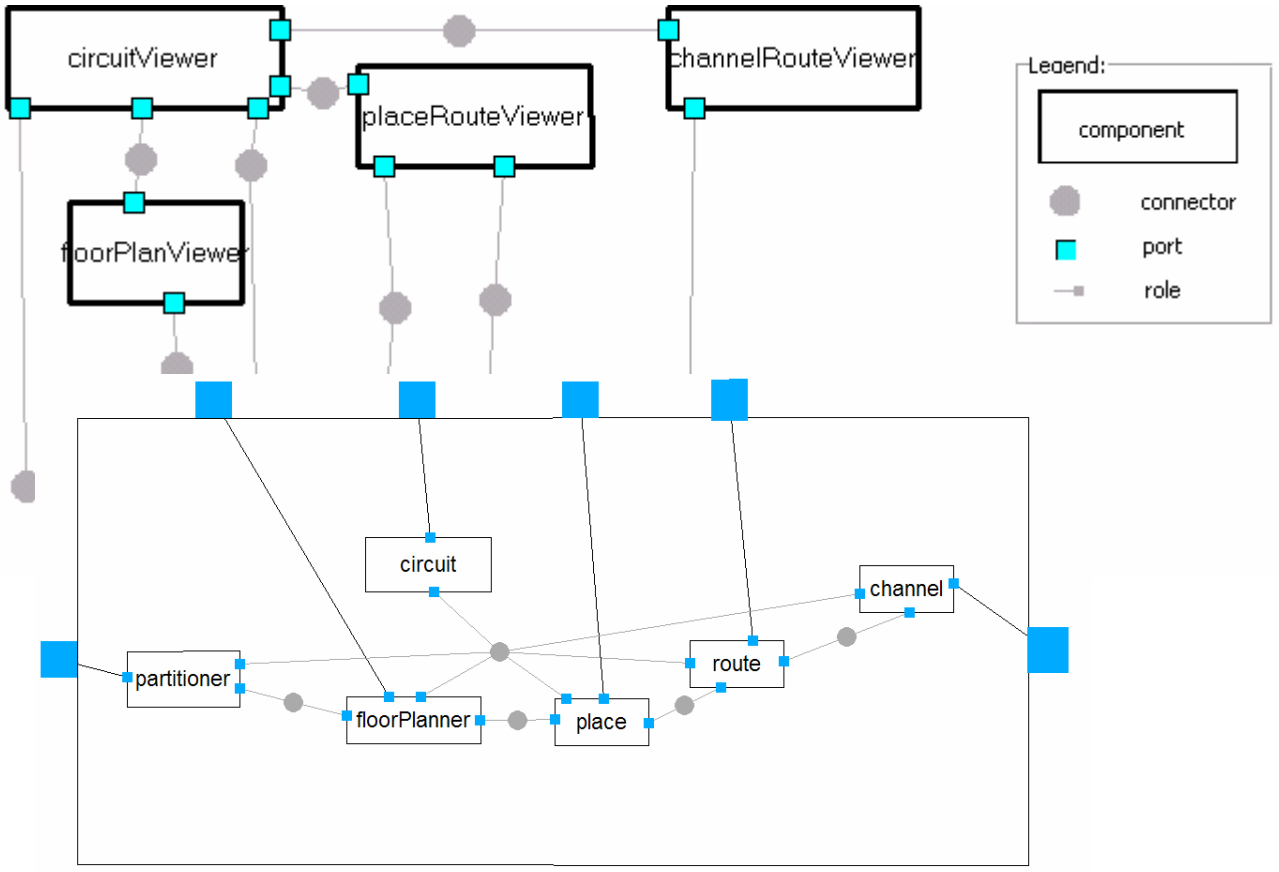
- Step 1: Setup
- Step 2: Match types
  - Optional (e.g., views are untyped)
  - Prevent matching nodes of incompatible types
- Step 3: Match instances
  - Identify renames, inserts, deletes, etc.
  - Build list of edits (edit script)
- Step 4: Modify edit script
  - Merge changes from one view into the other
  - Optional if only interested in seeing differences
- Step 5: Confirm edit script (optional)

Case Studies

- Aphyds (ArchJava application)
  - ArchJava: extension of Java
  - Embed C&C architecture in code
- Duke’s Bank (EJB application)
  - Enterprise Java Beans (EJB)

Case Study: Aphyds

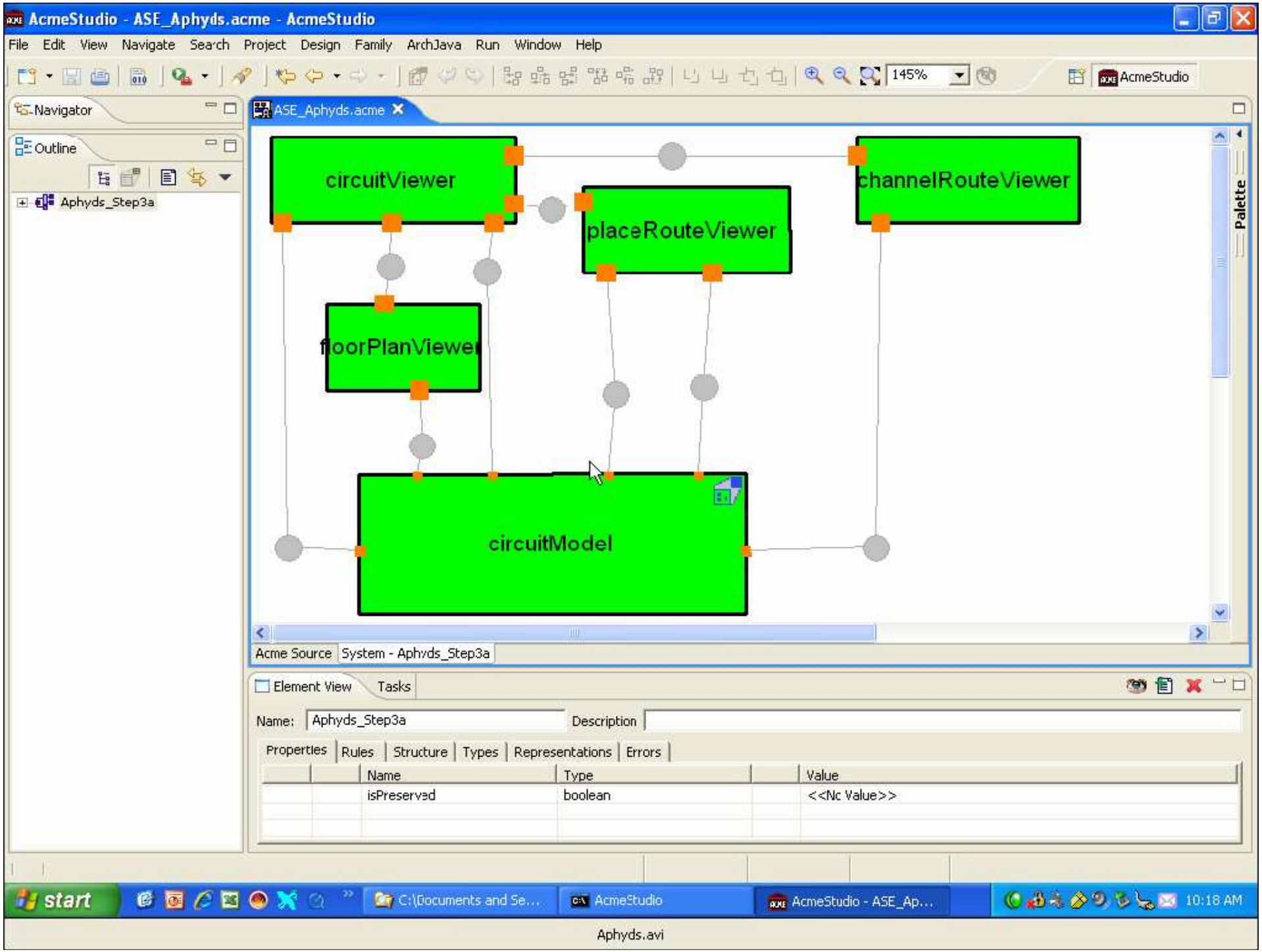




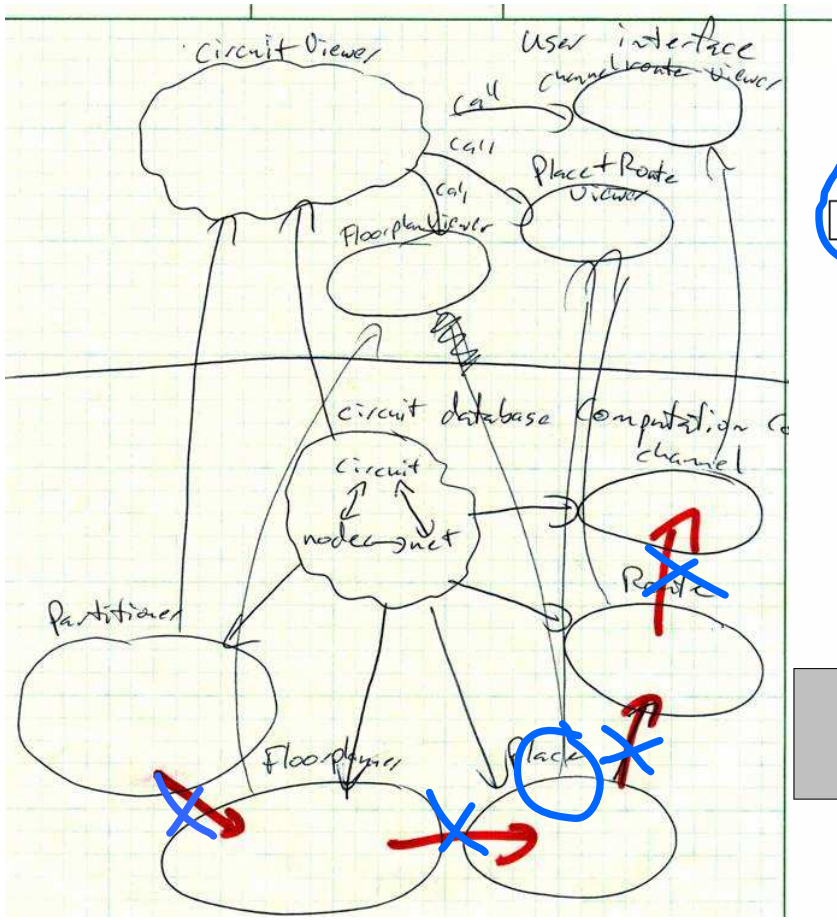
C&C View (Acme ADL) for the as-designed view

Tool Demonstration

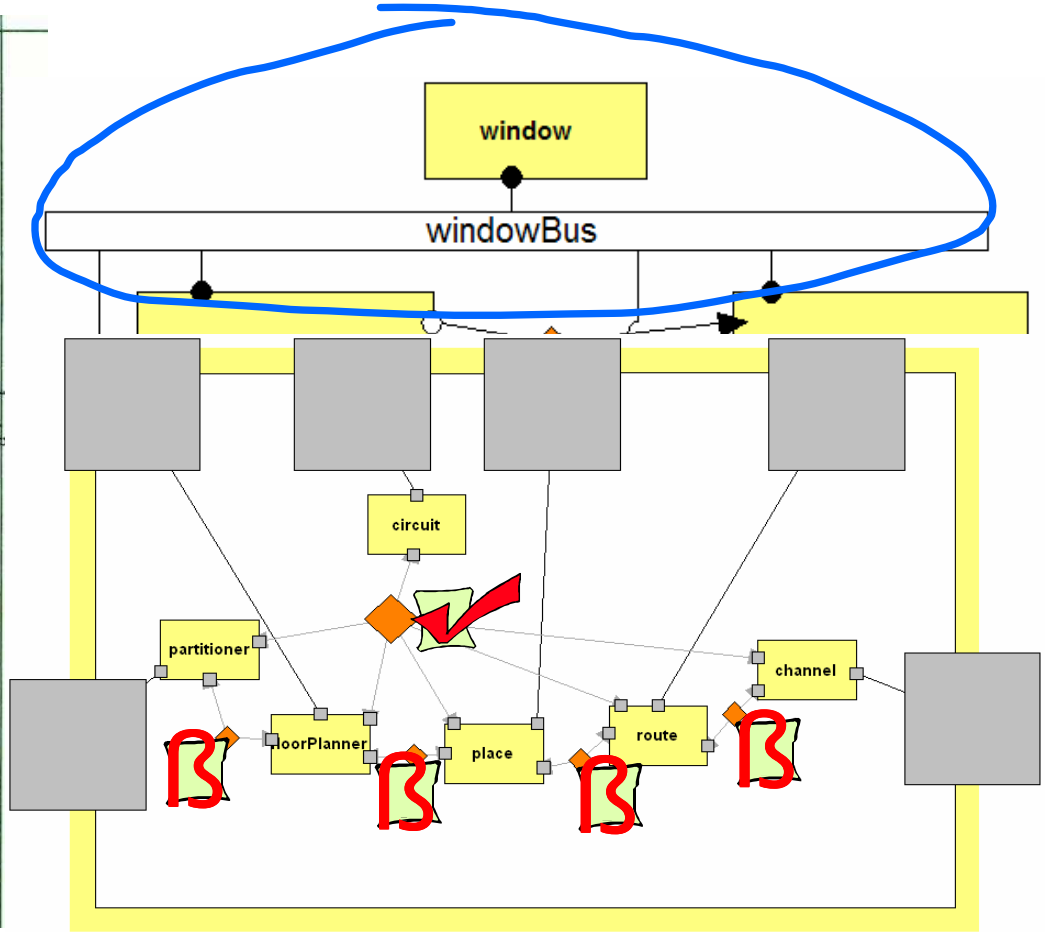
Aphyds Case Study


AcmeStudio - ASE_Aphyds.acme - AcmeStudio
File Edit View Navigate Search Project Design Family ArchJava Run Window Help
145%
AcmeStudio
Navigator
Outline
Aphyds_Step3a
ASE_Aphyds.acme
circuitViewer
placeRouteViewer
channelRouteViewer
floorPlanViewer
circuitModel
Palette
Acme Source
System - Aphyds_Step3a
Element View
Tasks
Name: Aphyds_Step3a
Description
Properties
Rules
Structure
Types
Representations
Errors
Name
Type
Value
isPreserved
boolean
<<Nc Value>>
start
C:\Documents and Se...
AcmeStudio
AcmeStudio - ASE_Ap...
10:18 AM
Aphyds.avi

Aphyds Case Study Summary



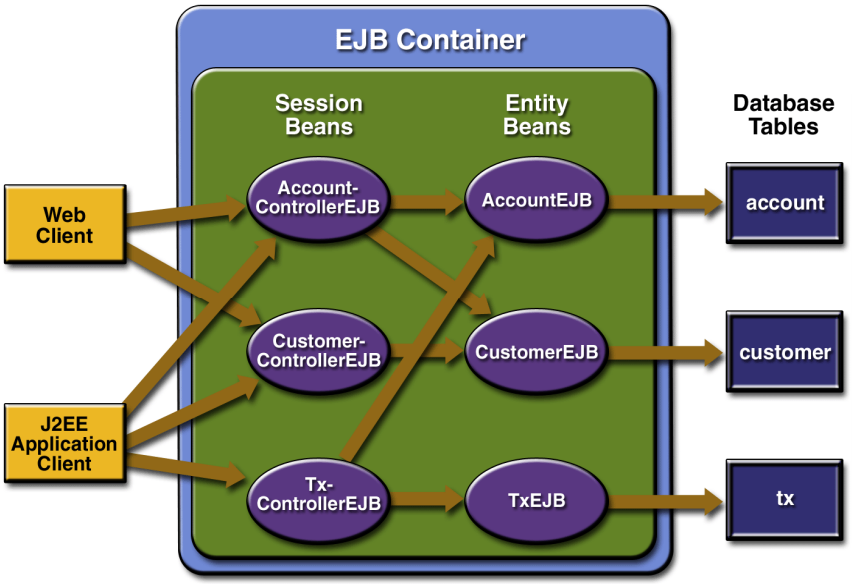
As-designed view



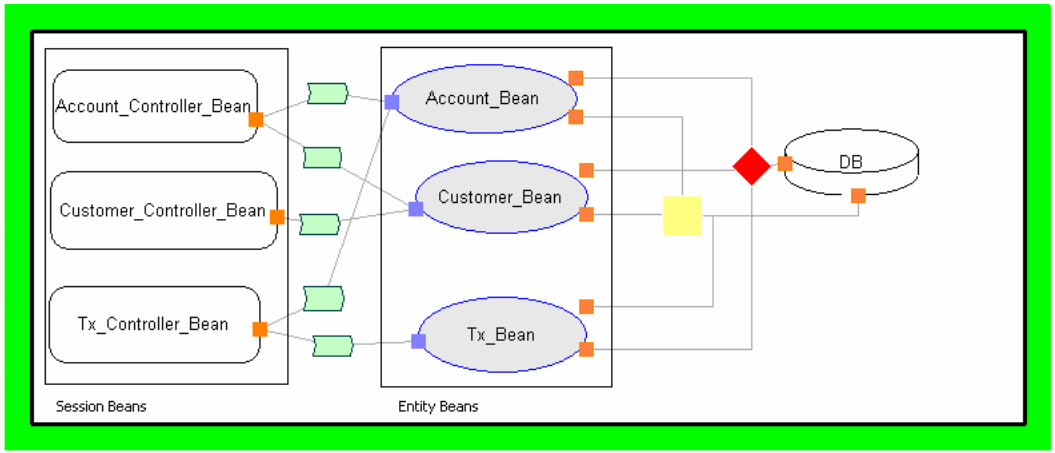
circuitModel
Representation - repmodel

Case Study: Duke's Bank

- Created model from informal diagram
  - Defined style and types based on EJB
  - Components inside an EJB container
  - Session and Entity Beans are grouped



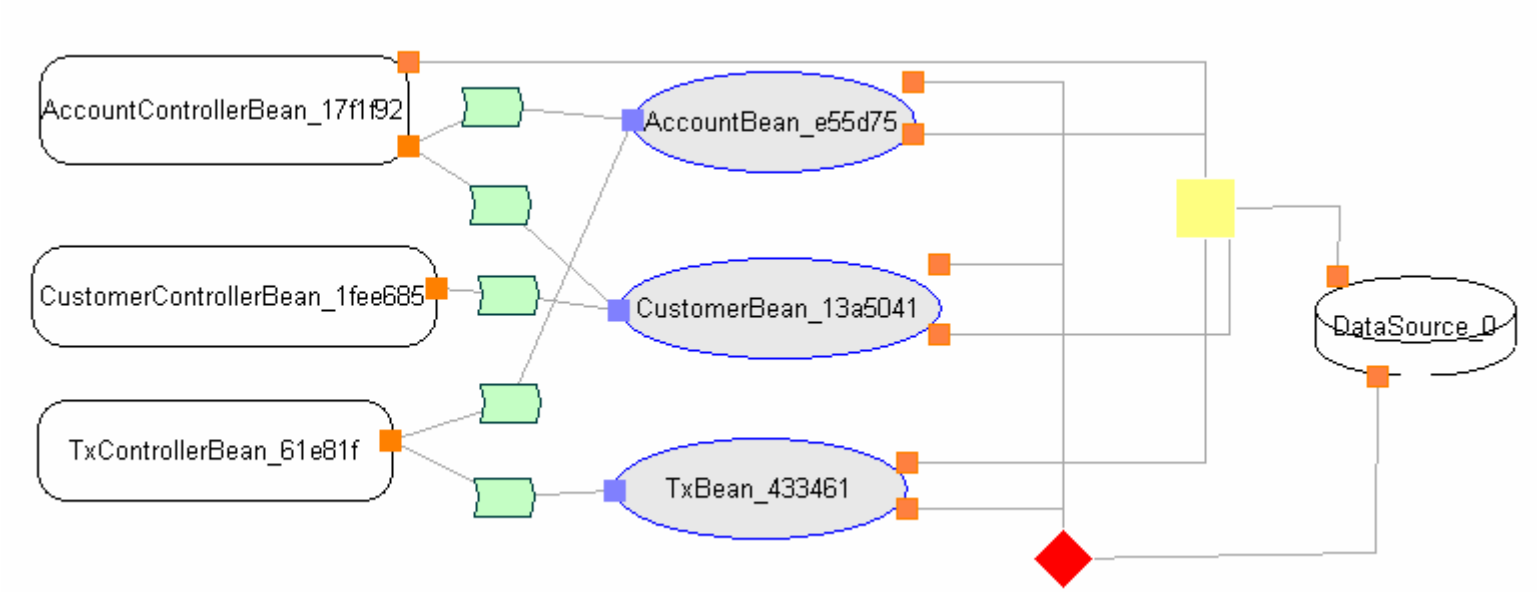
Informal Documented Diagram



C&C View (Acme ADL) for the as-designed view

Duke's Bank: As-built Architecture

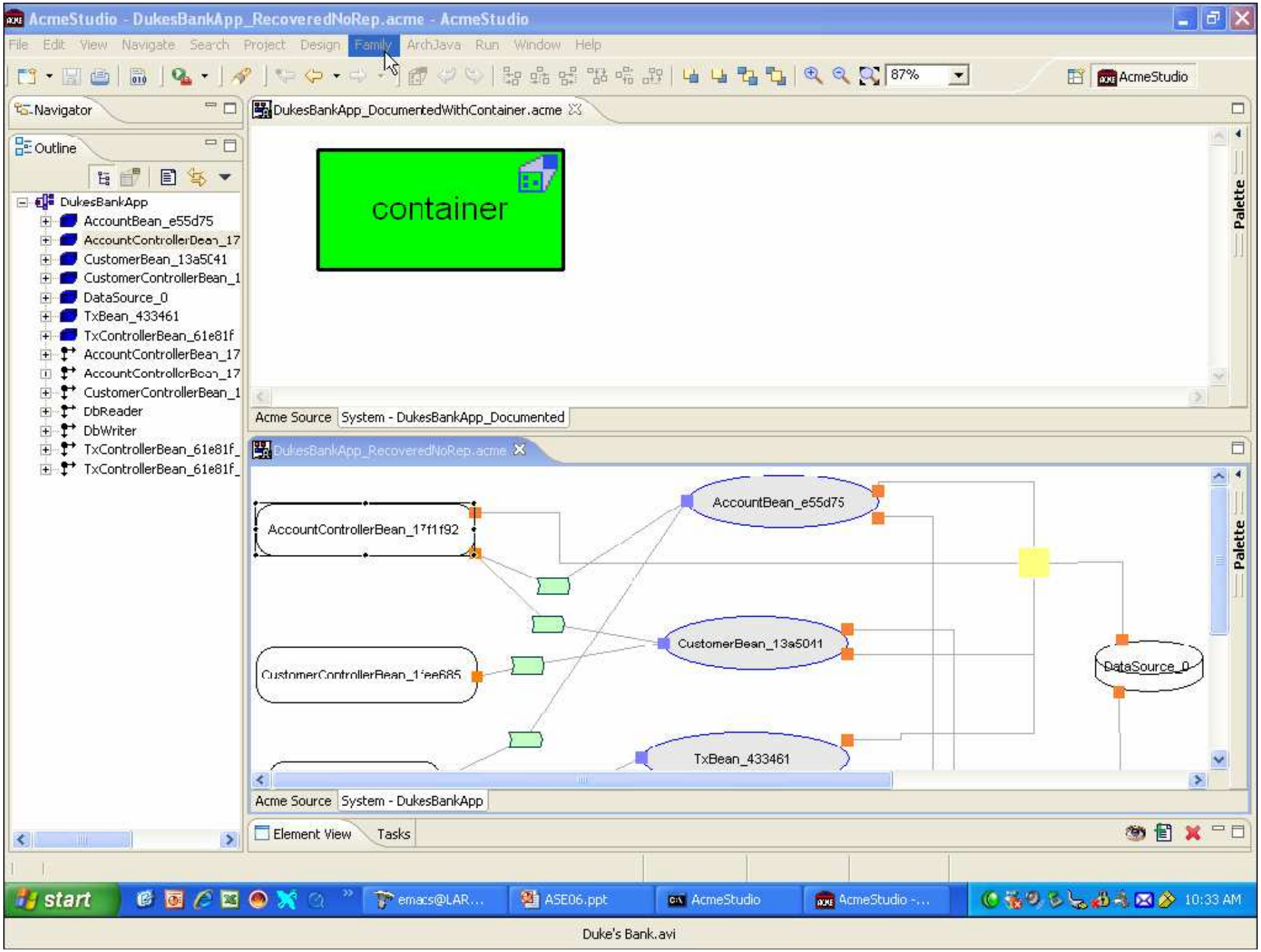
- Recovered by instrumenting running system (using DiscoTect)



As-built view

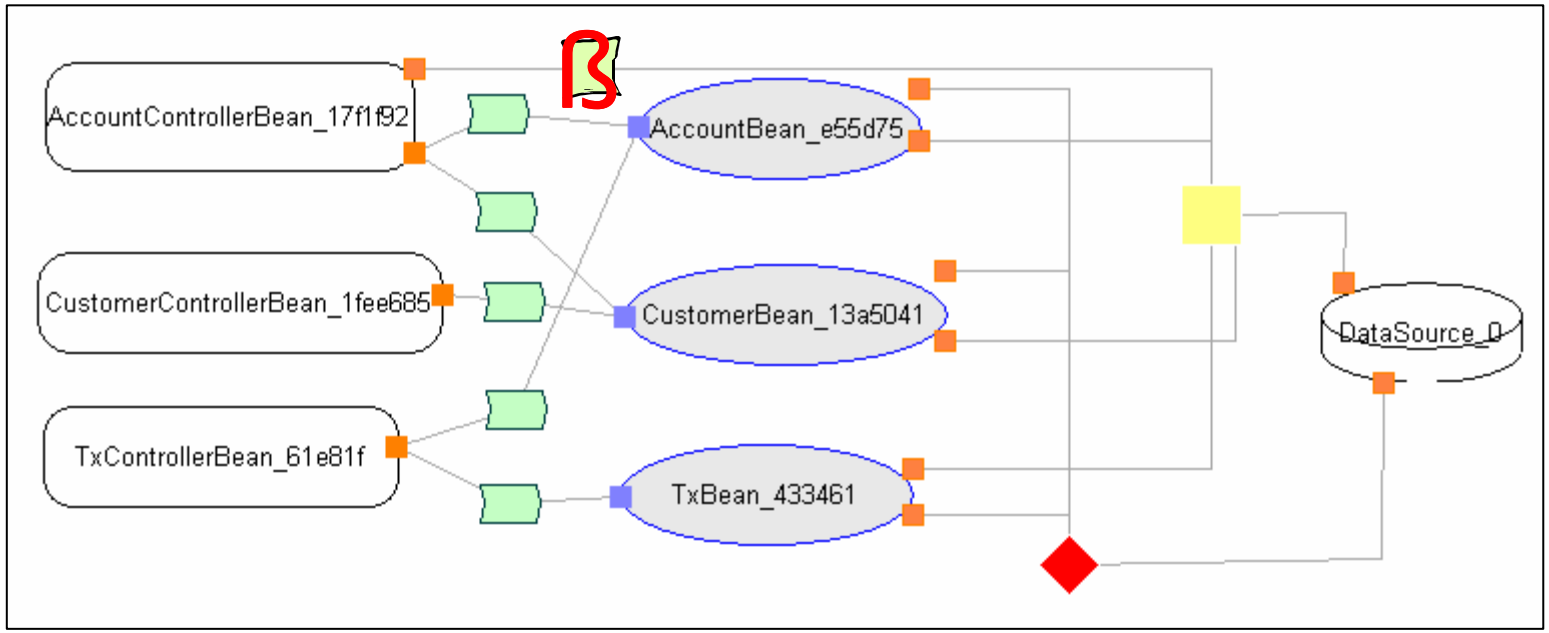
Tool Demonstration

Duke's Bank Case Study


AcmeStudio - DukesBankApp_RecoveredNoRep.acme - AcmeStudio
File Edit View Navigate Search Project Design Family ArchJava Run Window Help
87%
AcmeStudio
Navigator
Outline
DukesBankApp
AccountBean_e55d75
AccountControllerBean_17
CustomerBean_13a5C41
CustomerControllerBean_1
DataSource_0
TxBean_433461
TxControllerBean_61e81f
AccountControllerBean_17
AccountControllerBean_17
CustomerControllerBean_1
DbReader
DbWriter
TxControllerBean_61e81f_
TxControllerBean_61e81f_
DukesBankApp_DocumentedWithContainer.acme
container
Palette
Acme Source
System - DukesBankApp_Documented
DukesBankApp_RecoveredNoRep.acme
AccountBean_e55d75
AccountControllerBean_17f1f92
CustomerBean_13a5041
DataSource_0
TxBean_433461
Palette
Acme Source
System - DukesBankApp
Element View
Tasks
start
emacs@LAR...
ASE06.ppt
AcmeStudio
AcmeStudio -...
10:33 AM
Duke's Bank.avi

Duke's Bank: Case Study Summary

- Found inconsistency with specification
  - Undocumented port on AccountControllerBean communicating with DB through DbWriter connector
  - All database access must be through entity beans



Summary

- Novel algorithm for differencing and merging tree-structured data
  - Detect moves
  - Manually force/prevent matches
  - Empirical data in paper and tech report
  - Compares favorably to existing algorithms
- Tools that incorporate the algorithm
  - Case studies have shown tools to be useful
  - Found interesting anomalies