Static Extraction and Conformance Analysis of Hierarchical Runtime Architectural Structure

Marwan Abi-Antoun
Assistant Professor*
Dept. of Computer Science
Wayne State University
Detroit, Michigan

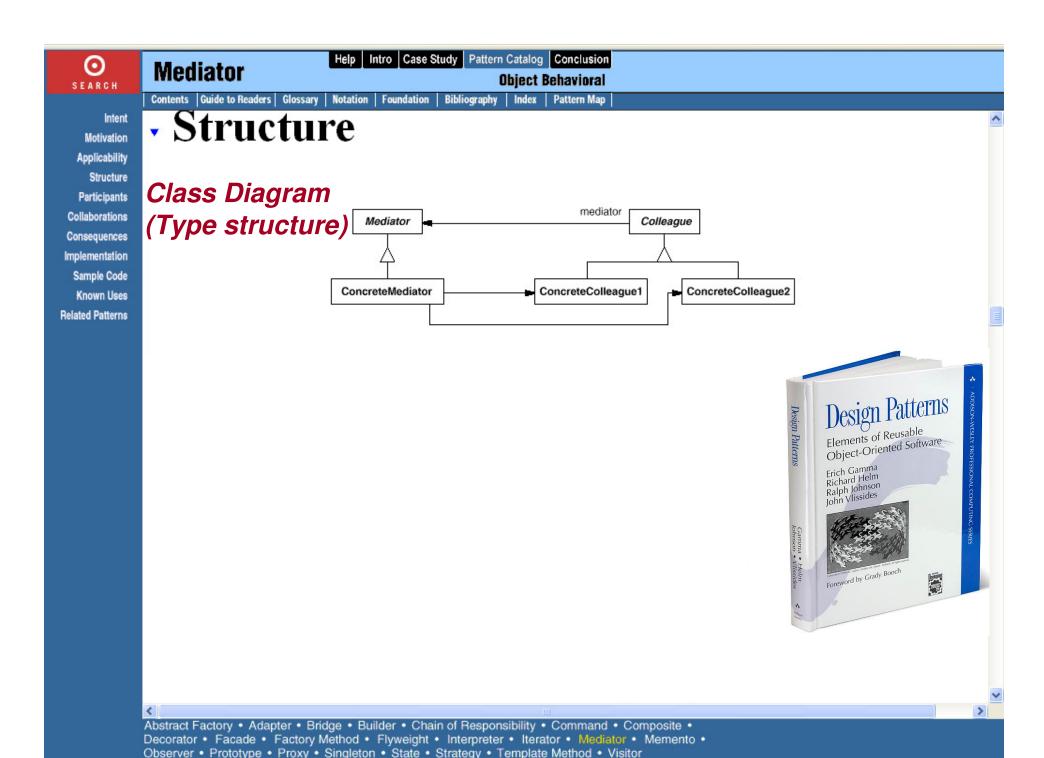
Jonathan Aldrich
Associate Professor
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

* This work was conducted while a Ph.D. student at Carnegie Mellon University

Software architecture: high-level description of a system's organization

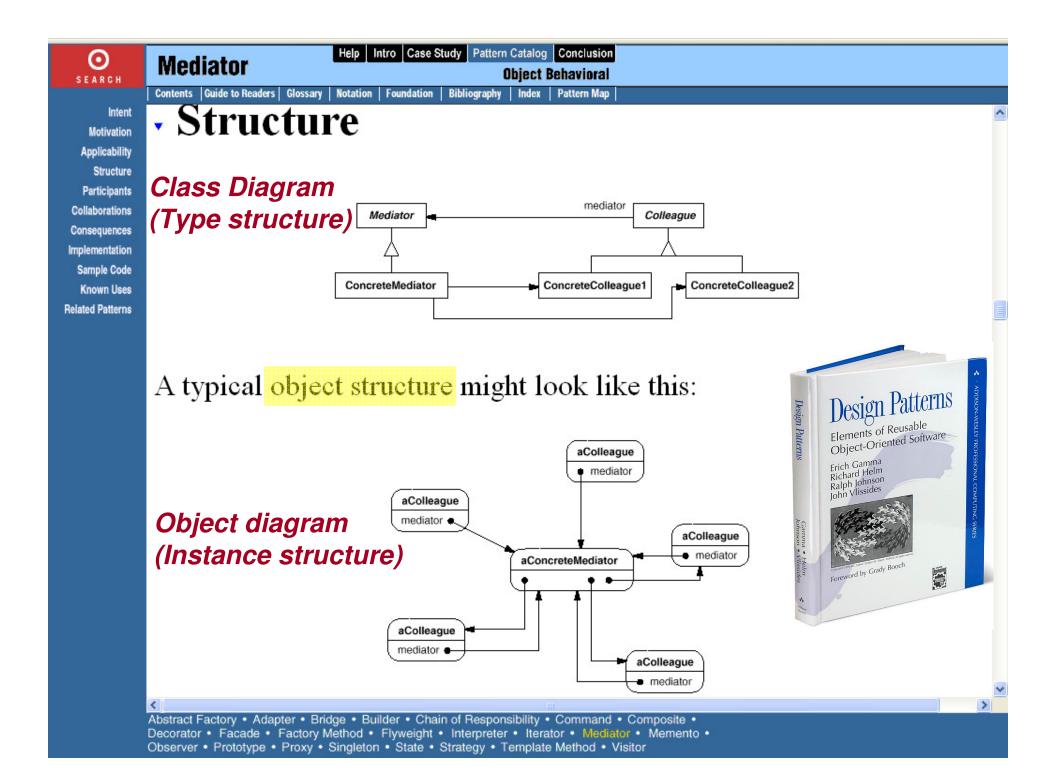
[Perry and Wolf, 1992] [Garlan and Shaw, 1993—] [Medvidovic et al., 1995—]

- Communication between stakeholders
- Qualitative architectural evaluation
- Quantitative architectural analyses
- Different perspectives or views:
 - Distinct but complementary
 - Here, we focus on structure not behavior



Code architecture shows code structure (e.g., UML class diagram)

- Static code structure of system:
 - Classes, packages, modules, layers, ...
 - Inherits from class, implements interface
 - Dependencies: imports, call graphs, etc.
- Impacts qualities like maintainability
- Mature tool support



Runtime architecture shows objects (e.g., object diagram) and their relations

- Runtime architecture of system:
 - Runtime component = sets of objects
 - Runtime interaction = e.g., points-to relation
- Impacts qualities such as security, performance, reliability, etc.
- Immature tool support

Architectural extraction: state-of-the-art

- Using dynamic analysis
 - Analyze one or more program runs
 - May omit important objects or relations that arise only in other program runs
- Using static analysis still open problem
 - Can capture all possible program runs
 - Extract low-level non-architectural views
 - Precise analyses often do not scale

Flat object graphs do not provide architectural abstraction

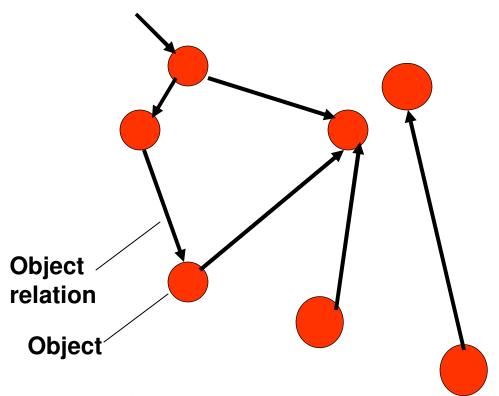
 Low-level objects mixed with architecturally significant objects

No scale-up to large programs

Output of WOMBLE (MIT) [Jackson and Waingold, TSE'01] on 8,000-line system.

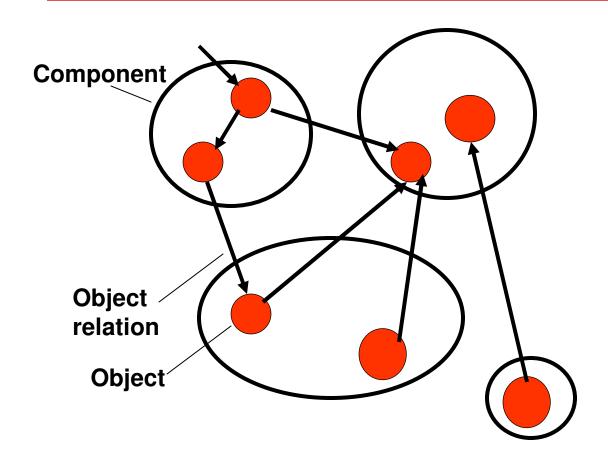
Architectural abstraction

At runtime, an object-oriented system appears as a Runtime Object Graph (ROG)

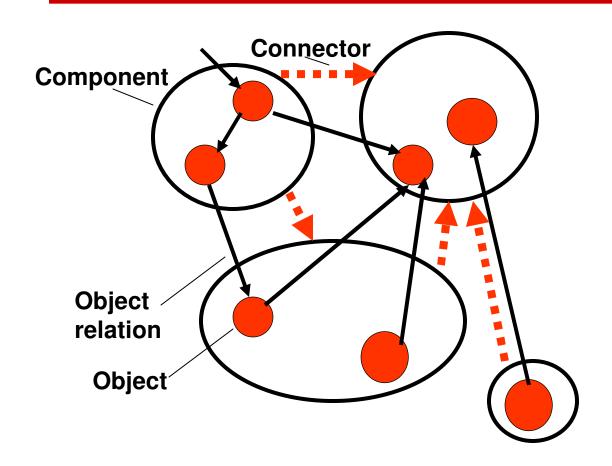


- A node represents a runtime object,
- An edge represents a points-to relation

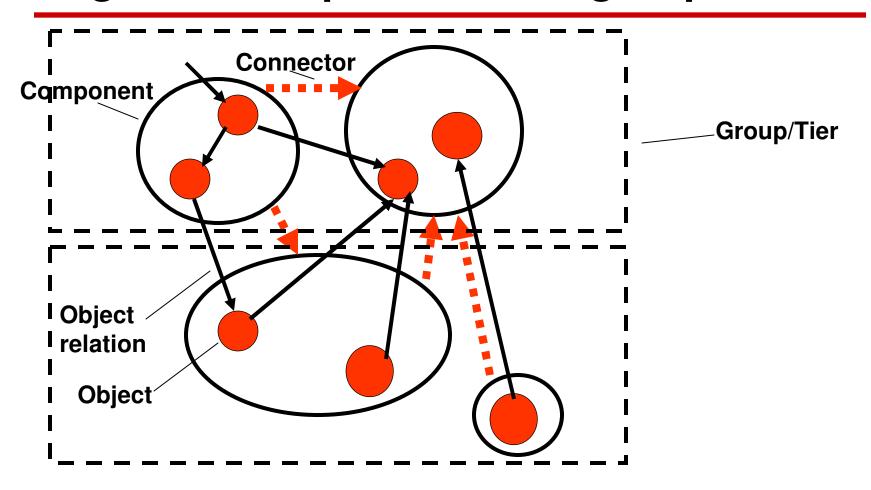
Abstract objects into "components"

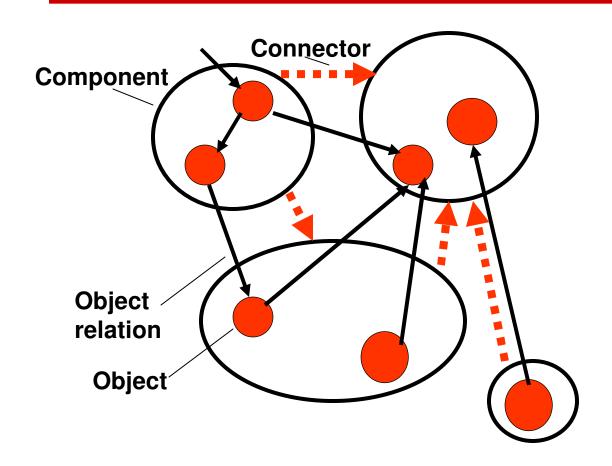


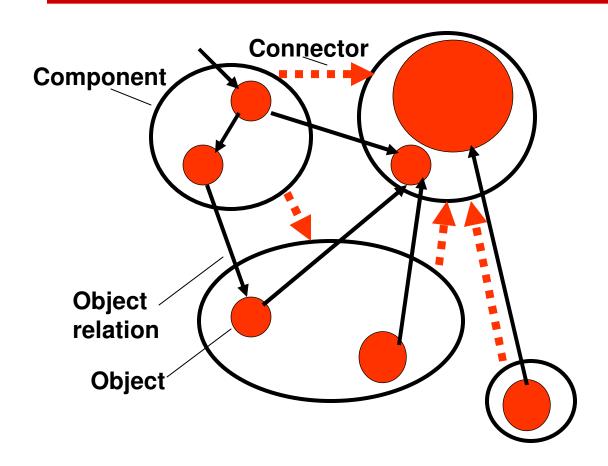
Abstract relations between components

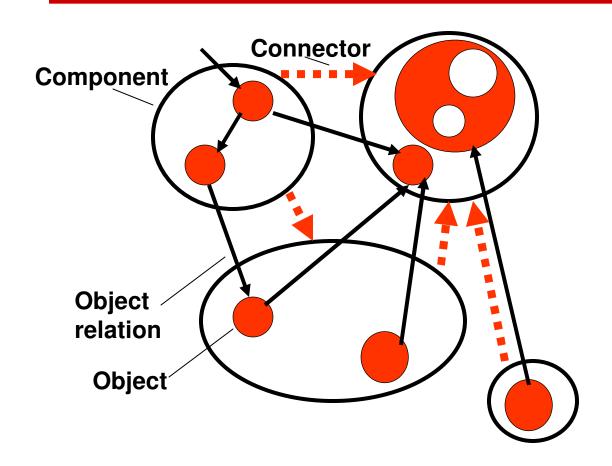


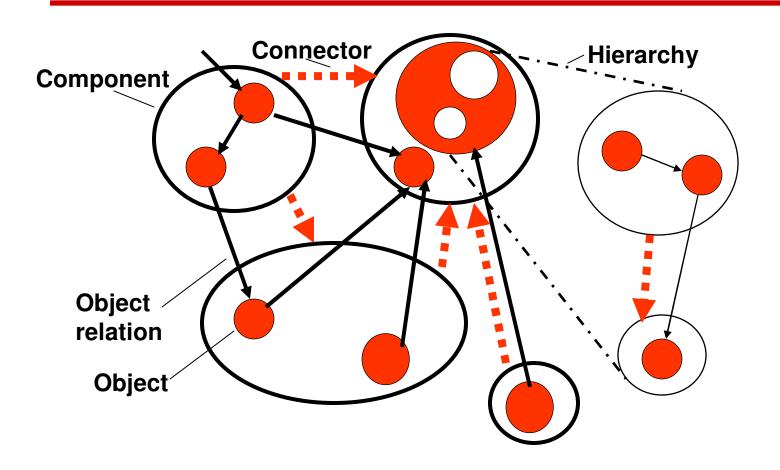
Organize components into groups/tiers











Central difficulty

Architectural **hierarchy** not readily observable in program written in general purpose programming language

Key idea: use hierarchy to convey architectural abstraction

- Pick top-level entry point
- Use ownership to impose conceptual hierarchy
- Convey abstraction by ownership hierarchy:
 - Architecturally significant objects near top
 - Low-level objects further down

```
🖃 system : Main
   ■ MODEL
       ⊞⊸c : Channel.

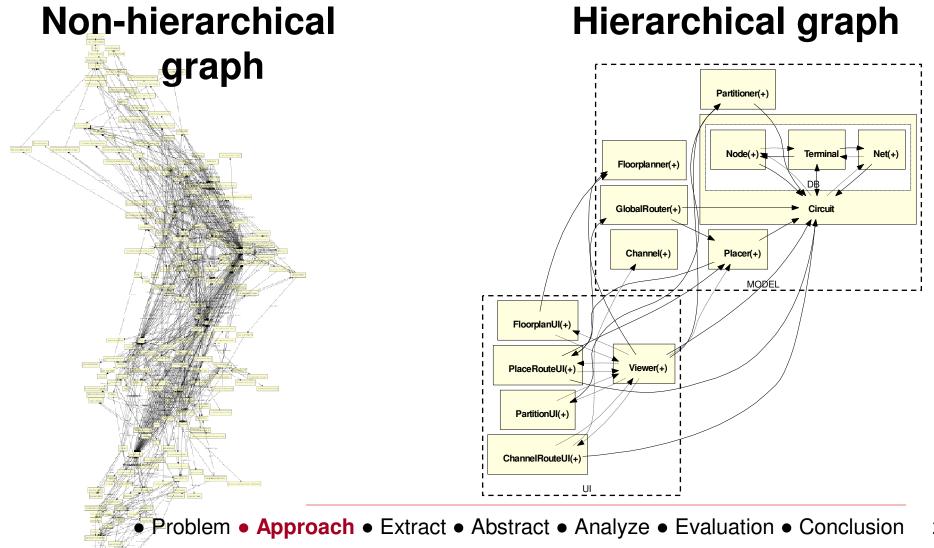
    CircuitGlobalRouting : GlobalRouter

         fp : Floorplanner
         newCircuit : Circuit
          □ DB
             cell : Node
                 owned
                        Outputs: Vector<Terminal>
              □ nt : Net
                 owned
                        Destinations: Vector < Terminal >
                t : Terminal
                 fanIter: EnumerateFanout
                 Nets: Hashtable < String, Net >
                 Nodes: Hashtable < String, Node >

    part : Partitioner

         placement : Placer
   i∃. UI
       ii circuitViewer : Viewer
       🖮 floordialog1 : FloorplanUI
         partdialog1 : PartitionUI
         placeroutedialog1 : PlaceRouteUI
```

Collapse objects based on ownership (and types) to achieve abstraction



Research contribution: SCHOLIA

SCHOLIA: static conformance checking of object-based structural views of architecture.

Scholia are annotations inserted on the margin of an ancient manuscript. The approach supports existing, i.e., legacy systems, and uses annotations.

Key idea: hierarchical object graph extracted statically

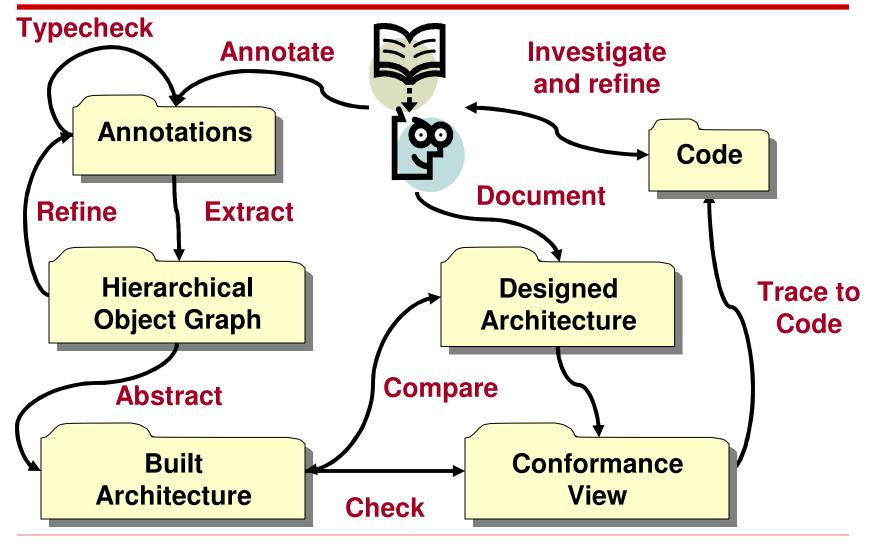
- Extract global object graph
 - Convey architectural abstraction
 - by ownership hierarchy; and
 - (optionally) by types
- Use static analysis
- Achieve soundness

Key idea: rely on ownership annotations

- Rely on local, modular, statically typecheckable ownership annotations
 - Use language support for annotations
 - Minimally invasive hints about architecture
 - Do not require new language or library
- Follow extract-abstract-analyze model

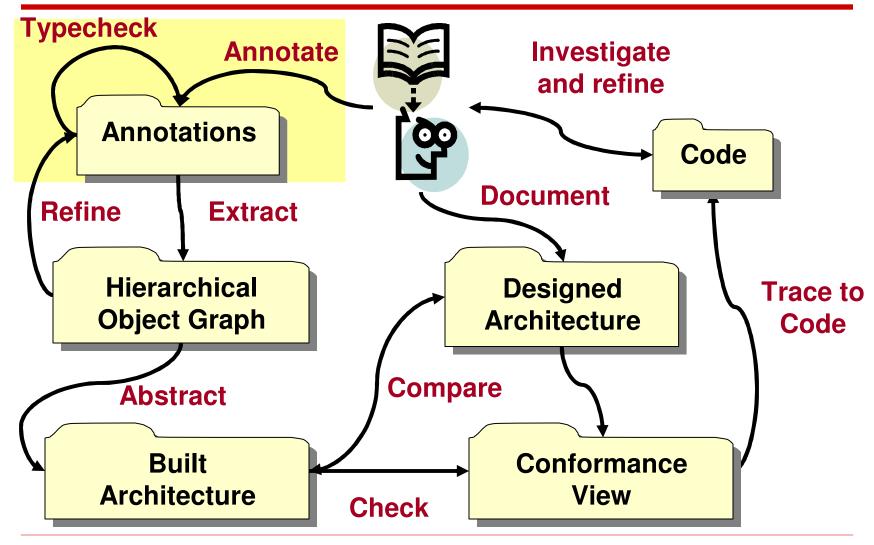
Scholia's extract-abstract-check strategy

modeled closely after Reflexion Models [Murphy et al., TSE'01]



Problem • Approach • Extract • Abstract • Analyze • Evaluation • Conclusion

Scholia: annotate + typecheck



Ownership domains are groups of objects

[Aldrich and Chambers, ECOOP'04] [Krishnaswami and Aldrich, PLDI'05]

```
· Circuit Viewer
                     Main
                                MODEL
         UI
                                                                  (911
        viewerUI
                                 circuit
                                                             circuit databa
class Main {
  domain UI, MODEL;
                                      object:
                                             Object
                                      Type
                                             Type
                                      Type
  UI Viewer viewerUI;
                                                              Declarations
  MODEL Circuit circuit;
                                                              are simplified
```

- Ownership domain = conceptual group of objects
- Each object in exactly one domain

Each class can declare domains

[Aldrich and Chambers, ECOOP'04] [Krishnaswami and Aldrich, PLDI'05]

```
· Circuit Viewer
                     Circuit
                      DB
                                                                      (911
                                 net
         node
                                                                circuit databa
class Circuit {
                                                                Circuit
   public domain DB;
                                        object:
                                               Object
                                        Type
                                               Type
                                        Type
   DB Node node;
                                                                 Declarations
   DB Net net;
                                                                 are simplified
}
```

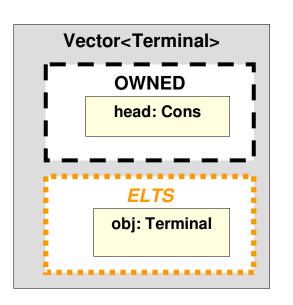
Domain parameters allow state sharing

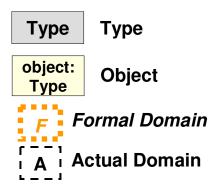
[Aldrich and Chambers, ECOOP'04] [Krishnaswami and Aldrich, PLDI'05]

- Reusable or library code often parametric with respect to ownership
 - Typically, Vector does not "own" its elements
 - Takes domain parameter ELTS for elements

```
class Vector<ELTS> {
   domain OWNED;

   ELTS Terminal obj;
   OWNED Cons head;
   ...
}
```

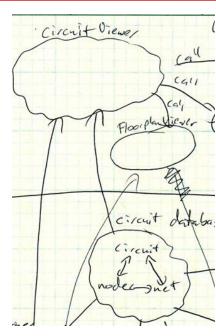




Scholia's tools use Java 1.5 annotations

[Abi-Antoun and Aldrich, IWACO'07]

```
@Domains({"UI", "MODEL"})
class Main {
    @Domain("UI") Viewer viewerUI;
    @Domain("MODEL") Circuit circuit;
    ...
}
```



- Tools use existing language support for annotations (available in Java 1.5, C#, ...)
- Annotations do not change runtime semantics

ArchCheckJ: check annotations modularly; address warnings

```
틎 Java - Aphyds PublicDomains/src/Main.iava - Eclipse SDK
Elle Edit View Source Refactor Navigate Search Project Family Design JavaArchSuite JavaDomains Run Window Help
 🎮 🖫 i ArchCog ArchConf CodeTraceJ ArchMod ArchSynchro i ArchCheckJ ArchRecJ i 🏂 🕡 🗘 🖳 😢 👑 👺 🤣 🕶
 學 / 1 9 - 2 - 4 4 - -
                          □ □ Main.java 🛭
Package Explorer
       🖨 👪 (default package)
                                    ⊕import edu.cmu.cs.aliasjava.annotations.Domain; [
         Bucket.java

    ⊕ Channel.java

                                     @Domains( { "MODEL", "UI" })
         public class Main (
         @Domain("UI<UI, MODEL>")Viewer circuitViewer = new Viewer();
         ★ CircuitDisplayer.iav
         public Main() (

■ InumerateNets.iav

         ■ InumerateNGR.jav.

■ □ EnumerateNodes.ia

                                           public void run() {
         // Add the following code if you want the Look and Feel
         FloorplanDisplayer.
                                                    // to be set to the Look and Feel of the native system.
         * try ( UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName()); ) catch (Exception e) ( )
         ■ □ GlobalRouter.iava

■ ImageGenerator.ja

■ JAboutDialog.java

                                                    // Create a new instance of our application's frame, and make it visible.

■ JChartSized.java

         🗏 🛂 Main.java
                                                    circuitViewer.setVisible(true);
            catch (@Domain("unique") Throwable t) {

■ NetDialog.iava

                                                    t.printStackTrace();

■ NetGlobalRouting.ia

                                                    // Ensure the application exits with an error condition.
         H- D Node iava
                                                    System.exit(1);
         NodeDialog.java

■ Partitioner.java

■ PartitionTranscript.;

■ PartTransDisplayer.

                                           public static void main(@Domain("lent[shared]")String args[]) {

■ PartTransViewerUI.

                                               @Domain("lent") Main system = new Main();
         ∄ PartUI.java
                                               system.run();

■ PlacementGraphUI.

    Placer, java

                                 Problems X

    PlaceRouteDisplaye

■ PlaceRouteUI.iava

■ Queue.java

                                  Description 4
                                                                                                                                           Resource Path
                                                                                                                                                                      Locat... Type

    Randomizer.iava

                                     i Alias annotation does not match expected annotation lent at assignment return this; 🗆
                                                                                                                                          FloorplanUI.... Aphyds_PublicDo... line 82 Problem

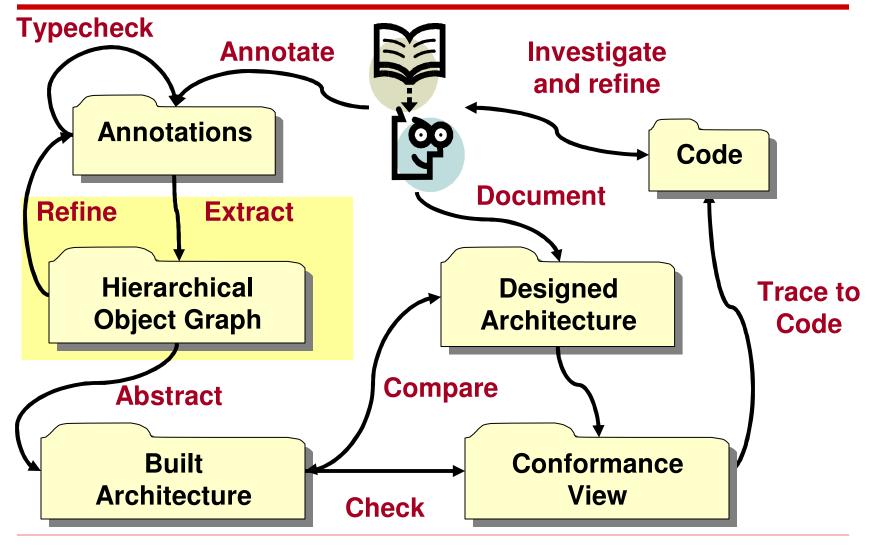
■ SlicingTree.java

                                       Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getNet(netname)
                                                                                                                                          CircuitDispla... Aphyds_PublicDo... line 205
         🗓 🛂 SwingWorker.java
                                       Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getNode(e.getActionCommand()) CircuitDispla... Aphyds_PublicDo... line 146 Problem

■ In Terminal.java

                                       Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getNode(nodename)
                                                                                                                                          CircuitDispla... Aphyds_PublicDo... line 177
         Alias annotation fpdb does not match expected annotation fp.DB at assignment fp.getBestFloorplan()
                                                                                                                                          FloorplanUI.... Aphyds_PublicDo... line 108 Problem
         GlobalRouter.junk.txt
                                                                                                                                          FloorplanUI.... Aphyds PublicDo... line 100 Problem
                                       Alias annotation fpdb does not match expected annotation fp.DB at assignment fp.getTreeRoot()
         Oueue.txt
                                     i Alias annotation fpdb does not match expected annotation null at assignment (SlicingTree)node.getUserObject()
                                                                                                                                          FloorplanUI.... Aphyds PublicDo... line 197 Problem
         src-ps.bat
src.acme
                                                                                                                                          NetGlobalRo... Aphyds PublicDo... line 492 Problem
                                     Alias annotation owner does not match expected annotation lent at assignment newDest. Source withis
                                       Alias annotation ptdb does not match expected annotation partitioner.DB at assignment pt=partitioner.partitionCircuit(pdiag)
                                                                                                                                          PartUI.iava Aphyds PublicDo... line 417 Problem
         src hat
                                       Alias annotation shared does not match expected annotation null at assignment e.getActionCommand()
                                                                                                                                          CircuitDispla... Aphyds_PublicDo... line 146 Problem
```

Scholia: extract object graph



Generate ObjectGraph by abstract interpretation of program

- ObjectGraph: graph of objects and domains (no types/classes)
 - Analyze local, modular annotations
 - Generate global object graph
 - Start from a root class
- Abstractly interpret/execute program:
 - New expression → Object
 - Domain declaration → Domain
 - Field declaration → Edge
- A kind of a points-to analysis

Challenge: ObjectGraph must show all objects in each domain

- At runtime, each domain parameter bound to some actual domain
- Track bindings of formal domain → actual domain

```
[this → c]
Bindings := [ ]
class Circuit { ...
  public domain DB;
  DB Node nd = new Node<DB>();
  analyze(Node, nd, [Node::OWNER → c.DB] )
  ...
}
[this → c.DB.nd]
Bindings := [Node::OWNER → c.DB ]
class Node<OWNER> {
}
```

Challenge: must handle possible aliasing

- We do not use an alias analysis
 - Rely on precision about aliasing from ownership domain annotations
 - Objects in different domains cannot alias
 - Objects in same domain may alias

ObjectGraph: data type declarations

OGraph

- D ::= ODomain(Id = D_{id}, Domain = C::d)
- O ::= $OObject(Id = O_{id}, Owner = D, Type = C)$
- E ::= $OEdge(From = O_{src}, To = O_{dst})$
- Here, declarations are simplified
 - OObject also has domain parameters D_i
 - See dissertation for full details

ObjectGraph: abstractly interpret new expression into OObject c

```
LEGEND
Private
domain
                                                                              C:
I Public
                                                                            Circuit
I domain
                                                                             (00)
 object:
            Circuit c = new Circuit();
  Type
            Oobject(c, null, Circuit) (00)
            class Circuit {
```

ObjectGraph: analyze class Circuit in the context of OObject c

```
LEGEND
Private
domain
                                                                                  C:
I Public
                                                                                Circuit
<sup>I</sup> domain
                                                                                  (O0)
 object:
             Circuit c = new Circuit();
  Type
             oobject(c, null, Circuit) (00)
             analyze(Circuit, c, [ ])
             class Circuit {
```

ObjectGraph: abstractly interpret domain declaration into ODomain c.DB

```
LEGEND
Private
domain I
I Public
                                                                                  Circuit
I domain
                                                                                    (O0)
                                          <sup>(D1)</sup> DB
 object:
  Type
             Circuit c = new Circuit();
             OObject(c, null, Circuit) (00)
             [this \rightarrow c]
             Bindings := [ ]
             class Circuit {
               public domain DB;
               ODomain(c.DB, Circuit::DB) (D1)
```

ObjectGraph: abstractly interpret new expression into OObject c.DB.nd

```
LEGEND
                                                       (O1) node: Node
Private
domain
I Public
                                                                                    Circuit
I domain
                                                                                      (00)
                                           <sup>(D1)</sup> DB
 object:
  Type
              [this \rightarrow c]
             Bindings := [ ]
             class Circuit {
                DB Node nd = new Node<DB>();
                OObject(c.DB.nd, c.DB, Node) (01)
```

ObjectGraph: abstractly interpret field declaration into OEdge

```
LEGEND
                                                      (O1) node: Node
Private
domain
I Public
                                                                                   Circuit
I domain
                                                                                    (00)
                                          <sup>(D1)</sup> DB
 object:
  Type
             [this \rightarrow c]
             Bindings := [ ]
             class Circuit {
               DB Node nd = new Node<DB>();
               OObject(c.DB.nd, c.DB, Node) (01)
               OEdge(c, c.DB.nd) (E1)
```

ObjectGraph: analyze class Node in context of OObject c.DB.nd

```
LEGEND
                                                      (O1) node: Node
Private
domain
I Public
                                                                                  Circuit
I domain
                                                                                    (00)
                                          <sup>(D1)</sup> DB
 object:
  Type
             [this \rightarrow c]
             Bindings := [ ]
             class Circuit {
               DB Node nd = new Node<DB>();
               analyze(Node, c.DB.nd, [Node::OWNER→ c.DB])
             class Node<OWNER> {
```

ObjectGraph: abstractly interpret domain declaration into ODomain

```
LEGEND
                                                    (O1) node: Node
Private
domain I
I Public
                                                                                Circuit
I domain
                                                                                 (O0)
                                         <sup>(D1)</sup> DB
 object:
  Type
             [this → c.DB.nd]
             Bindings := [Node::OWNER → c.DB]
             class Node<OWNER> {
              ODomain(c.DB.nd.OWNED, Node::OWNED) (D3)
              domain OWNED;
```

ObjectGraph: abstractly new expression into OObject c.DB.nd.OWNED.trms

```
LEGEND
                                                  (O1) node: Node
Private |
                                               trms: Vector<Terminal>
                                             (04)
domain
                                                   (D3) OWNED
I Public
                                                                            Circuit
I domain
                                                                              (00)
                                       <sup>(D1)</sup> DB
 object:
  Type
            [this → c.DB.nd]
            Bindings := [Node::OWNER → c.DB]
            class Node<OWNER> {
             Oobject(c.DB.nd.OWNED.trms, c.DB.nd.OWNED, Vector<Terminal>) (04)
             OWNED Vector<OWNER Terminal> trms = new Vector<...>();
             OEdge(c.DB.nd, c.DB.nd.OWNED.trms) (E5)
```

ObjectGraph: analyze class Vector in the context of OObject c.DB.nd.OWNED.trms

```
LEGEND
                                                   (O1) node: Node
Private |
                                              (04)
domain I
                                                 trms: Vector<Terminal>
                                                     (D3) OWNED
I Public
                                                                              Circuit
I domain
                                                                                (00)
                                        <sup>(D1)</sup> DB
 object:
  Type
             [this → c.DB.nd]
            Bindings := [Node::OWNER → c.DB]
            class Node<OWNER> {
               analyze(Vector, c.DB.nd.OWNED.trms, [Vector::ELTS → c.DB])
            class Vector<ELTS T> {
```

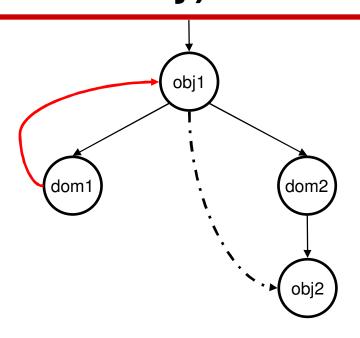
ObjectGraph: abstractly interpret field declaration into OEdge

```
LEGEND
                                                    (O1) node: Node
Private |
                                               (04)
domain
                                                  trms: Vector<Terminal>
                                                      (D3) OWNED
I Public
                                                                                Circuit
I domain
                                                                 (E6)
                                                 (O3)
                                                                                 (O0)
                                                       term: Terminal
                                         <sup>(D1)</sup> DB
 object:
  Type
             [this → c.DB.nd.OWNED.trms]
             Bindings := [Vector::ELTS → c.DB]
             [T \rightarrow Termina]
             class Vector<ELTS T> {
              OObject(c.DB.term, Terminal) in lookup(c.DB, Terminal)
              OEdge(c.DB.nd.OWNED.trms, c.DB.term) (E6)
              ELTS T obj;
```

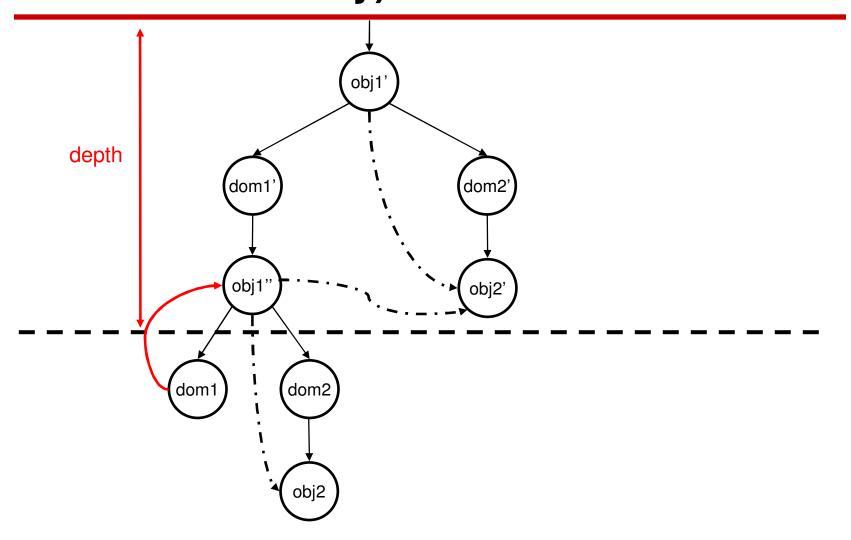
Challenge: ObjectGraph can have cycles. Unfold it for visualization (DisplayGraph)

- Recursive types create cycles in ObjectGraph
 - This avoids non-termination
 - Justifies ODomain not having a unique owning OObject
 - Details in paper/dissertation
- Visualization unfolds ObjectGraph to limited depth

Unfold ObjectGraph to limited depth (for visualization only)



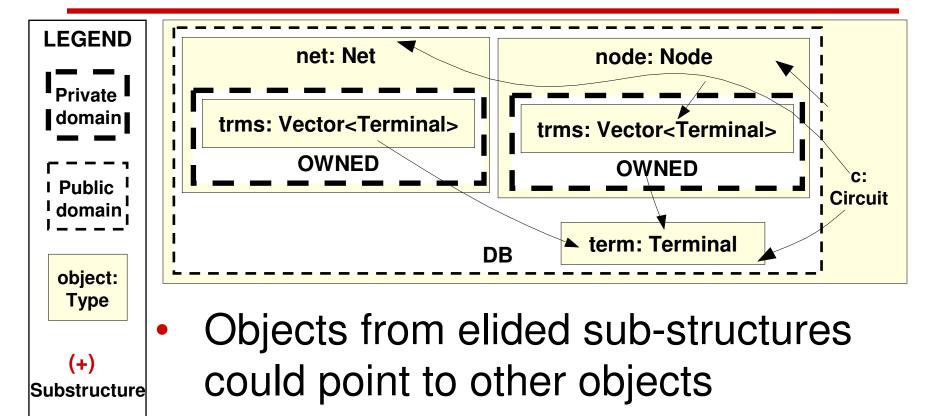
Unfold ObjectGraph to limited depth (for visualization only)



Developer interacts with DisplayGraph

- Control unfolding depth
- Collapse/expand selected elements
- Control abstraction by types

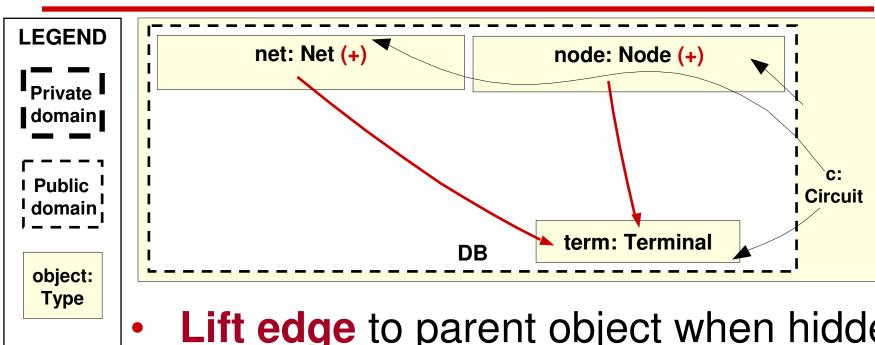
Expand/collapse objects



Expand/collapse objects

(+)

Substructure



Lift edge to parent object when hidden sub-object points to external objects

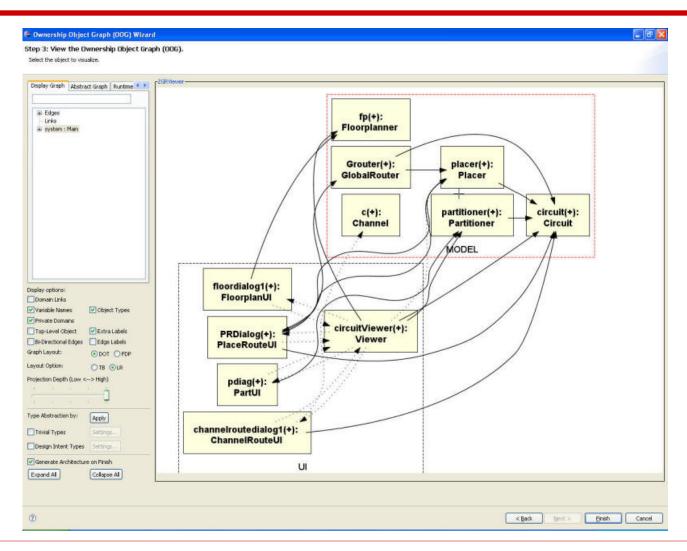
Extraction key property: soundness

- Map each object to exactly one node
- Show all edges between objects

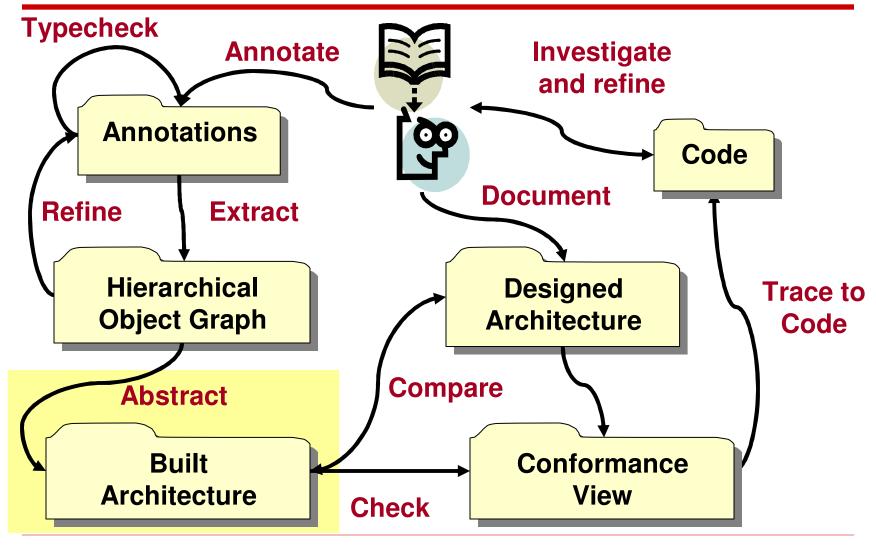
Demonstrating soundness

- Featherweight Java [Igarashi, Pierce and Wadler, TOPLAS'01]
 - + ownership domains [Aldrich and Chambers, ECOOP'04]
- Constraint-based specification
- Soundness proof
 - Instrumented runtime semantics
 - Approximation relation between runtime states and ObjectGraph
 - Standard Progress, Preservation theorems
 - Details in dissertation

ArchRecJ: extract object graph



Scholia: abstract object graph

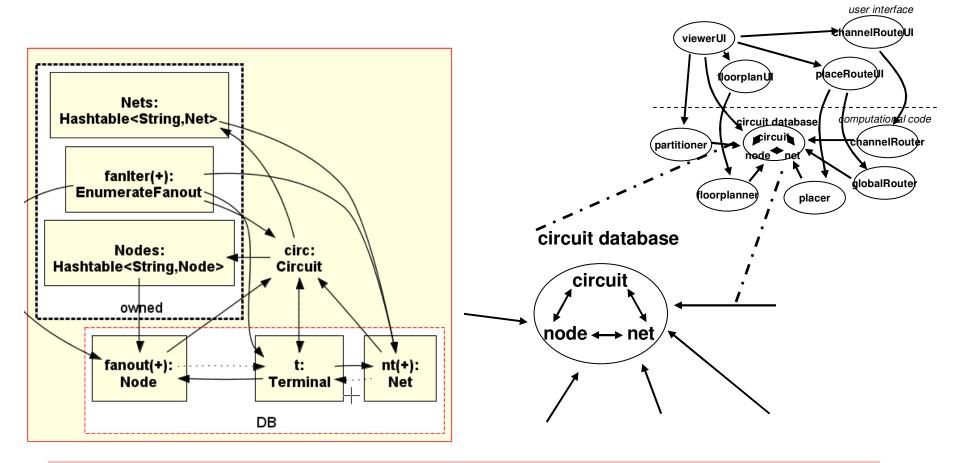


Why abstract an object graph?

- Extracted object graph provides architectural abstraction by ownership hierarchy and by types
- Often, object graph not isomorphic to architect's intended architecture

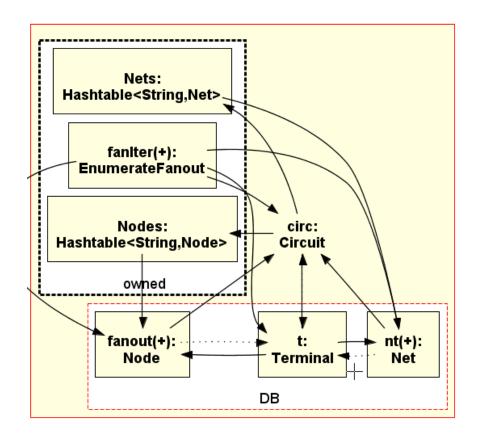
Object graph vs. target architecture

Aphyds object graph Aphyds target architecture

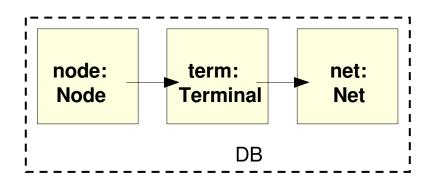


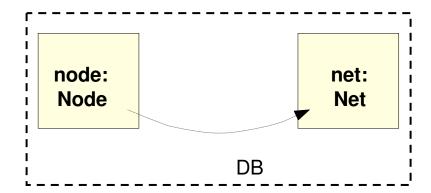
Soundly summarize private domains

- Private domains hold representation
- Public domains hold visible state
- Eliding private domains reduces clutter
- Must be done soundly



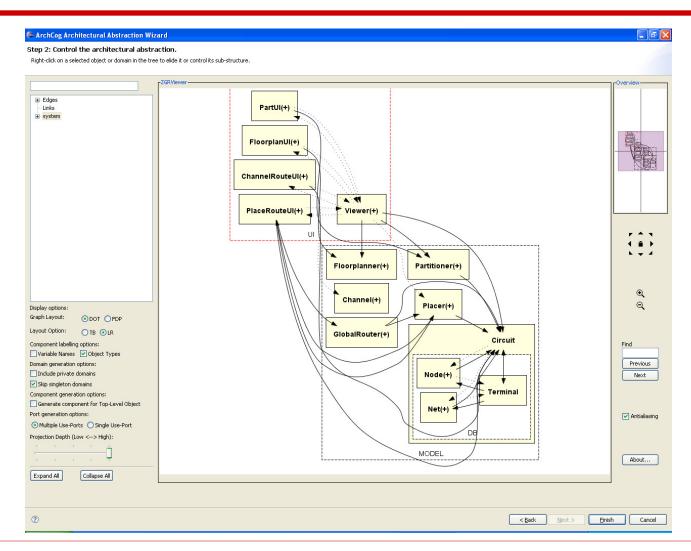
Soundly summarizing elided objects



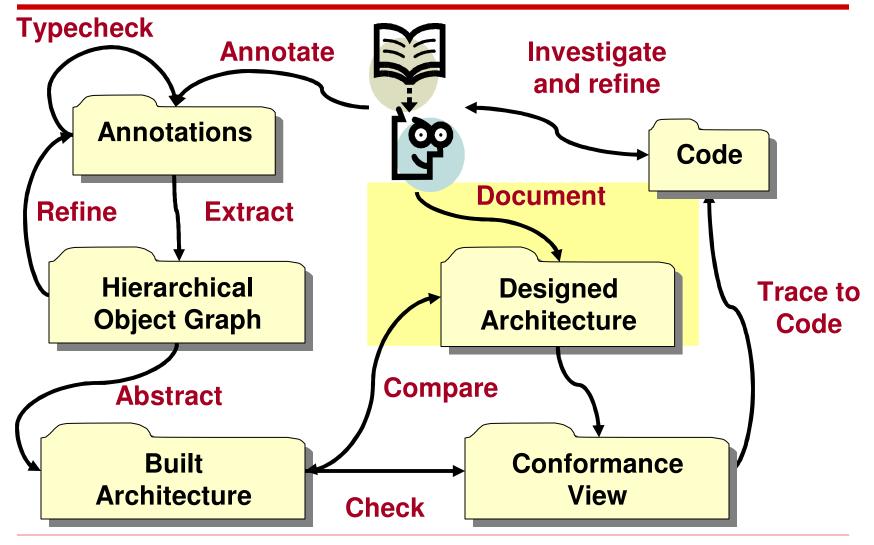


- Eliding object 'term' leads to summary edge to show transitive communication
- Effectively, abstracts object into edge
- Notion of rich connector in architecture

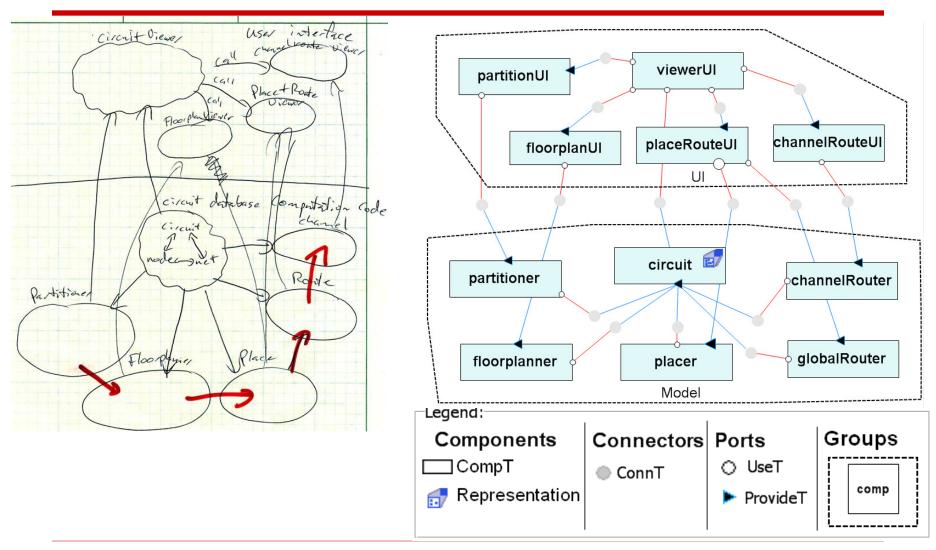
ArchCog: abstract object graph; present in architecture description language



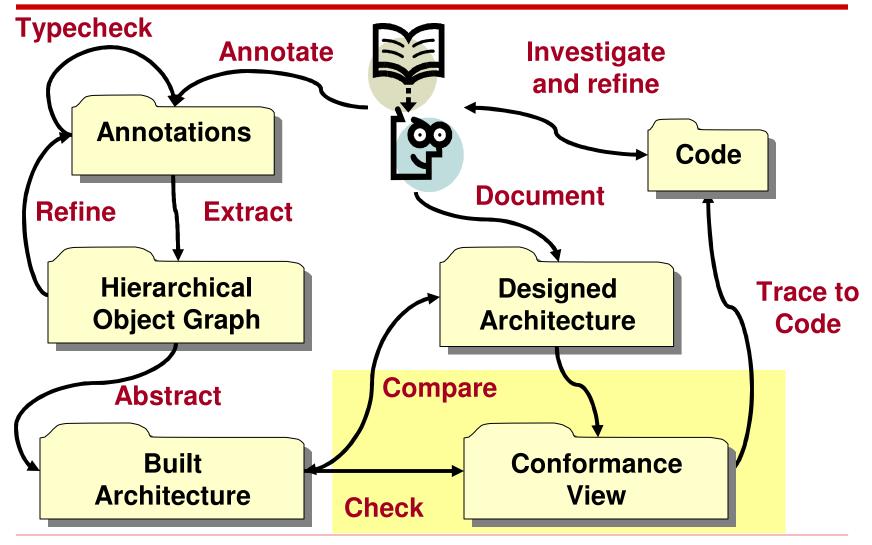
Scholia: document target architecture



Aphyds: document designed architecture in architecture description language



Scholia: analyze conformance



Analyzing conformance of system to target architecture

- Key property: communication integrity
 [Moriconi et al., TSE'95] [Luckham and Vera, TSE'95]
 - <u>Definition</u>: each component in the implementation may only communicate directly with the components to which it is connected in the architecture.
- Informal diagrams omit communication;
 confirmed by experience at Microsoft
 [Murphy et al., TSE'01] [Aldrich et al., ICSE'02]

Why different from view synchronization?

- View synchronization makes two architectural views identical
- Conformance analysis
 - Enforce communication integrity
 - Account for communication in built view that is not in designed view
 - Do not propagate all implementation objects

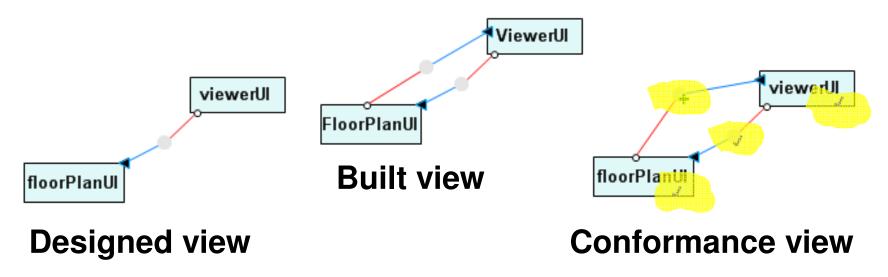
Conformance analysis identifies following key differences

- Convergence: node or edge in both built and in designed view
- Divergence: node or edge in built view, but not in designed view
- Absence: node or edge in designed view, but not in built view

Terminology adopted from Reflexion Models [Murphy et al., TSE'01]

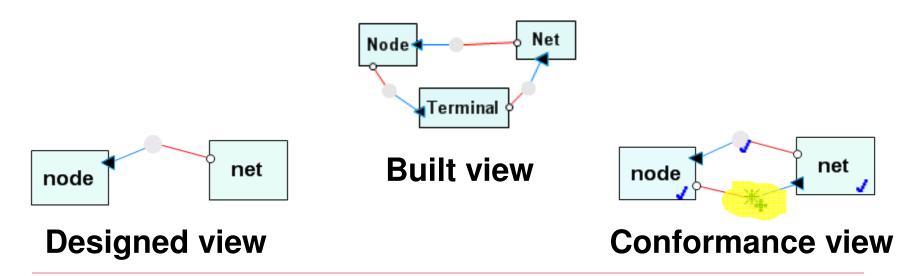
Highlight differing connections, but use the names from the built view

- Structurally match components in built view to those in designed view
- Show differing connections as divergences or absences



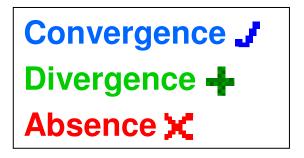
Summarize divergent components

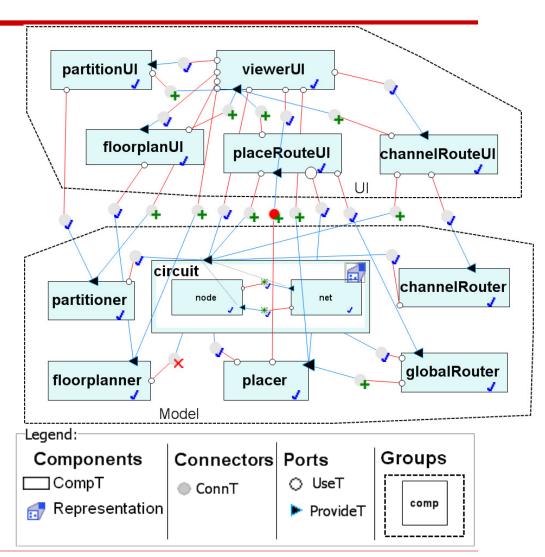
- Do not directly propagate additional components
- Summarize additional components in built architecture using summary edges **



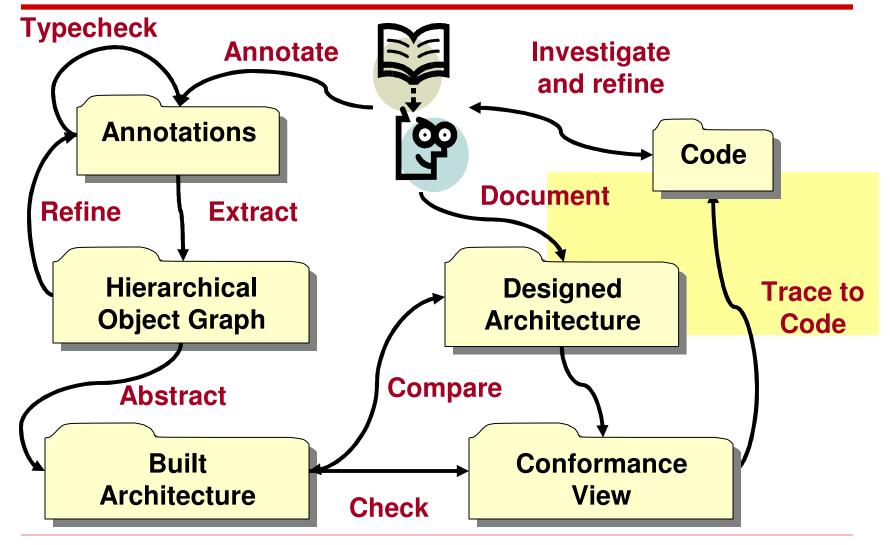
Developer investigates reported differences

- Study findings
- Trace to code



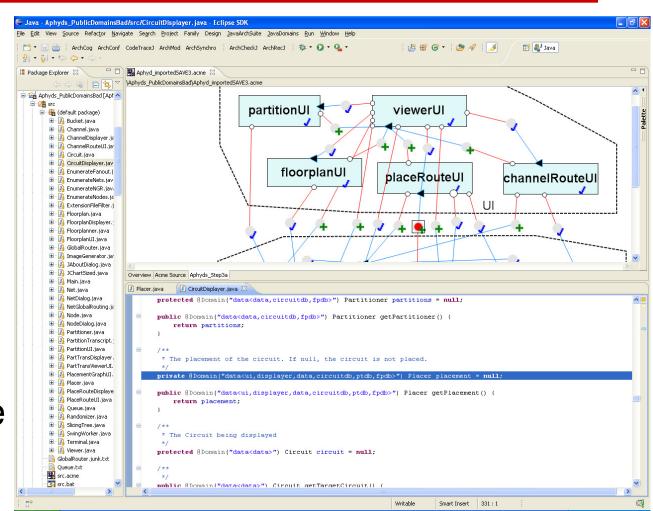


Scholia: trace finding to code; iterate



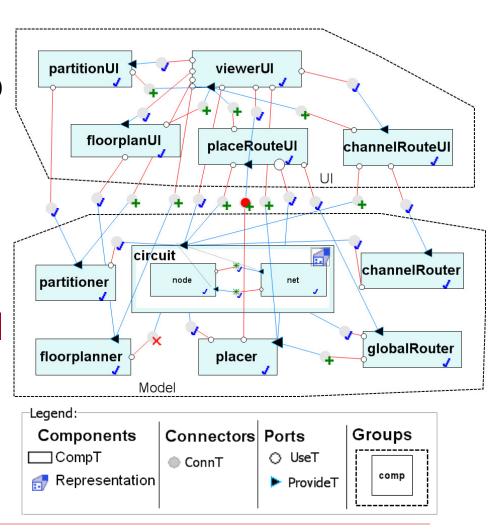
CodeTraceJ: trace from runtime architecture to lines of code

- Trace finding to code
- Previously, only UML class diagrams supported this feature



Aphyds: summary of findings

- Callback from placer in MODEL to placeRouteUI in UI (significant in a multi-threaded app)
- Many connections really bi-directional



Evaluation of the SCHOLIA approach

Several extended examples and field study

System	Size	Comments
JHotDraw	15 KLOC	Designed by experts in object- oriented analysis and design
HillClimber	15 KLOC	Designed by undergraduates at UBC
Aphyds	8 KLOC	Original developer drew architecture
LbGrid	30 KLOC	Extracted object graphs, showed them to outside developer
CryptoDB	3 KLOC	Compelling target architecture designed by security expert

Limitations and Related Work

Limitations

False positives

- Possible in any sound static analysis
- Few when developer fine-tunes annotations, controls abstraction steps, structural comparison, etc.
- Type system expressiveness limitations
 - Annotated systems have warnings remaining
 - Can incorporate some published research
- Manual annotations
 - Impractical without annotation inference
 - Inference active area of research

Previous static analyses

- Object graph analyses
 - Without relying on annotations
 [Jackson and Waingold, ICSE'99,TSE'01]
 [O'Callahan, Ph.D. thesis'01] [Spiegel, Ph.D. thesis,'02]
 - Using non-ownership annotations [Lam and Rinard, ECOOP'03]
 - Some unsound w.r.t. aliasing or inheritance
- Points-to analysis
 - e.g., [Milanova et al., TOSEM'05]
- Shape analysis
 - e.g., [Sagiv et al., POPL'99]

Architectural conformance: state-of-the-art

Dynamic analysis

[Sefika, Sane and Campbell, ICSE'96] [Schmerl, Aldrich, Garlan *et al.*, ICSE'04, TSE'06]

- Runtime instrumentation and monitoring
- Throw runtime exception when violation occurs
- Cannot check all possible program runs
- Conformance by design
 - Code generation [Shaw et al., TSE'95]
 - Recent trend in model-driven development
 - Hard to use for legacy systems
 - More general to use extract-abstract-check

Architectural conformance: state-of-the-art (continued)

Library-based solutions

[Medvidovic et al., FSE'96] [Malek, Mikic-Rakic and Medvidovic, TSE'05]

- Relies on style guidelines
 [Luckham and Vera, TSE'95]
- No tools to automatically enforce them

Language-based solutions

ArchJava [Aldrich et al., ECOOP'02]

- Specify architectural constructs in code
- Restrictions on object references
- Require re-engineering existing systems

[Aldrich, Chambers and Notkin, ICSE'02]

[Abi-Antoun and Coelho, WICSA'05]

[Abi-Antoun, Aldrich and Coelho, JSS'07]

Summary

- First approach, SCHOLIA, to guarantee at compile-time communication integrity between arbitrary Java code and hierarchical intended runtime architecture
 - Uses backward-compatible statically typecheckable annotations
 - Instead of languages or libraries
- Evaluation on real systems very promising