Static Extraction and Conformance Checking of Runtime Architectures

Marwan Abi-Antoun (Advisor: Jonathan Aldrich)
mabianto@cs.cmu.edu Carnegie Mellon University
institute for SOFTWARE RESEARCH

How accurate is a manually generated runtime architecture?

A ***runtime architecture*** shows **organization** of system in terms of runtime entities and their interactions. Crucial for reasoning about **performance**, **security**, **reliability**, etc.

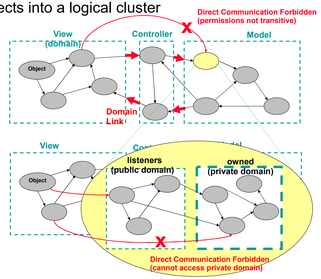
Extraction extracts **as-built** architecture from implementation. **Conformance checking compares** as-built architecture to **as-designed architecture**.

Requirements on an Adoptable Solution

To be adoptable, approach must support:
- **Existing languages**, **frameworks**, **patterns** — i.e., **no language extensions**
- Analysis for object-oriented programs must deal with **aliasing**, **inheritance**, etc.
- Dynamic analysis cannot prove that program always satisfies particular property
- **Static checking** is ideal if can be achieved and is **sound**
- **Soundness**: reveal all entities and relations that may exist at runtime
- **After-the-fact checking:** do not assume code generation, monitoring of changes, etc.

Ownership Domains [Aldrich and Chambers, ECOOP'04]

- An ***ownership domain*** groups related objects into a logical cluster
  - Each object is in exactly one domain
  - Object contains one or more domains
- Domain name conveys **design intent**
- Domain can represent a **runtime tier**
  - E.g., "Model", "View", "Controller"
- References can cross domain boundaries only if there is a ***domain link*** between the two domains



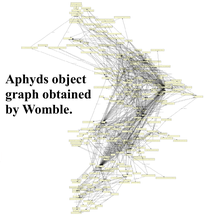
Implementation:
- Uses **Java 1.5 annotations**
- Does not require language extensions
- Uses Eclipse Java infrastructure

Key insight: Ownership domain annotations enable the extraction of sound hierarchical runtime object graphs using static analysis.

- **Rewriting Rules:** relate runtime graph to annotated program
- **Proof of Soundness:** relate store typing to extracted runtime graph
- **Evaluation:**
  - Case Studies (JHotDraw, HillClimber, Aphyds)
  - Field Study (LbGrid)
- Concrete application:
  - **Conformance checking**

System	Size	Comments
JHotDraw	15 KLOC	Designed by OOA&D experts
HillClimber	15 KLOC	Designed by undergraduates
Aphyds	8 KLOC	Original developer drew architecture
LbGrid	30 KLOC	Part of 250-KLOC commercial system

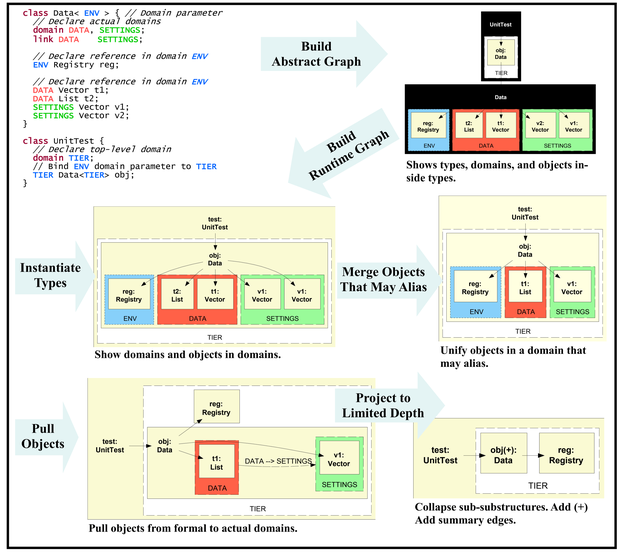
Previous static analyses:
- Show **detailed interactions**
- **No architectural abstraction**
- Do not scale to large programs
- Low-level objects at same level as important objects
- Sometimes, do not handle **aliasing**
- **Not comparable** to human-generated as-designed architectures



Aphyds object graph obtained by Womble.

Static Extraction of Hierarchical Runtime Architectures

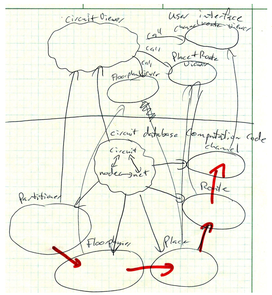
Provide **architectural abstraction** by **ownership hierarchy** and by **types**
- **Summarization**: summarize **multiple objects** at runtime with one object
- **Aliasing**: if two variables might alias at runtime, show as single element
- **Hierarchy**: **primary objects** appear at top-level. Each primary object has sub-structure that contains **secondary objects**, until low-level objects are reached
- **Expanding** or **collapsing** sub-structures enables **varying abstraction level**
- **Object Lifting**: show all objects that are in domain at runtime
- **Edge Lifting**: lift any edges to or from elided object to visible ancestor



Conformance Checking Strategy

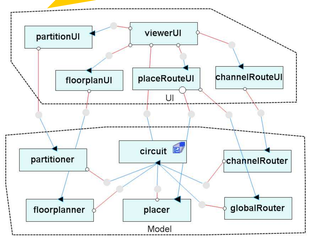
We extend the **extract-abstract-check** strategy
- **Document as-designed** architecture
- **Abstract as-built** architecture from code
  - **Annotate** code to clarify architectural intent
  - **Extract** sound approximation of runtime object graphs
  - **Abstract** into as-built runtime architecture
- **Check** and **measure** structural conformance
  - Structurally **compare as-built and as-designed** views
  - Display results graphically on as-designed view
  - Compute **measure** of conformance
  - **Trace to code unexpected conformance finding**
  - Fix **architectural violations** in code
  - Adjust as-designed architecture

Illustration of End-To-End Approach



Aphyds as-designed architecture, drawn by original developer.

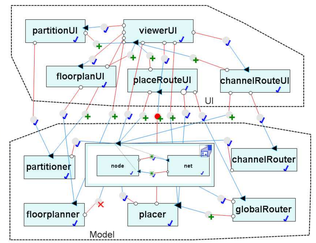
Document



In Eclipse AcmeStudio perspective, document as-designed architecture in architecture description language.

View Conformance

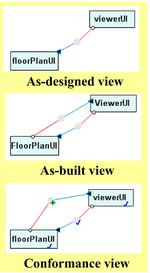
Conformance results:
- **Missing** component, partitionUI
- **Callback** from placer to placeRouteUI
- Many connections really **bi-directional**

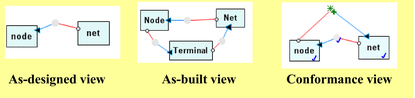


ArchConf conformance checking tool displays results Study conformance view. Investigate differences.

Conformance Checking Analysis

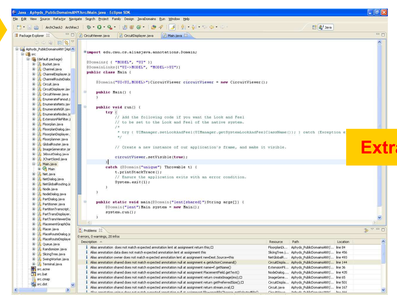
Goal is **not to make the two views identical**
- Additional sub-structures in as-built architecture
- Innocuous differences, e.g., **renames**
- As-designed **view more authoritative**
- Included components more relevant than omitted ones
- Names convey architectural intent
- Allow as-built view to contain low-level details
- Account **for all communication** in as-built view that is not in as-designed view
- Include **transitive communication** through elided objects





Conformance check highlights key differences between as-built and as-designed views:
- **Convergence**: node or edge **in both** as-built and in as-designed view ✓
- **Divergence**: node or edge **in as-built** but **not in as-designed** view +
- **Absence**: node or edge **in as-designed** but **not in as- built** view ✕

Annotate

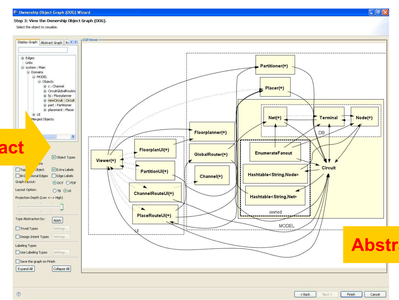


```
1  class Circuit {
2    public domain DB; // Declare public domain DB
3    domain OWNED; // Declare private domain OWNED
4    DB Node node; // Declare Node reference in DB
5    // Outer annotation is for container; inner one for its elements
6    OWNED Hashtable<String, DB Node> nodes;
7  }
8  class Viewer<D> { // Declare domain parameter M
9    M Circuit circuit; // Declare Circuit reference in M
10 }
11 class Main {
12   domain MODEL, UI; // Declare top-level domains
13   MODEL Circuit circuit;
14   // Bind domain parameter M to actual domain MODEL
15   UI Viewer<MODEL> viewer;
16 }
```

Annotations express:
- object encapsulation
- logical containment
- architectural tiers
- communication permissions (domain links)

In Eclipse Java development perspective, add ownership domains as Java 1.5 annotations. ArchCheckJ typechecking tool shows warnings in Eclipse problem window.

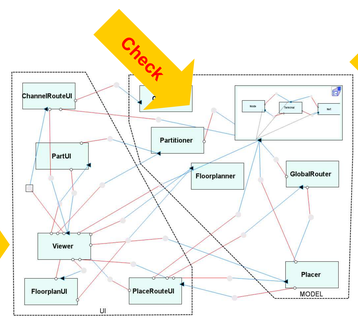
Extract



ArchRecJ architectural extraction tool extracts representation of as-built hierarchical runtime object graph.

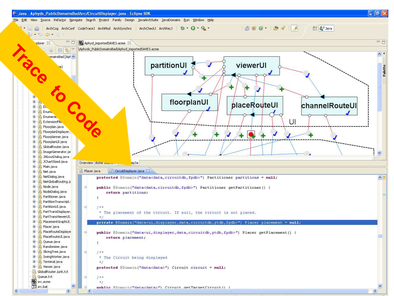
Abstract

Check



ArchCog architectural abstraction tool maps hierarchical runtime object graph to as-built runtime architecture.

Trace to Code



Relate architectural elements to code. Fix serious architectural violations. Or refine as-designed architecture.