OOGRE: Inferring Ownership Domains from Refinements

WAYNE STATE UNIVERSITY

Ebrahim Khalaj **Marwan Abi-Antoun** {mekhalaj, mabiantoun}@wayne.edu https://github.com/aroog

1 Understanding Object Structures

- Better tools for understanding object structures are needed
- Answer key developer questions such as:
  - Can (malicious) clients obtain direct references to mutable objects?
  - Does the runtime structure follow a two-tiered design?
- Our proposal: developer use refinement to expressing object hierarchy
  - From refinements, static analysis [1] infers valid type qualifiers
  - Qualifiers can be added to the code for documentation
  - List of refinements can be replayed on the evolving code
  - Another static analysis [3] abstractly interprets the code with qualifiers to extract sound hierarchical abstract object graphs
  - Constraints on the object graphs enforce architectural policies [3], security policies [2], etc.

2 Simple Ownership Domains

Ownership type qualifiers express object hierarchy and aliasing invariants that cannot be directly expressed in mainstream programming languages.

Modifier: an ownership keyword
Qualifier: a pair of modifiers <*p*, *q*> on a variable in the program

Simple Ownership Domains modifiers:
- one private domain **owned**;
- one public domain **PD**;
- **owner**: domain of the **this** object;
- one domain parameter **p**;
- one global domain **shared**;
- unaliased object is **unique**;
- borrowed object is **lent**;

```
class Circuit<...> {
  private domain owned;
  public domain PD;
  Vector<owned, ...> nodes;
}

class Vector<owner, p> {
  private domain owned;
  public domain PD;
  Node<owned, p> head;
  Iterator<PD, ...> getIterator() {
  }
}
```

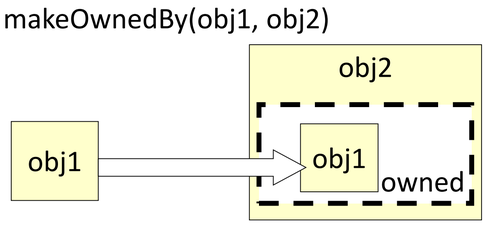
Manually adding qualifiers to existing code is burdensome. Using **OOGRE** [1], developers perform refinements, from which qualifiers are inferred.

3 Refinements as Abstract Qualifiers

Refinement: abstraction of a qualifier, pins first part *p* of qualifier pair <*p*, *q*>
Refinements express ownership relations between objects.
- Each object contains zero or more domains;
- Each object is in exactly one domain.
- Domain can be public or private;

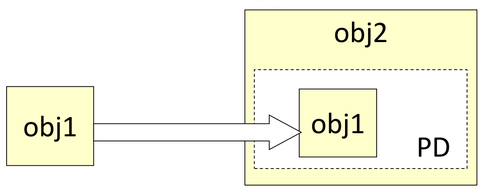
Domains also give some precision about aliasing:
- Objects in the same domain **may** alias;
- Objects in different domains **may not** alias;

makeOwnedBy(obj1, obj2)



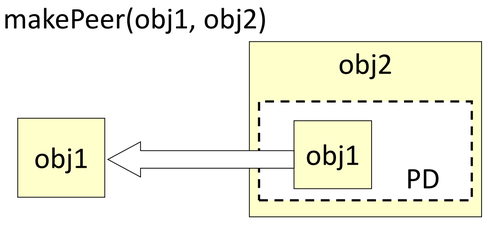
Make obj1 strictly encapsulated within obj2

makePartOf(obj1, obj2)



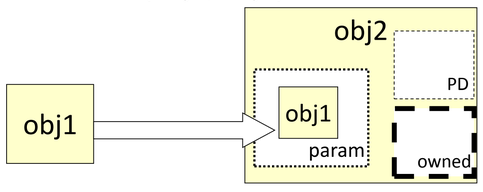
Make obj1 logically contained by, i.e., part-of, obj2

makePeer(obj1, obj2)



Make obj1 peer with obj2

makeParam(obj1, obj2)

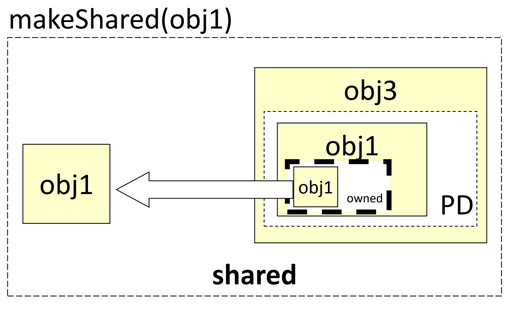


Place obj1 in parametric domain of obj2

Developers perform refinements using operations on an abstract object graph.
Or by adding annotations on variables in the code.

```
class Circuit {
  Vector /*makeOwnedBy*/ nodes;
}
```

makeShared(obj1)

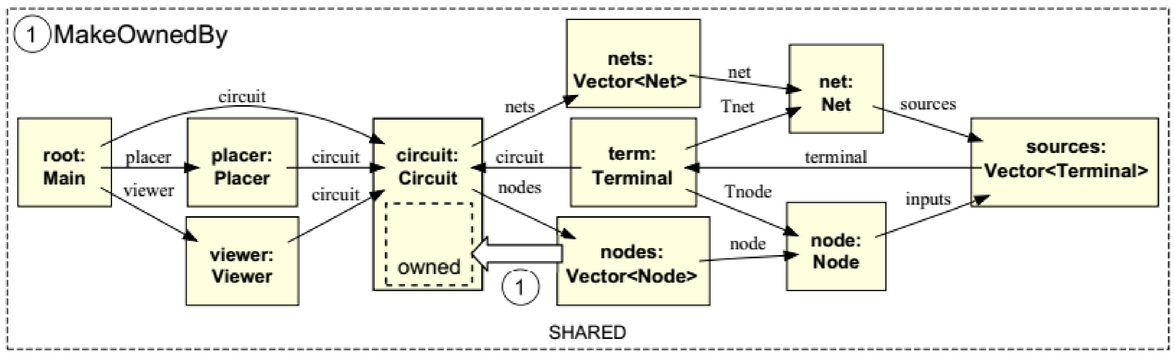


Place obj1 in the global domain **shared**

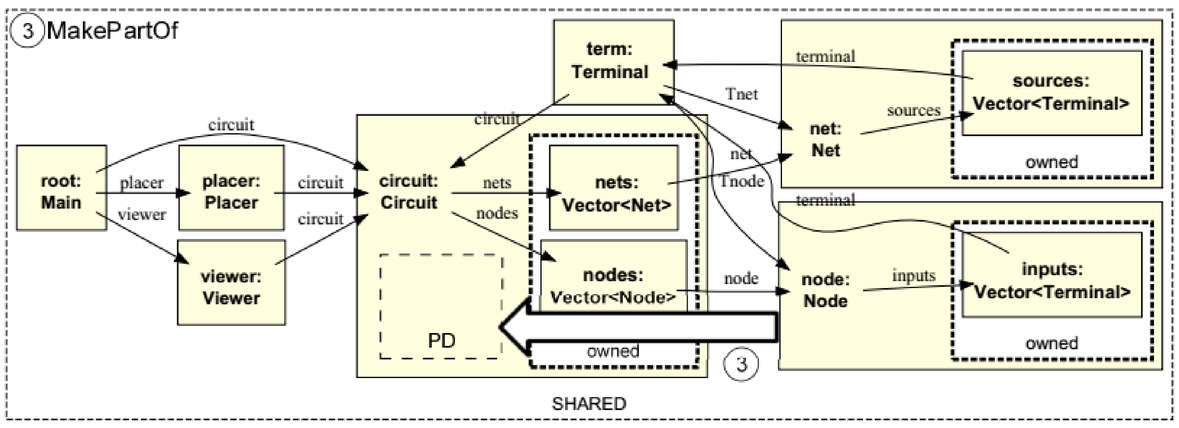
Inference analysis sets each refinement status:
- Status = done ➔ code matches developer's design intent
- Status = skipped ➔ mismatch between code and developer's design intent
  - Developers may have discovered unintended aliasing;
  - Developers change the code and re-attempt refinement; or
  - Developers refine their understanding of the code and change the refinement.

4 Refinements Express Design Intent

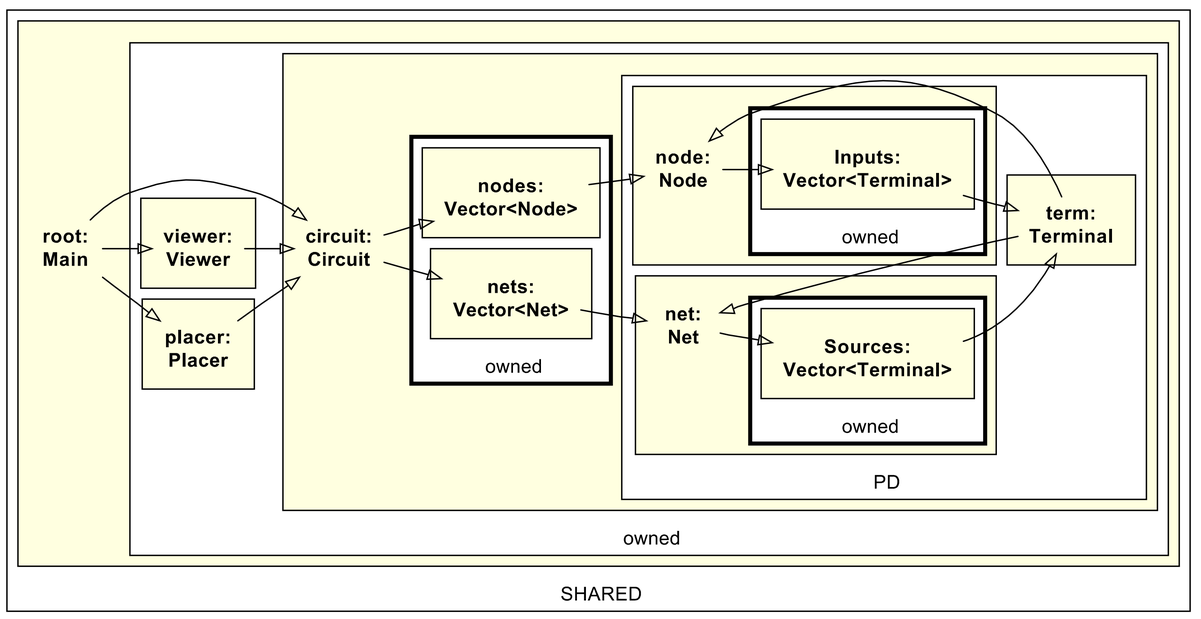
As the first step, OOGRE infers default qualifiers, and extracts a flat object graph.



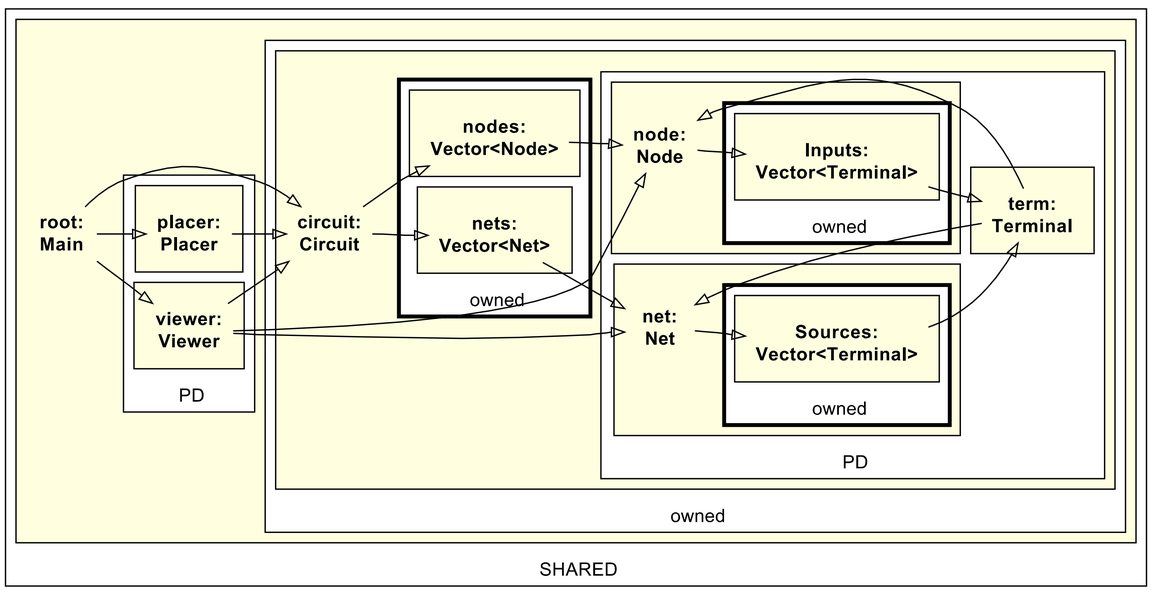
Refinement **makedOwnedBy** makes vectors strictly encapsulated to prevent clients from breaking important invariants such as clearing the Node objects.



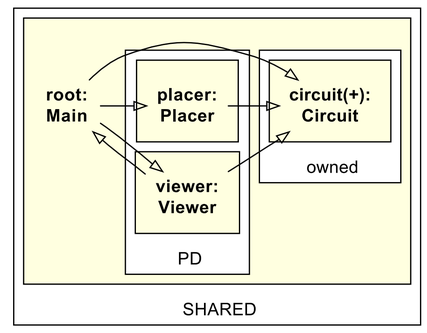
Refinement **makePartOf** makes Node, Net and Terminal objects conceptually part of the Circuit object, which creates them and manages them but still makes them accessible to the outside.



Using the two domains on the root object, additional refinements express a two-tiered Document-View architecture



Objects in hierarchical object graph can be collapsed to convey high-level understanding, and expanded to reveal more detail.



5 For More Information

[1] E. Khalaj and M. Abi-Antoun. Inferring Ownership Domains from Refinements. In International Conference on Generative Programming: Concepts & Experience (GPCE), 2018.
[2] R. Vanciu and M. Abi-Antoun. Finding Architectural Flaws using Constraints. In Automated Software Engineering (ASE), 2013.
[3] M. Abi-Antoun and J. Aldrich. Static Extraction and Conformance Analysis of Hierarchical Runtime Architectural Structure using Annotations. In Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), 2009.
[4] J. Aldrich and C. Chambers. Ownership Domains: Separating Aliasing Policy from Mechanism. In European Conference on Object-Oriented Programming (ECOOP), 2004.