

A Static Analysis for Extracting Runtime Views from Annotated Object-Oriented Code

Marwan Abi-Antoun

Jonathan Aldrich

Institute for Software Research

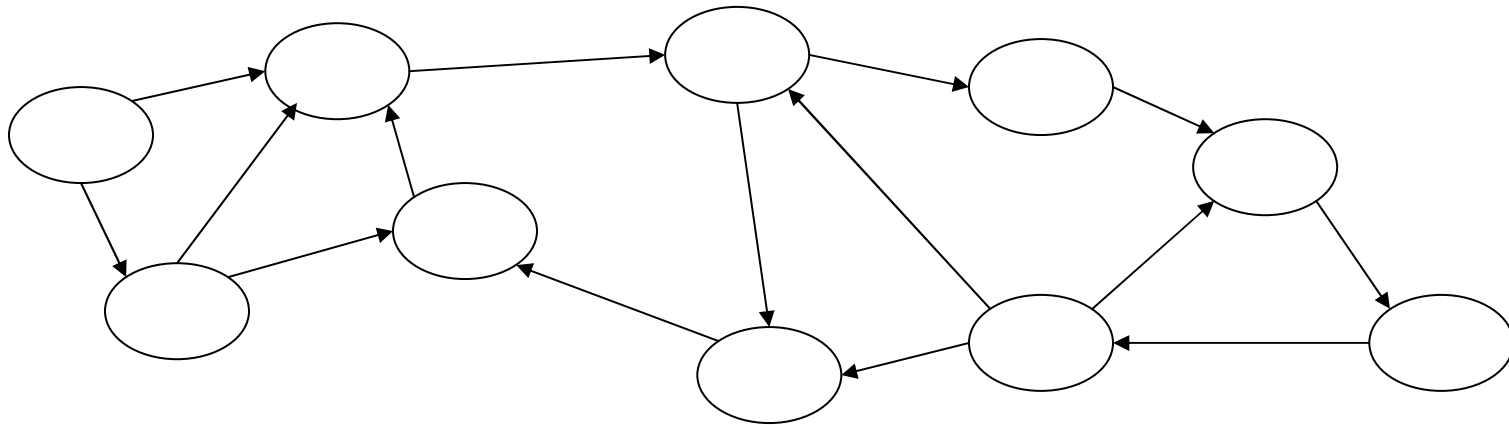
Carnegie Mellon University



OOPSLA 2006

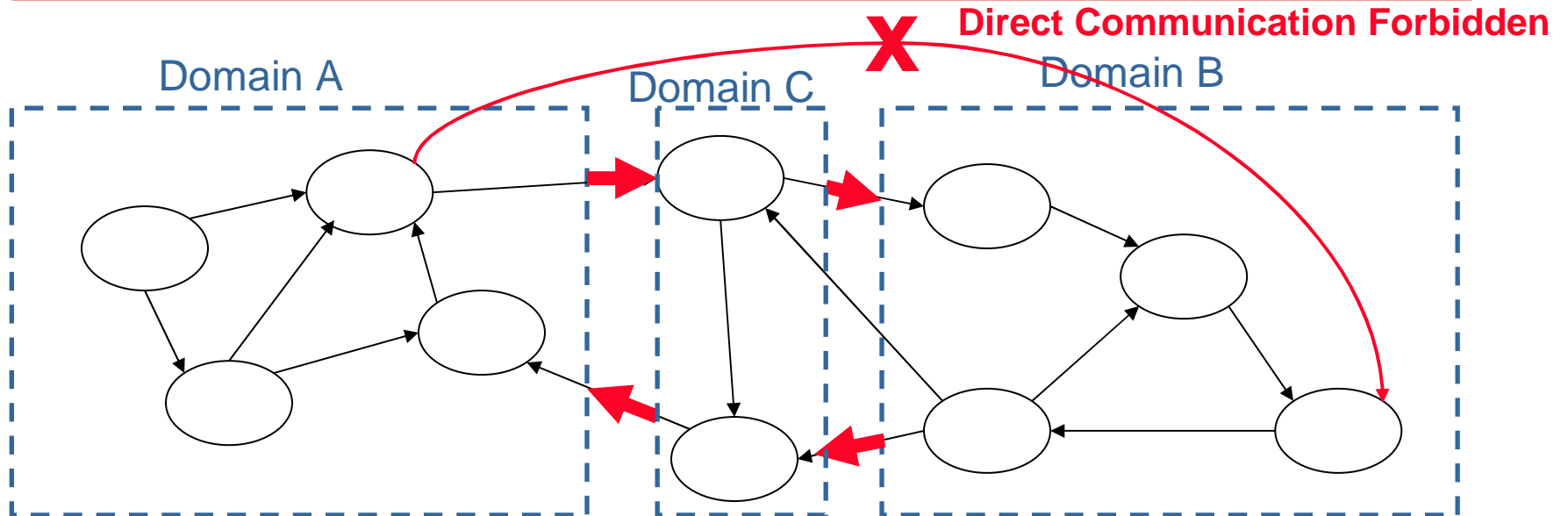
ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications
October 22-26, Portland, Oregon, USA

Object-oriented programs at runtime



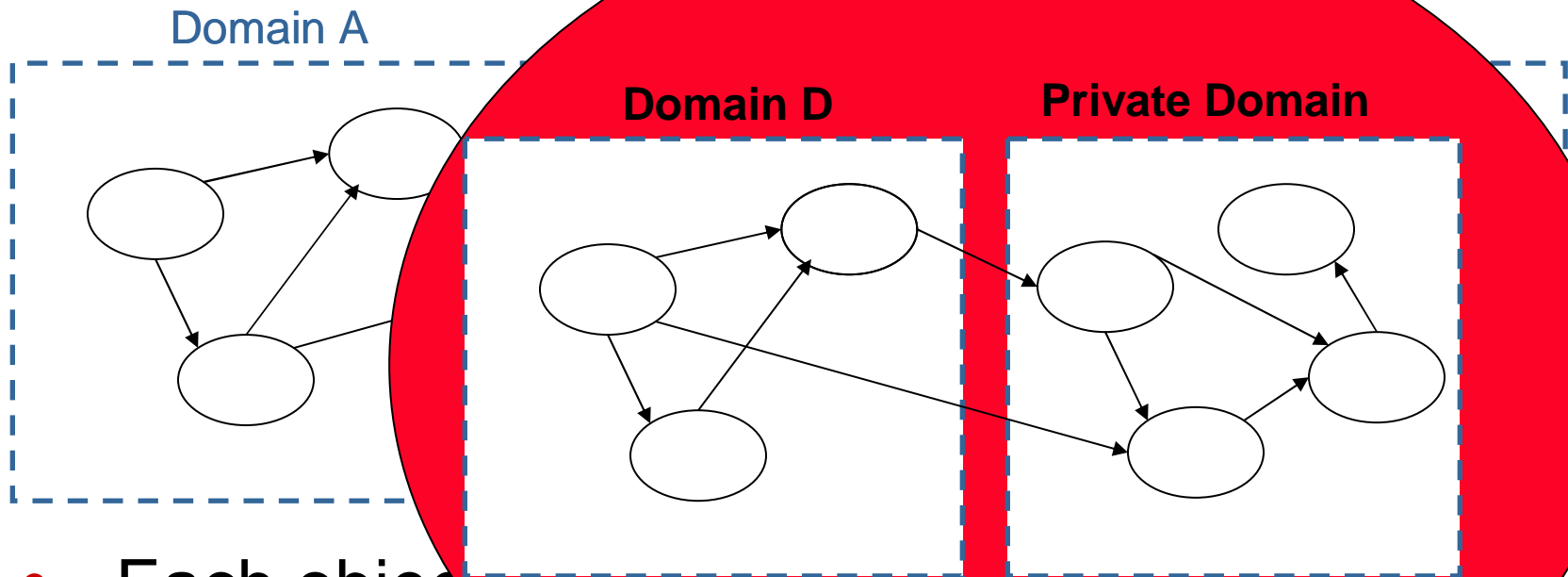
- Object graph where nodes represent objects, edges represent creation, usage, reference
- **Flat** general directed graph with cycles
- Evolves when the program is running
- Static analyses can conservatively approximate it

Abstracting runtime views



- Groups of objects into “ownership domains”
- Domain names specify abstract design intent
- Abstract communication into “links”
- Objects communicate only when permitted

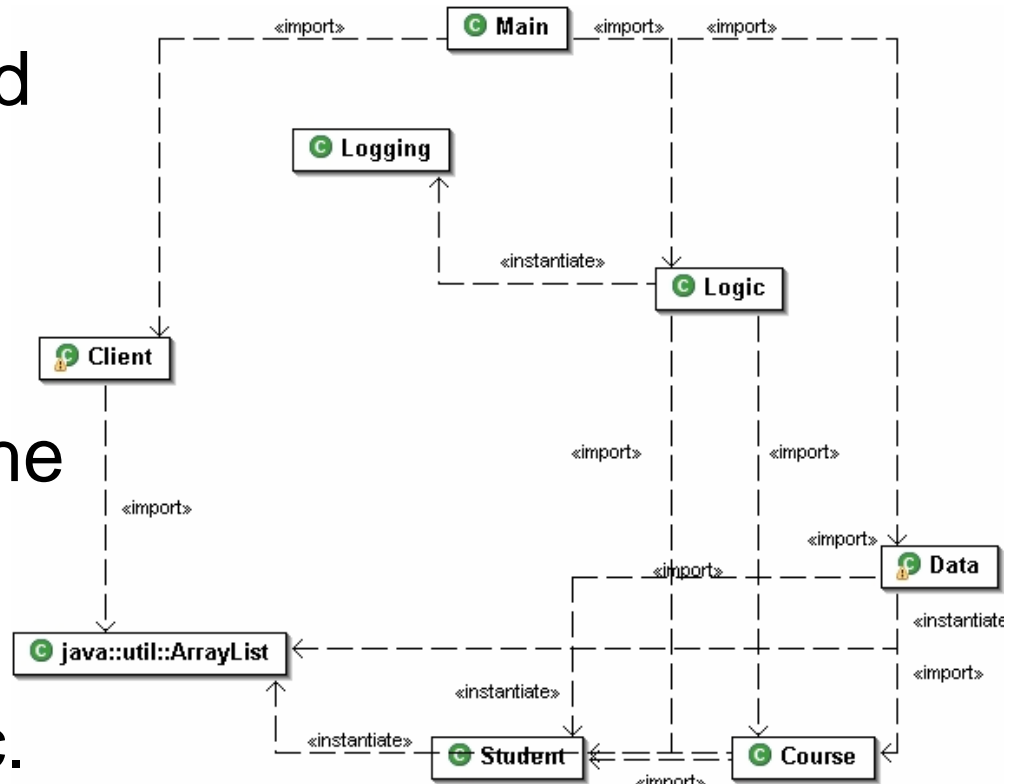
Add hierarchy for scalability



- Each object
- An object com
- A domain can be p (internal)
- Hide non-architecturally significant objects

Runtime views vs. module views

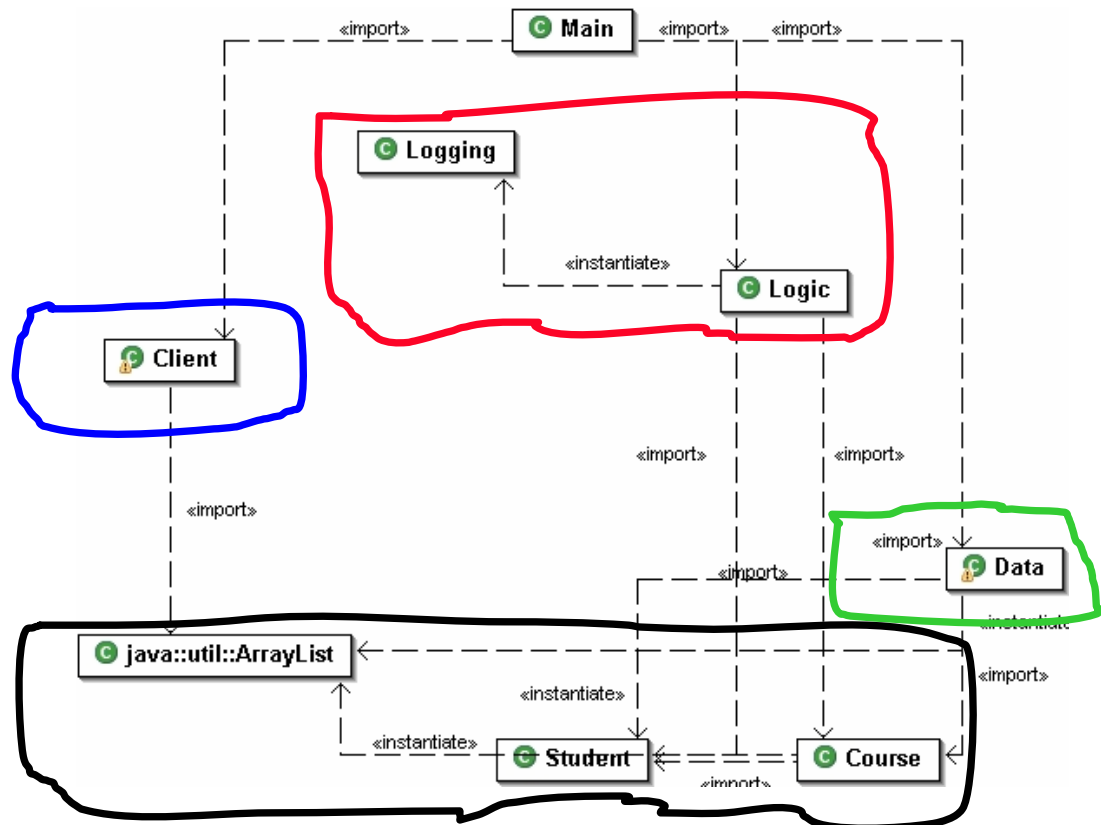
- Module views extracted by several tools
- Do not distinguish between conceptually different instances of the same class
- Extra details: abstract classes, interfaces, etc.
- No hierarchy
- No abstract design intent



Source: UML diagram produced by Omondo Eclipse UML tool

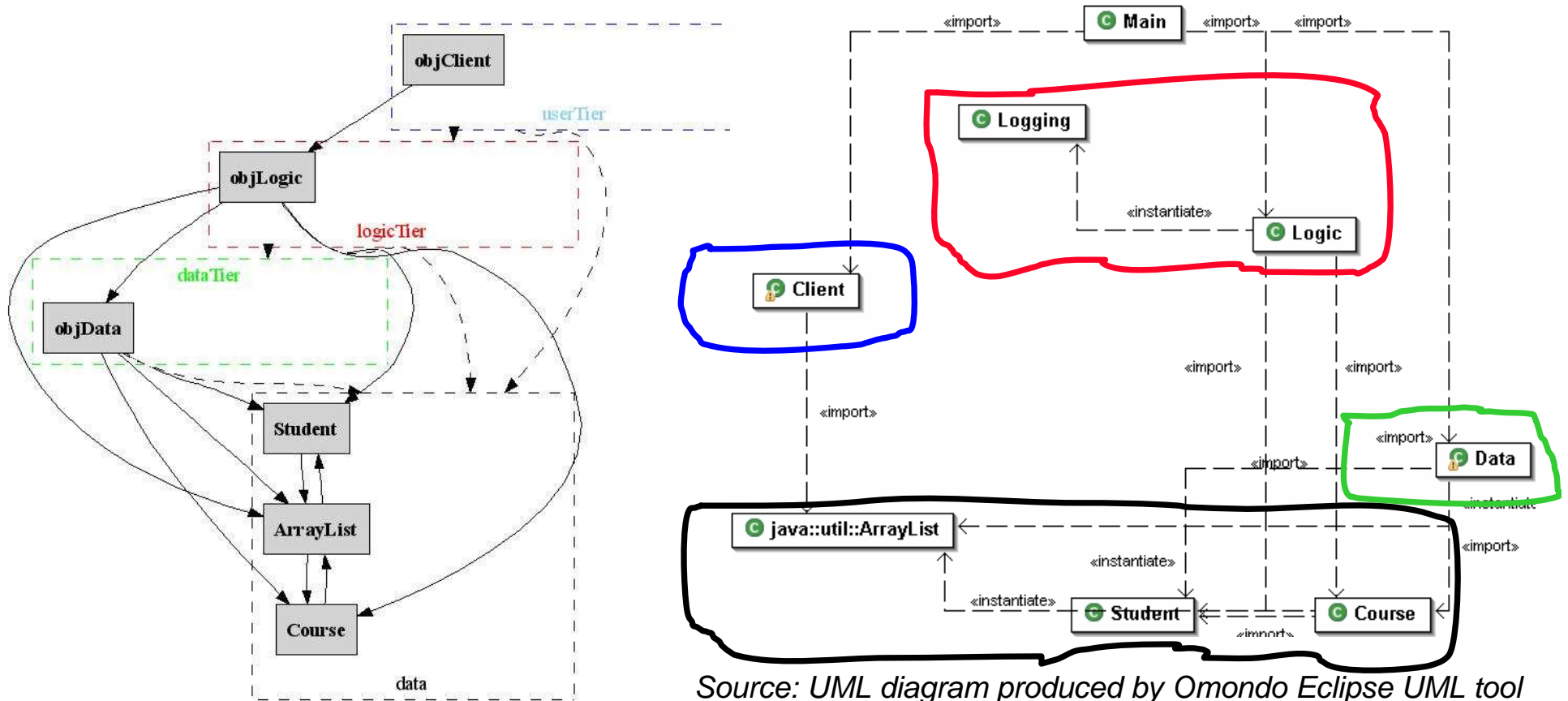
Runtime views vs. module views

- Add tiers
 - **User**
 - **Logic**
 - **Data**
 - **State**
- Add hierarchy
 - Logic
 - Logging



Source: UML diagram produced by Omondo Eclipse UML tool

Runtime views vs. module views



Source: UML diagram produced by Omondo Eclipse UML tool

Annotation language

- @Domains: declare ownership domains
- @DomainParams: declare *formal* domain parameters
- @DomainLinks: declare domain link specifications
- @DomainInherits: specify parameters for supertypes
- @DomainReceiver: specify annotation on receiver
- @Domain: specify object annotation, *actual* domain parameters and (optionally) array parameters
“*annotation*<*domParam*, ...> [*arrayParam*, ...]”
- Annotation:
 - Special: “lent”, “unique”, “owned”, “shared”
 - Common: “iters” or “obj.iters”

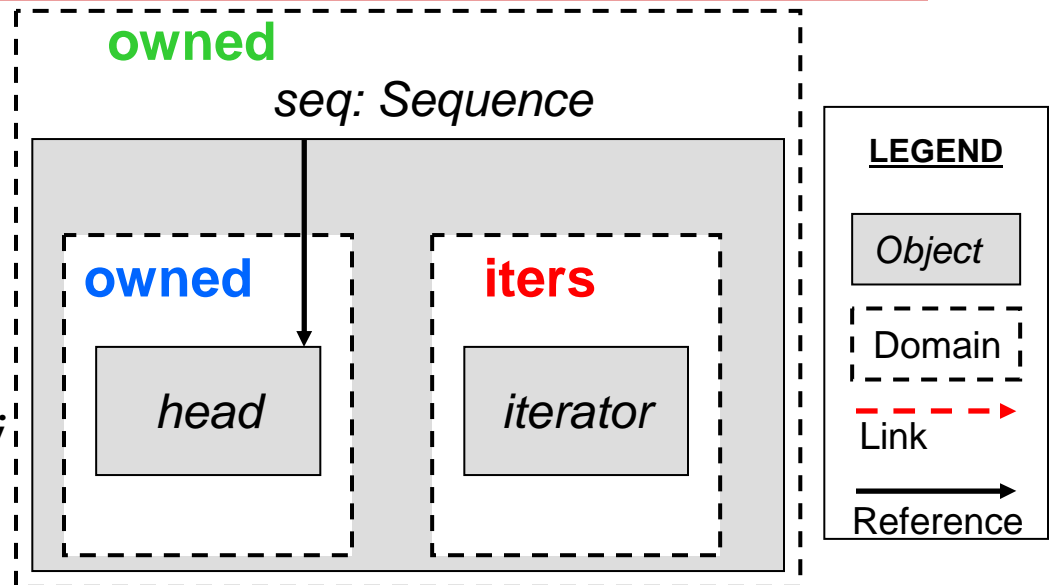
Special alias types

- **owned**: instance confined within object (default domain)
- **unique**: instance passed linearly from one object to another
- **lent**: temporary alias within method
- **shared**: shared persistently or globally

Public, private ownership domains

```
@Domains ( { "iters" } )
class Sequence {
  @Domain ( "owned" ) Cons head;

  public @Domain ( "iters" )
  Iterator getIter () {
    return new Iterator ( head );
  }
}
```



```
@Domain ( "owned" ) Sequence seq = new Sequence ();
```

- Every object is in exactly one domain
- E.g., list in domain **owned**; iterators in domain **iters**
- Every object can have one or more domains
- E.g., domains **owned** and **iters** declared in Sequence

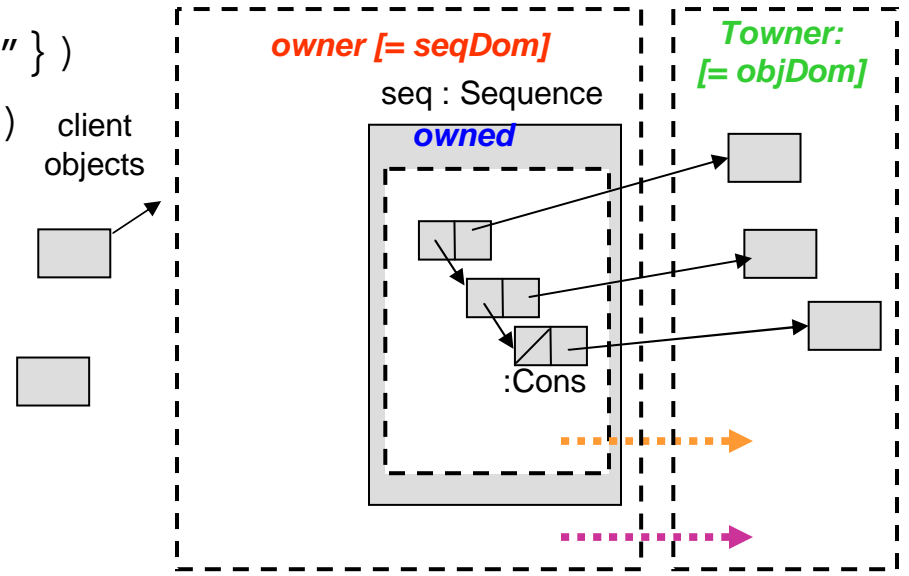
Domain parameters + Links

```

@DomainParams ( { "Towner" } )
@DomainAssumes ( { "owner -> Towner" } )
@DomainLinks ( { "owned -> Towner" } )
class Sequence {
  @Domain ( "owned<Towner>" )
  Cons head;
  ...
}
@DomainParams ( { "Towner" } )
class Cons {
  @Domain ( "Towner" ) Object obj;
  @Domain ( "owner<Towner>" ) Cons next;
}

```

Cons.owner == Sequence.owned



```

@Domain ( "seqDom <objDom>" )
Sequence seq = new ...

```

- Add domain parameter to hold elements in list
- Link declarations give *Sequence.owner*, *Cons.owner* (*Sequence.owned*) access to parameter *Towner*

Ownership Object Graph (OOG)

- Show object instances with:
 - nested domains and
 - objects inside of those domains
- Additional heuristics for visualization
 - Merge object instances
 - Lift objects instances
 - Add edges
 - Merge field links from base classes

Step 3: View the Ownership Object Graph (OOG)

Select the object to visualize.

Visual Graph | Abstract Graph

- Edges
- Links
 - seq, iters -> seq, owned
 - seq, iters -> seq, Towner
 - seq, iters -> system, state
 - seq, owned -> seq, Towner
 - seq, owned -> system, state
- system : SequenceClient

- Top-level OOG for Sequence example
- Solid edges are field references
- Dashed edges are domain-links
- Dashed borders are actual domains

Show Domain Links
 Show Variable Names
 Show Object Types
 Show Field Links

< Back Next > Finish Cancel

Step 3: View the Ownership Object Graph (OOG)

Select the object to visualize.

Visual Graph | Abstract Graph

Edges
Links
system : SequenceClient
Domains
owned
Objects
seq
shared
Object
state
Objects
int5 : Integer
obj : Object*
unique
Objects
Merged Objects

Display Subgraph
Show Internals
Show Formals
Visible

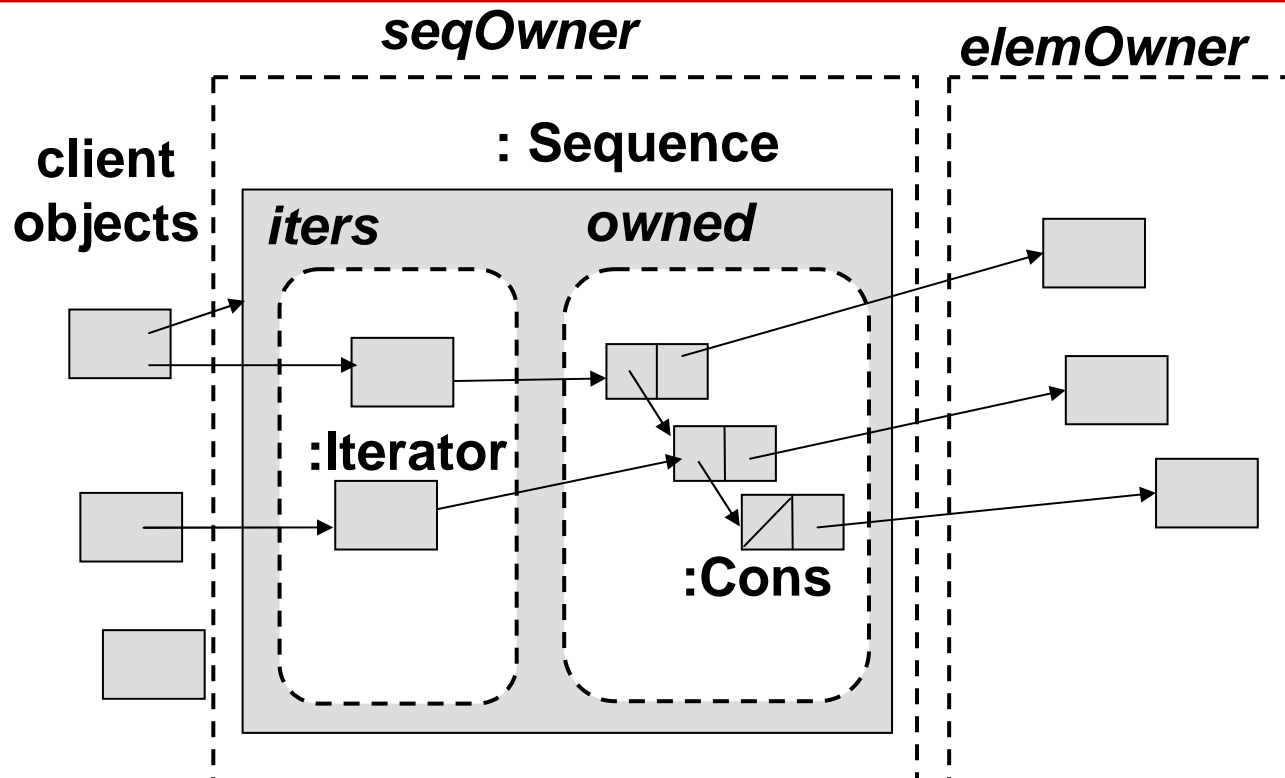
owned

- Second-level OOG for Sequence example
- Showing actual domains (dashed border)
 - owned (private domain)
 - iters (public domain)
- Showing formal domains (dotted border)
 - Towner: domain parameter
 - Use to store elements of the list

Show Domain Links
 Show Variable Names
 Show Object Types
 Show Field Links

< Back Next > Finish Cancel

OOG Visualization: intuition

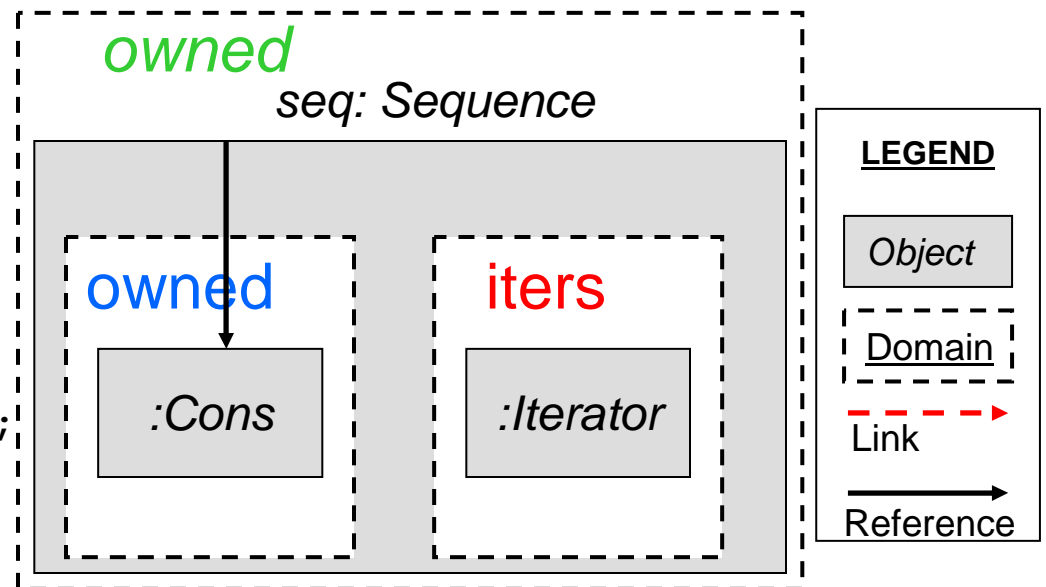


- Merging objects of same type
- Pulling objects into “actual” domain

Merging object instances

- Merge objects of the same type that are owned by the same domain.

```
@Domains ( { "iters" } )  
class Sequence {  
  @Domain ( "owned" ) Cons head;  
  
  public @Domain ( "iters" )  
  Iterator getIter () {  
    return new Iterator ( head );  
  }  
}
```



```
@Domain ( "owned" ) Sequence seq = new Sequence ();
```

Lifting object instances

- Lift each object declared in formal domain transitively to show it in actual domain

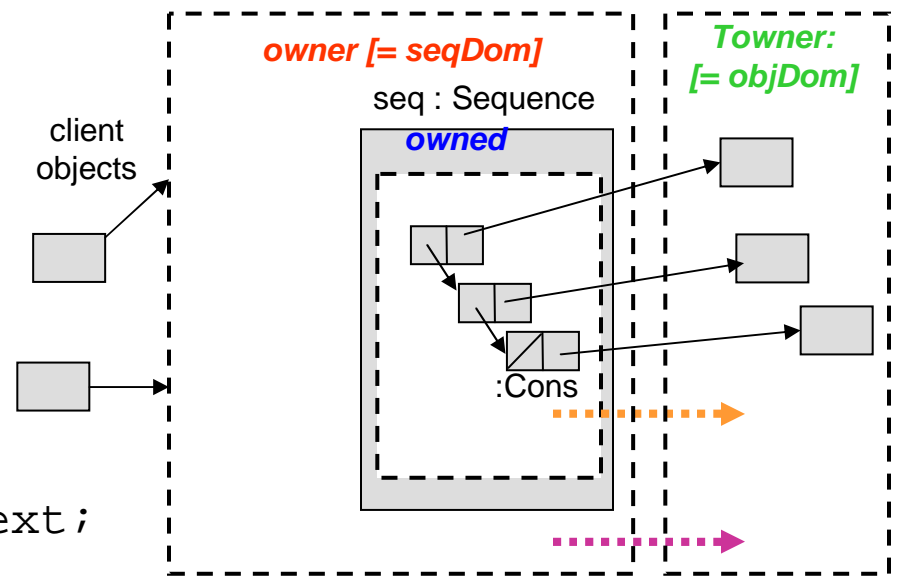
```

@DomainParams ( { "Towner" } )
class Sequence {
  @Domain ( "owned<Towner>" )
  Cons head;
  ...
}

@DomainParams ( { "Towner" } )
class Cons {
  @Domain ( "Towner" ) Object obj;
  @Domain ( "owner<Towner>" ) Cons next;
}

```

Cons.owner == Sequence.owned



```

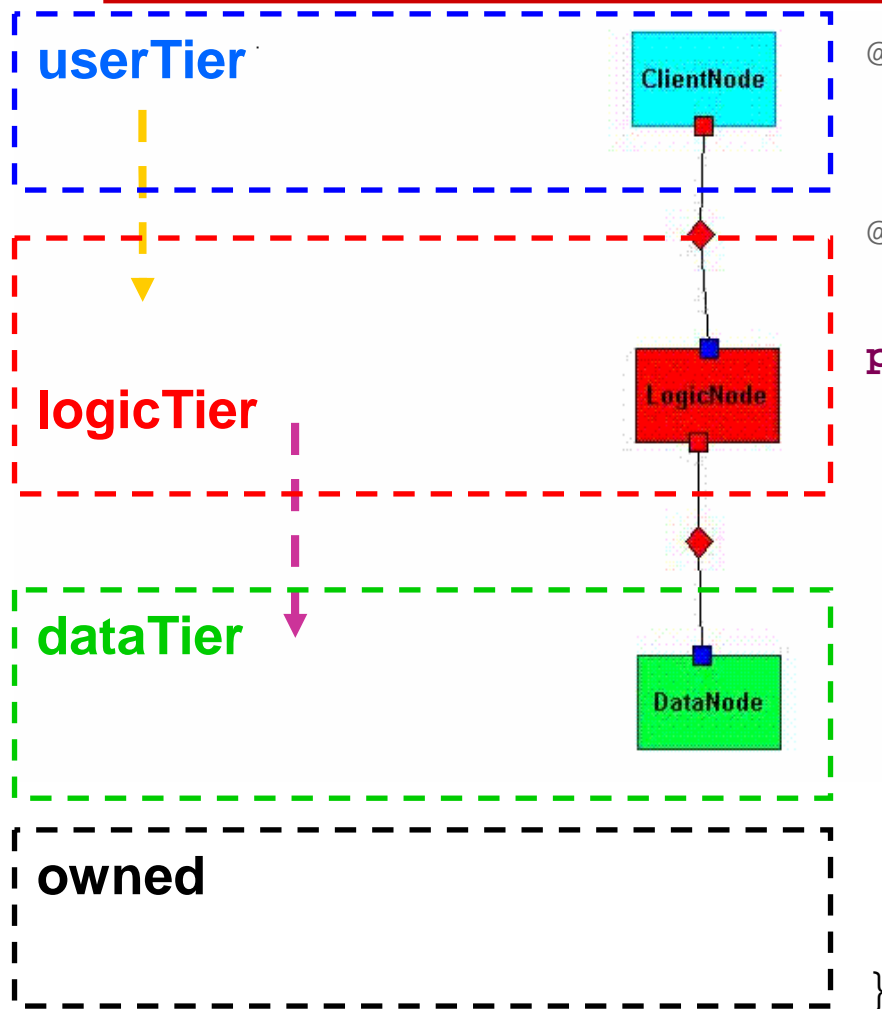
@Domain ( "seqDom < objDom" )
Sequence seq = new ...

```

Adding edges

- Domain edges
 - Domain link specifications (permissions)
 - Correspond to “architectural connections”
- Object edges
 - Creation edges: object allocation
 - Reference edges: field references, etc.
 - Usage edges: field access, method invocation, etc.

Demonstration: Simple 3-tier system



```
@Domains({ "userTier",
            "logicTier",
            "dataTier" })

@DomainLinks({ "userTier -> logicTier",
               "logicTier -> dataTier" })

public class Main {
    @Domain("dataTier<owned>")
    private Data objData = null;

    @Domain("logicTier<dataTier, owned>")
    private Logic objLogic = null;

    @Domain("userTier<logicTier, owned>")
    private Client objClient = null;
}
```

Demo: Courses OOG

- Step 0: Annotate program
- Step 1: Setup
 - Select Java project
 - Select top-level class
- Step 2: Select types to include/exclude
 - Exclude library types, etc.
- Step 3: Display object graph
 - Show/hide object labels/types
 - Show/hide object internals

Step 1: Select options for the Ownership Object Grapher (OOG)

Select the Java project to visualize. If the project is not in the workspace, please import it first.

1. Select the Java project to analyze:

Courses_Generics

2. Select the top-level class that you are interested in seeing the structure of:

courses.Main

3. Select the edge types to display:

- Show creation edges
- Show reference edges
- Show usage edges

4. Select the analysis options:

- Use ownership annotations

< Back

Next >

Finish

Cancel

Step 2: Select types for the Ownership Object Graph (OOG)

Select the Java types to include and exclude.

Check the Java types to include; uncheck the types to exclude:

- src
 - !JavaElementLabels.default_package!
 - courses
 - Client.java
 - courses
 - Client
 - Course.java
 - courses
 - Course
 - Data.java
 - courses
 - Data
 - IData.java
 - courses
 - IData
 - ILogic.java
 - courses
 - ILogic
 - Logging.java
 - courses
 - Logging
 - Logic.java
 - courses
 - Logic
 - Main.java
 - courses
 - Main
 - RWLock.java
 - courses
 - RWLock
 - Sequence.java
 - courses
 - Cons
 - Iterator
 - Sequence
 - SequenceIterator
 - Student.java
 - courses
 - Student
 - aliasjava.jar!JavaElementLabels.concat_string!C:\Eclipse3.1.1\runtime-workspace\Courses_Generics
 - !JavaElementLabels.default_package!
 - edu.cmu.cs.aliasjava.annotations
 - META-INF
 - JRE System Library [jdk1.5.0]

< Back

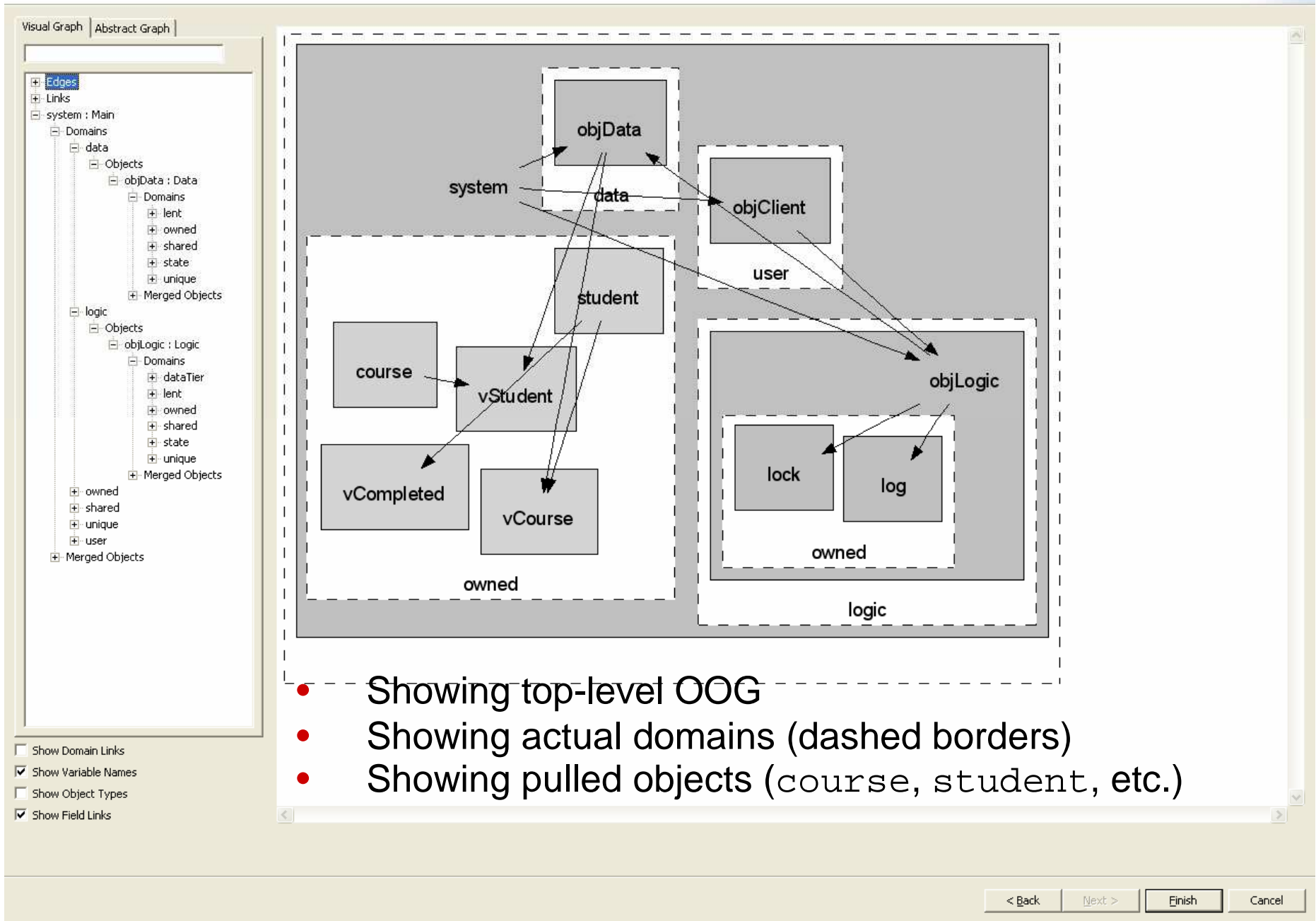
Next >

Finish

Cancel

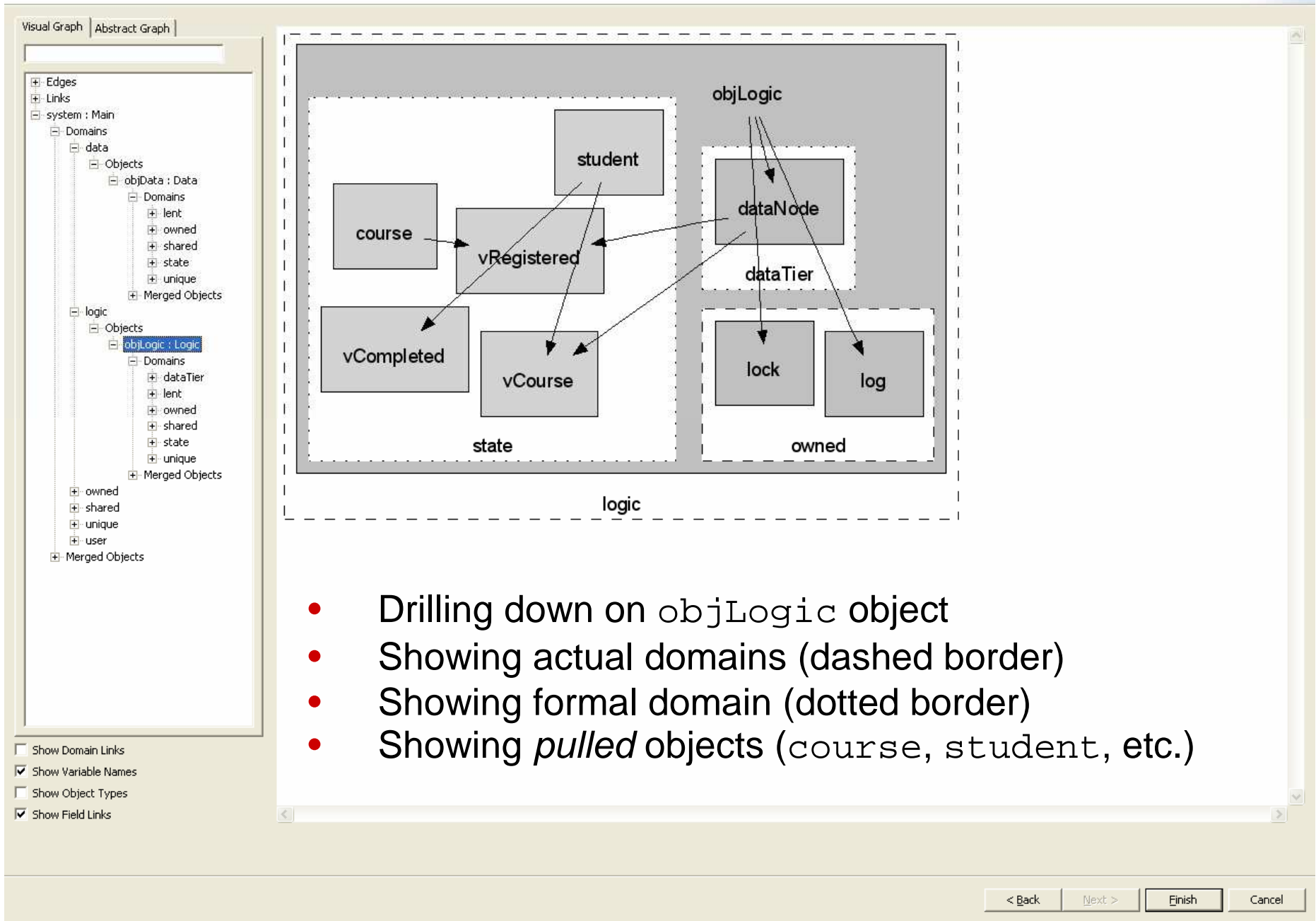
Step 3: View the Ownership Object Graph (OOG)

Select the object to visualize.



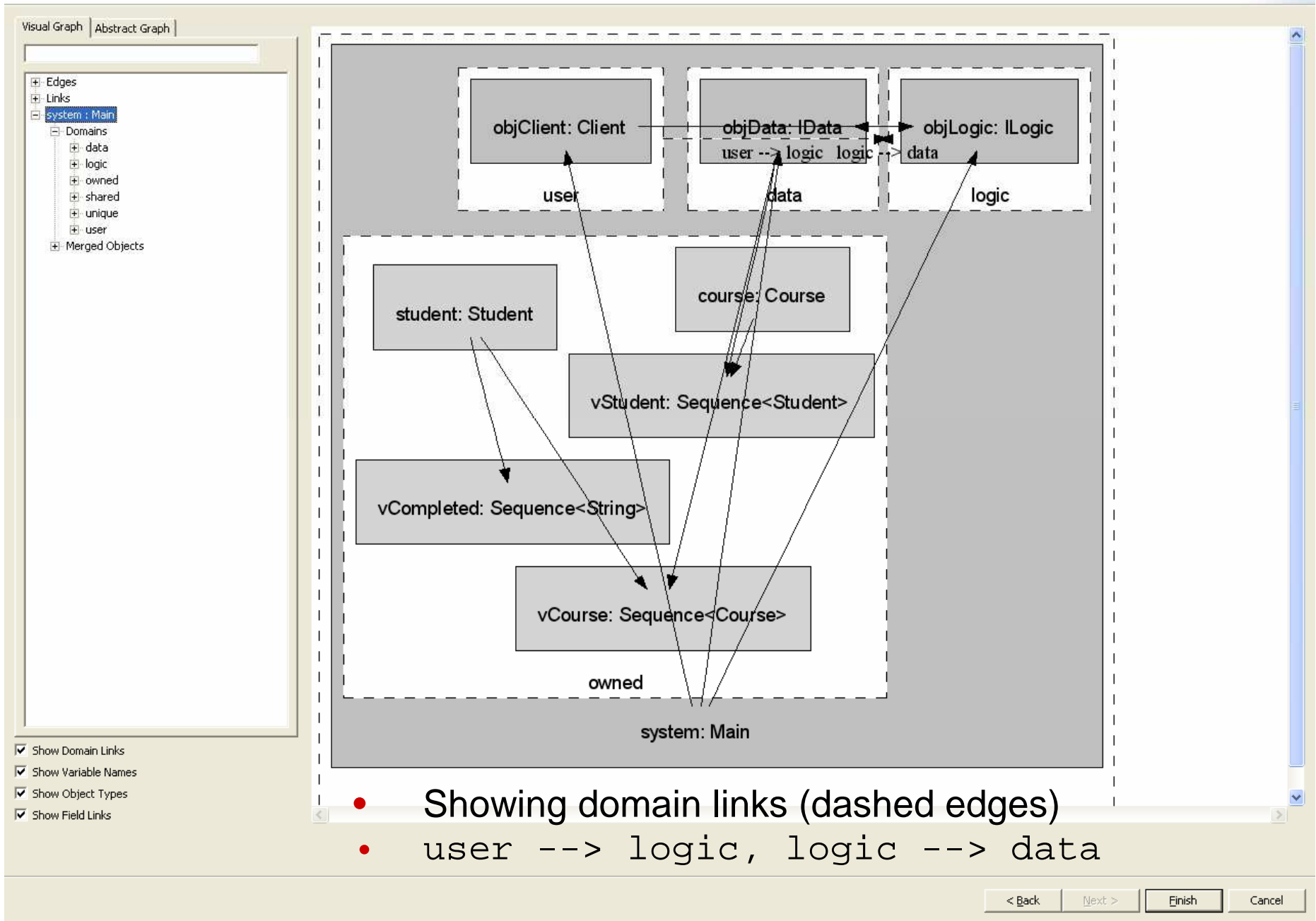
Step 3: View the Ownership Object Graph (OOG)

Select the object to visualize.



Step 3: View the Ownership Object Graph (OOG)

Select the object to visualize.



- Showing domain links (dashed edges)
- user --> logic, logic --> data

Navigating Ownership Object Graphs

- Display sub-graph
- Show/hide object internals
- Show/hide formal domains
- Show/hide domain links
- Show/hide variable names/object types
- Show/hide specific type of edges
- Show/hide selected elements

Implementation

- Eclipse Plug-in
 - Requires AliasJava annotation plug-in
 - Not all object edge types implemented
- For more information
 - Related Demonstration: “Bringing Ownership Domains to Mainstream Java”
 - <http://www.archjava.org>

Summary

- Demonstrated a static analysis for extracting instance-based hierarchical runtime views
- Adding ownership annotations to recover architectural runtime structure