# A Workbench for Template-Driven Program Transformation

Coen De Roover
cderoove@vub.ac.be

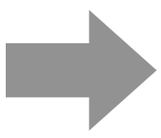
Siltvani
siltvani@gmail.com

Software Languages Lab
Vrije Universiteit Brussel
Belgium

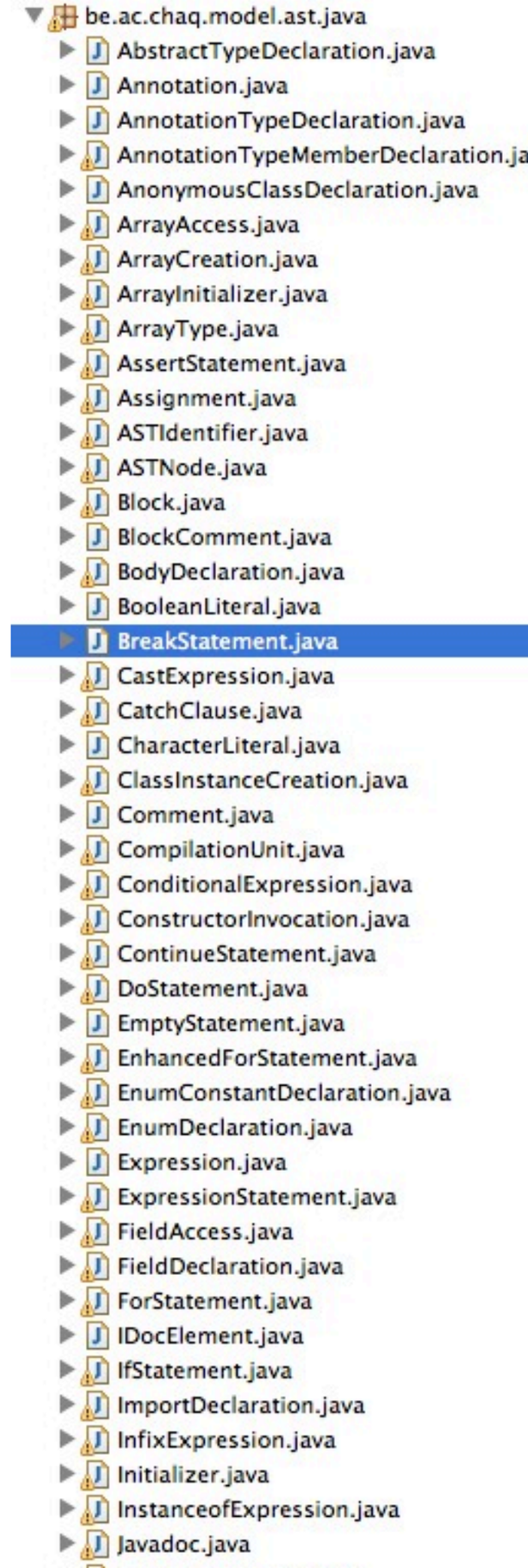
# Repeating Changes

## to similar, but non-identical code

```
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier label;

    public EntityIdentifier getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier label) {
        this.label = label;
    }
}
```



```
public class BreakStatement extends Statement {
    @EntityProperty(value = SimpleName.class)
    private EntityIdentifier<SimpleName> label;

    public EntityIdentifier<SimpleName> getLabel() {
        return label;
    }

    public void setLabel(EntityIdentifier<SimpleName> label) {
        this.label = label;
    }
}
```



# Repeating Changes

## using state-of-the-art program transformation technology

subjects of the transformation (LHS)

requires specifying

... their state afterwards or change actions (RHS)

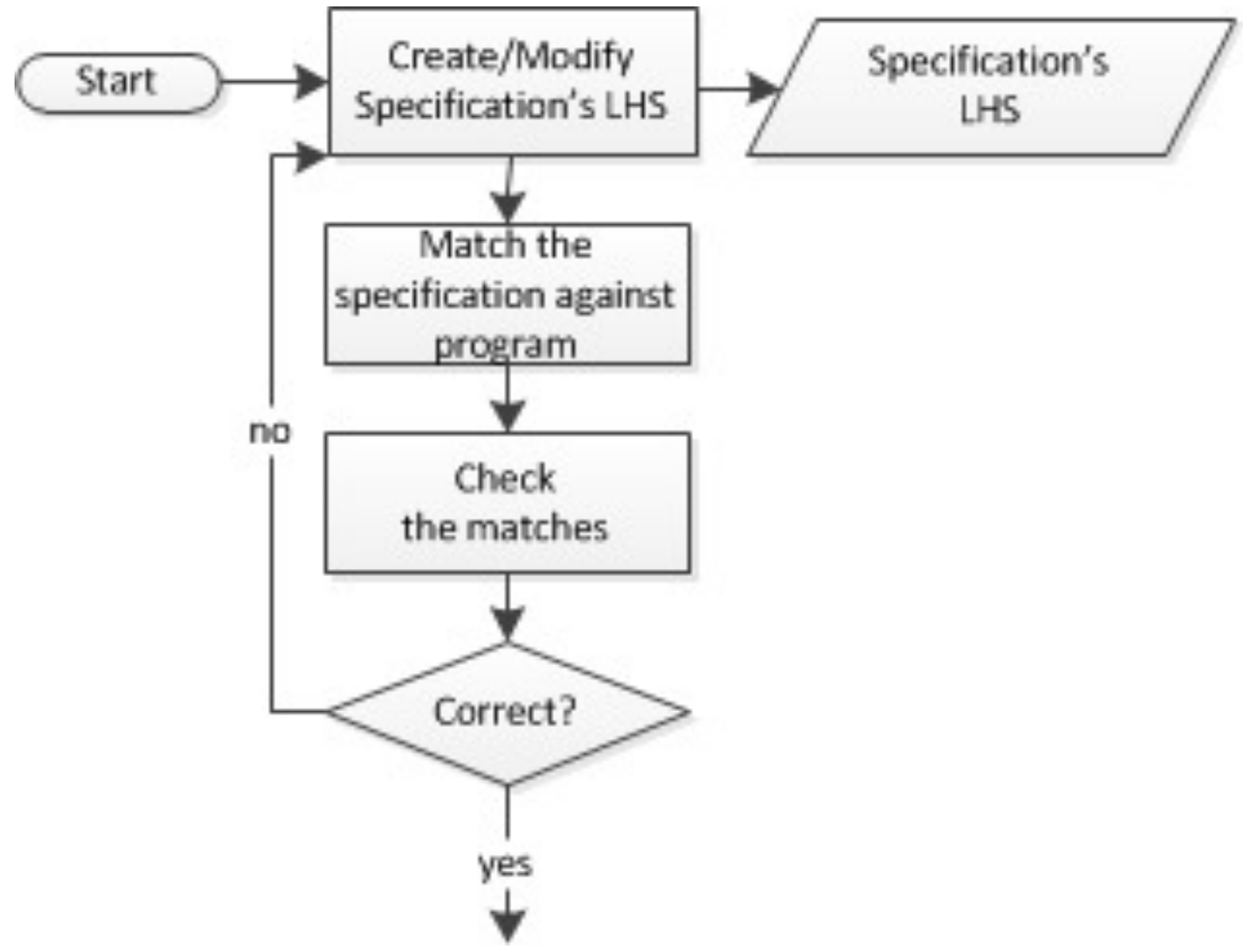
carefully ensuring

no required changes are missed

no unwarranted changes are applied

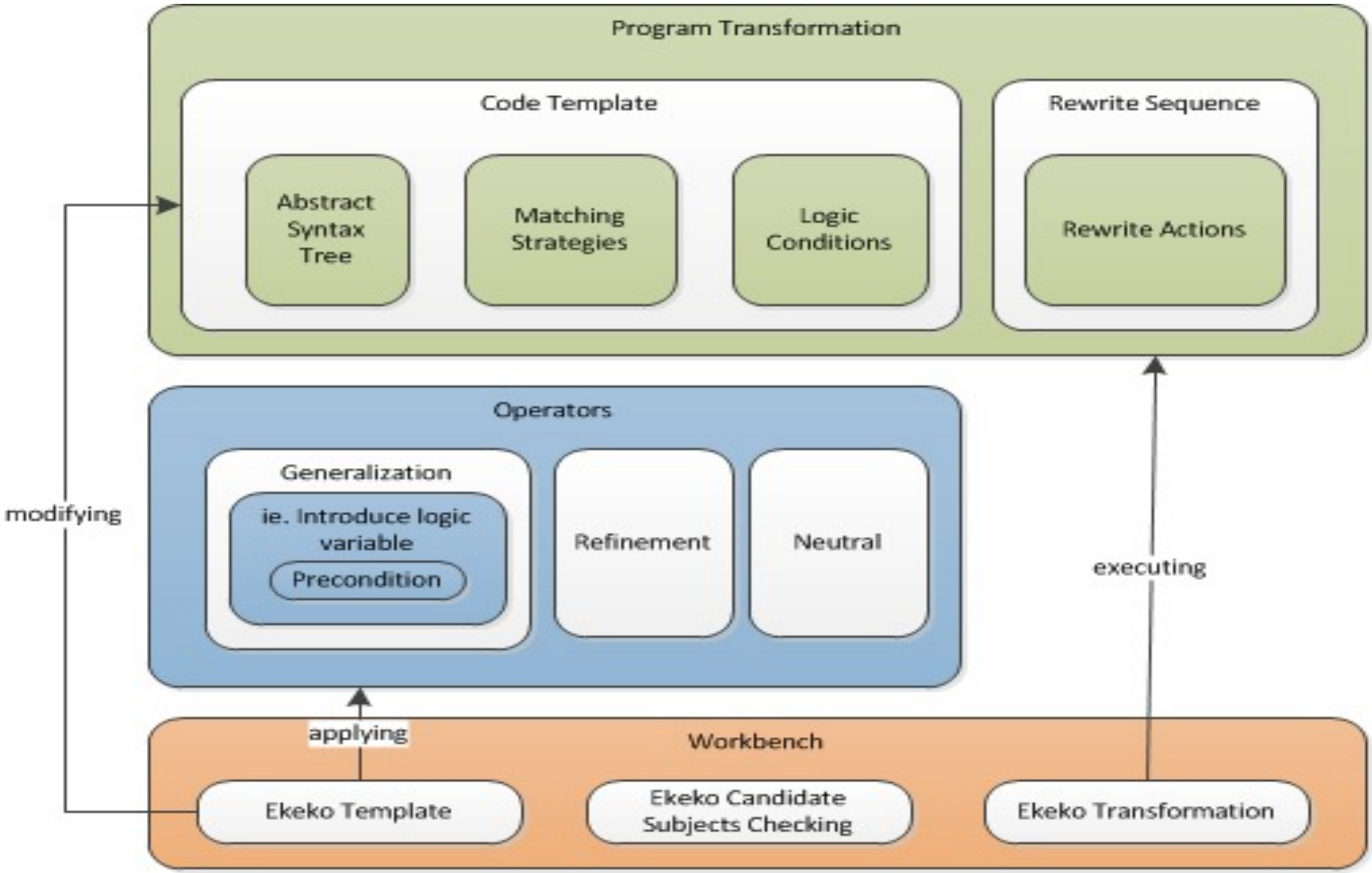
# Specifying Transformations

## iteratively and incrementally

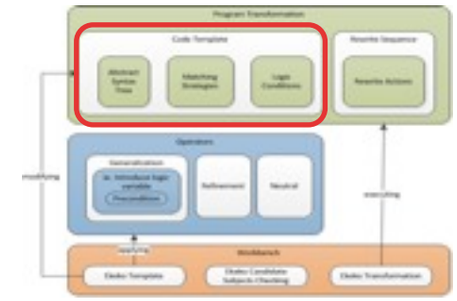



goal: provide language and tool support

# Overview of Our Approach



# Transformation Specification



## LHS: one or more code templates

concrete syntax

meta-variable

```
public void ?name(String str) {
  [if(!str.isEmpty())
    System.out.println(str);
  ]@[(relax-else)]
}
```

matching strategy

✔

```
public void printB(String str) {
  if(!str.isEmpty())
    System.out.println(str);
  else
    System.out.println(“Empty”);
}
```

?name➔printB

# Transformation Specification

## RHS: one or more rewrite actions

LHS

```
public class ?class {
  [Phone ?field;]@[(= ?target)]
}
```

RHS

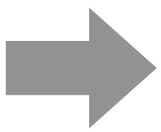
```
[List<Phone> ?field;]@[(replace ?target)]
```

replacement template

action

part of LHS match

```
public class Person {
  private Phone phone;
}
```



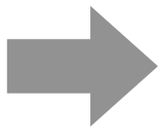
```
public class Person {
  private List<Phone> phone;
}
```

# Language Support

## operators for generalizing and refining specifications

(introduce-variable <template> <t-node> <var-name>)

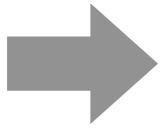
```
return age;
```



```
return ?exp;
```

(must-alias <template> <t-node> <t-node>)

```
return ?exp;
```



```
return [?exp]@[(must-alias ?field)];
```

```
void ?method(String str) [{
 int i = str.indexOf('A');
 if (i > 0)
  println(str + " : A at " + i);
}]@[(contains)]
```

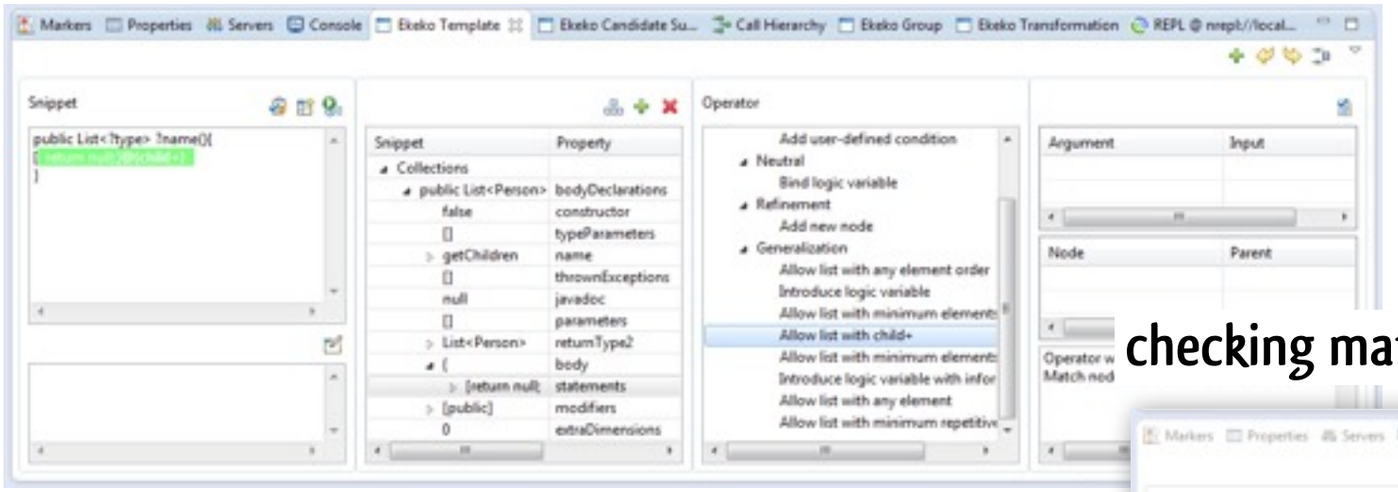
(generalize-aliases <template> <t-node>)

```
void ?method(String ?v) [{
 int i = [?v1]@[(must-alias ?v)].indexOf('A');
 if (i > 0)
  println([?v2]@[(must-alias ?v)] + " : A at " + i);
}]@[(contains)]
```

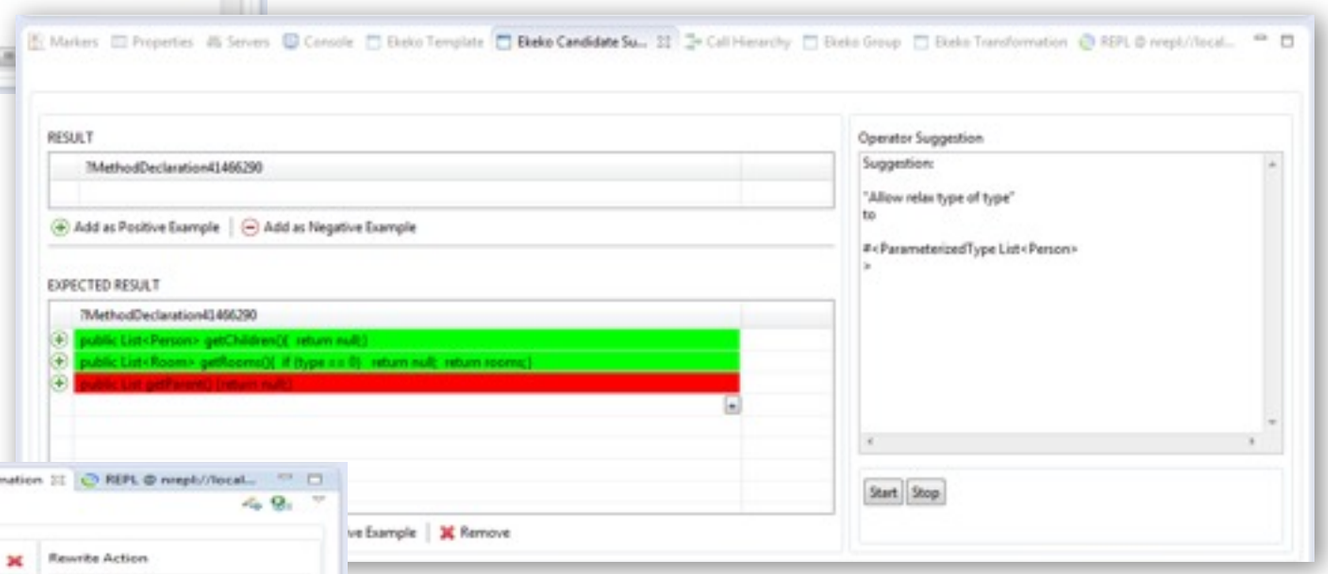
# Tool Support

## workbench of Eclipse plugins

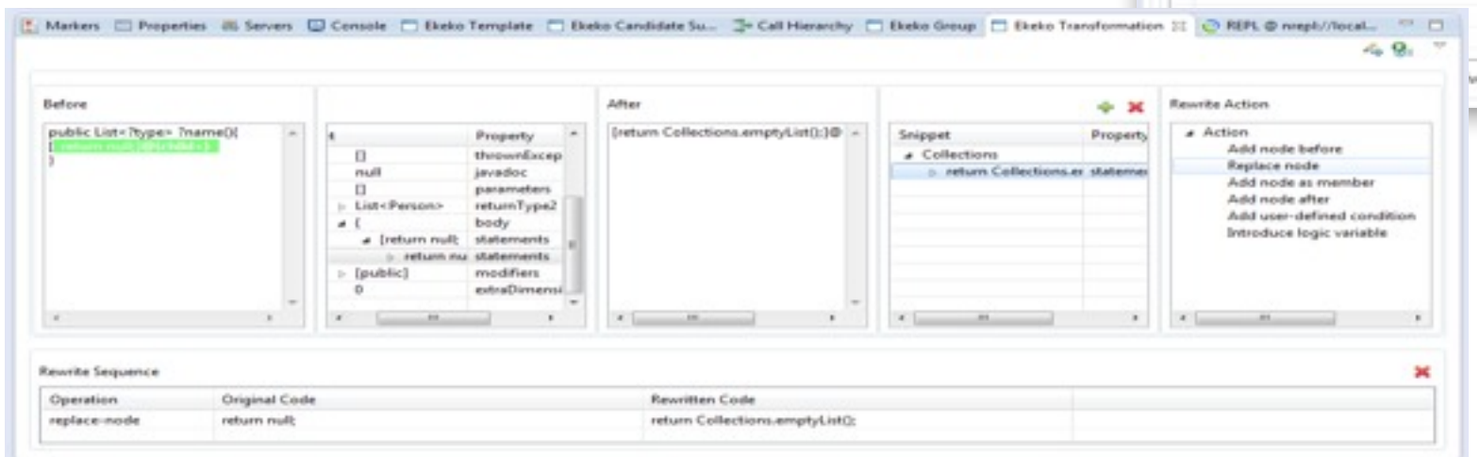
### defining LHS of specification and applying operators




### checking matches for LHS and getting operator suggestions



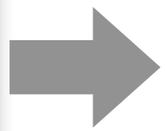

### defining RHS of specification



# Video Demo

## changing all list-valued null-returning methods

```
public List<Person> getChildren() {
  return null;
}
```



```
public List<Person> getChildren() {
  return Collections.emptyList();
}
```

✔

```
public java.util.List getParents() {
  return null;
}
```

✔

```
protected List<Room> getRooms() {
  if (type == 0)
     return null;
  return rooms;
}
```

✘

```
public List<Person> getStudents() {
  return students;
}
```

Java EE - Test/src/Demo.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Ekeko EkekoSnippets Run Window Help

Quick Access

Demo.java Demo2.java

```
List students;
List rooms;
int type;

public List<Person> getChildren() {
    return null;
}

public List getParent() {
    return null;
```

Markers Properties Servers Console Ekeko Template Call Hierarchy Ekeko Group REPL @ nrepl://localhost:54073 (damp.ekeko.snippets.operatorsrep)

Snippet

| Snippet | Property |
| --- | --- |
| Collections | |

Operator

- Operator
  - Other
    - Remove user-defined condition
    - Add user-defined condition
  - Neutral
    - Bind logic variable
  - Refinement
  - Generalization
    - Introduce logic variable
    - Introduce logic variable with information
    - Allow list with any element

| Argument | Input |
| --- | --- |

| Node | Parent |
| --- | --- |

Java EE Sign in to Google...

# A larger example: One Change Instance

## converting from a list to a keyed collection

```
public class Hospital {
  private LinkedList<Department> departments;
  private Register register;
  private GraphTree map;

  public Hospital() {
    departments = new LinkedList<Department>();
    register = new Register();
    map = new GraphTree();
  }

  public void addDepartment(Department dep) {
    departments.addFirst(dep);
  }

  public Department findDepartment(String name) {
    for (Department d : departments) {
      if (d.getName().equals(name))
         return d;
    }
    return null;
  }
}
```

```
import java.util.HashMap;

public class Hospital {
  private HashMap<String, Department> departments;
  private Register register;
  private GraphTree map;

  public Hospital() {
    departments = new HashMap<String, Department>();
    register = new Register();
    map = new GraphTree();
  }

  public void addDepartment(Department dep) {
    departments.put(dep.getName(),dep);
  }

  public Department findDepartment(String name) {
    return (Department)departmens.get(name);
  }
}
```

# A larger example: Generalized Specification

## converting from a list to a keyed collection

unparsed
(syntax is malleable)

LHS

```
public class ?class {
  [[private LinkedList<?type> ?list;]@(= ?field)

  public ?constructor(){
    [[[?list2]@(must-alias ?list)=new LinkedList<?type>();]@(= ?init)]@(contains)
  }

  [public void ?add(?type ?arg1){
    [?list3]@(must-alias ?list).addFirst([?arg3]@(must-alias ?arg1));
  }]@(= ?insert)

  [public ?type4 ?find(?argType ?arg5) {
    [[?el]@(var-type ?type).?getKey()]@(contains)
  }]@(= ?lookup)]
}
```

RHS

```
[private HashMap<?argType,?type> ?list;]@(replace-node ?field)

[?list2=new HashMap<?argType,?type>();]@(replace-node ?init)

[public void ?add(?type ?arg1){
  ?list3.put(?arg1.?getKey(),?arg1());
]@(replace-node ?insert)

[public ?type4 ?find(?argType ?arg5){
  return (?type4)?list2.get(?arg5);
}]@(replace-node ?lookup)
```

# A larger example: New Change Application

## converting from a list to a keyed collection

```
public class Department {
     private String name;
     private LinkedList<Room> rooms;
     private Room wroom;
     private SPriorityQueue waitingroom;
     private LinkedList<Device> equipment;

     public Department(String n) {
          name = n;
          rooms = new LinkedList<Room>();
          wroom = new Room(0);
          waitingroom = new SPriorityQueue();
          equipment = new LinkedList<Device>();
     }

     public void addRoom(Room room) {
          rooms.addFirst(room);
     }

     public void removeRoom(Room room) {
          rooms.remove(room);
     }

     public Room findRoom(Number id) {
          for (int i = 0; i < getNoOfRooms(); i++) {
               Room room = (Room) rooms.get(i);
               if (room.getId() == id) return room;
          }
          return null;
     }
}
```

```
public class Department {
     private String name;
     private HashMap<Number, Room> rooms;
     private Room wroom;
     private SPriorityQueue waitingroom;
     private LinkedList<Device> equipment;

     public Department(String n) {
          name = n;
          rooms = new HashMap<Number, Room>();
          wroom = new Room(0);
          waitingroom = new SPriorityQueue();
          equipment = new LinkedList<Device>();
     }

     public void addRoom(Room room) {
          rooms.put(room.getId(), room);
     }
     public void removeRoom(Room room) {
          rooms.remove(room.getId());
     }

     public Room findRoom(Number id) {
          return (Room) rooms.get(id);
     }
}
```

# Conclusions

template-driven program transformation language

language support for generalizing and refining transformation specifications

- beyond textual editing

tool support in the form of a workbench

- operator application suggestions

next steps

- assess usability
- state space exploration
- tackle API migration cases