# A Refactoring Discovering Tool based on Graph Transformation

*Javier Pérez, Yania Crespo*
*GIRO research group*
*Departamento de Informática*
*Universidad de Valladolid*
*{jperez,yania}@infor.uva.es*
*www.giro.infor.uva.es*

Our tool prototype implements a method, based on graph transformation to discover refactoring sequences between two versions of a software system. In case a refactoring sequence exists, our tool can also help reveal the functional equivalence between the two versions of the system, at least, as far as refactorings can assure behaviour preservation.

## Searching Refactorings

```
//-----------------------PrinterHub.java
package printing;
public class PrinterHub {
  public void printWithPDF(PDFPrinter p){
    p.printPDF();
  }
}
//-----------------------PDFPrinter.java
package printing;
public class PDFPrinter extends Printer{
  public void printPDF(){
  }
}
//--------------------------Printer.java
package printing;
public class Printer {
        public String content;
        public void setContent(String c){
                this.content = c;
        }
        public void print(){
        }
}
```

source system

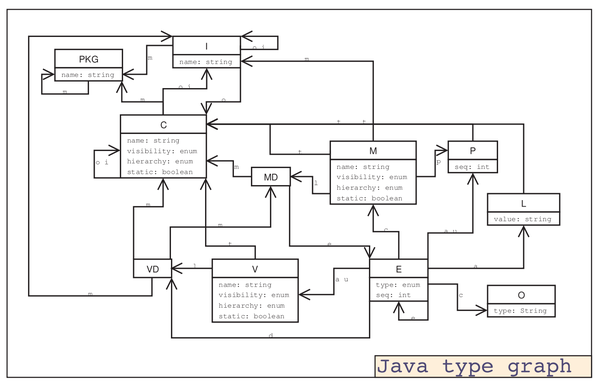
*Does a refactoring sequence exist between two versions of a system*

?

```
//-----------------------PrinterHub.java
package printing;
public class printerHub {
  public void print(Printer p){
    p.print();
  }
}
//--------------------------Printer.java
package printing;
public class Printer {
        public String content;
        public void setContent(String c){
                this.content = c;
        }
        public void print(){
        }
}
```

target system

## A Graph Transformation Approach



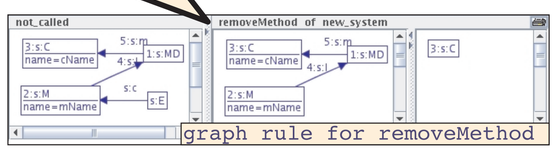

Java type graph

Java surce code — target system

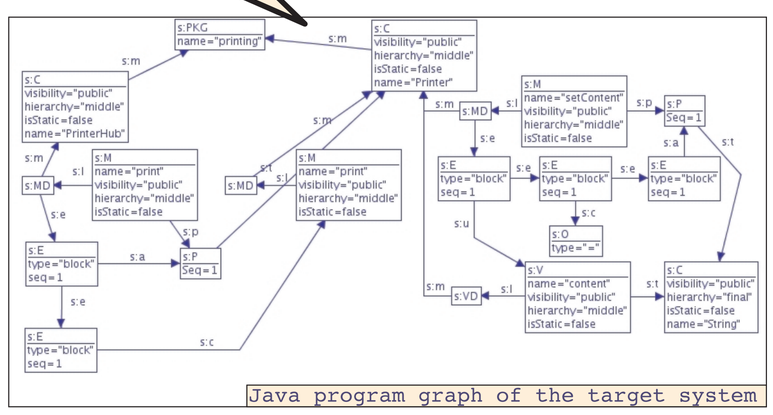
Based on the refactoring formalization works by Tom Mens, we have defined a Java metamodel (type graph) to represent:

- Java programas as **Java program graphs**
- refactorings as **graph transformation rules**

refactoring definition — removeMethod




graph rule for removeMethod



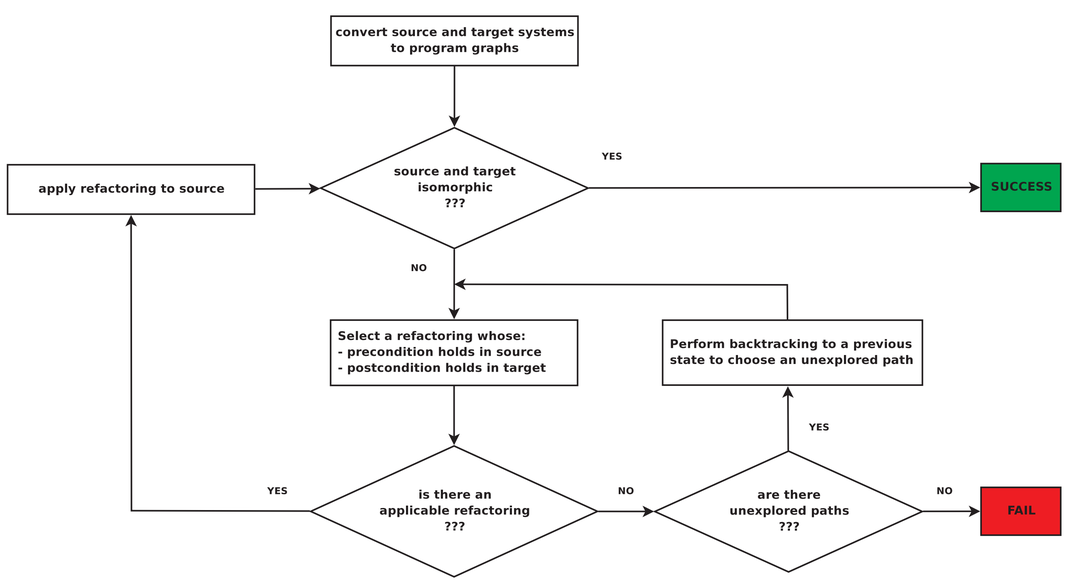
Java program graph of the target system

## Modeling the Problem

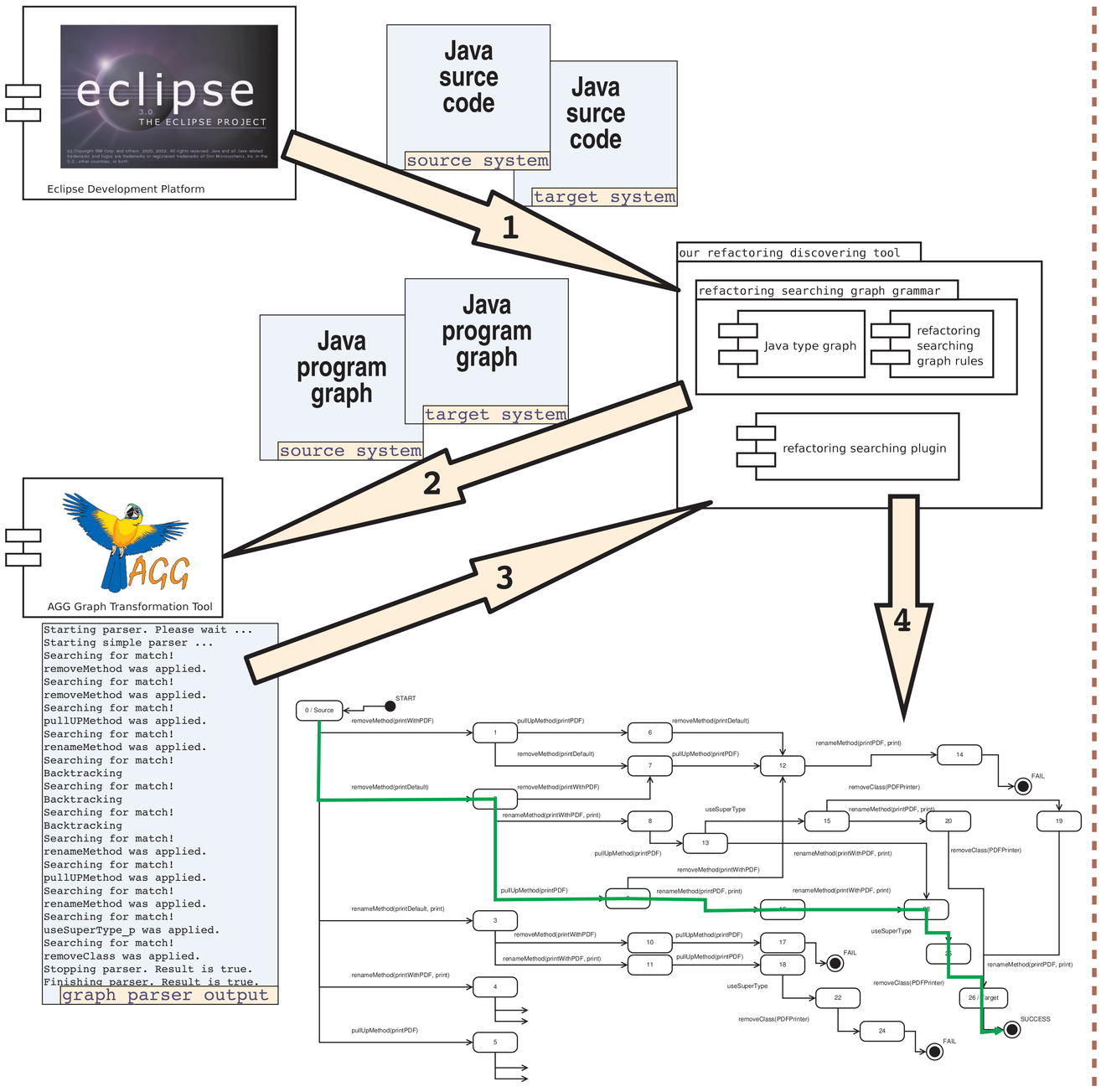
we model the sequence search as a state space search problem

| | | |
|---|---|---|
| source system | start state | START → 0 / Source |
| refactoring operations | state transitions | 1 —pullUpMethod(printPDF)→ 6 |
| target system | goal state | 26 / Target → SUCCESS |
| sequence existence? | goal reachability? | START → 0 / Source → ? → 26 / Target → SUCCESS |
| refactoring sequence | start-to-goal-path | START → 0 / Source - - 26 / Target → SUCCESS |

and we apply a depth-first search algorithm




## Our Tool as an Eclipse Plugin




## Results

- Approach validated with a toy example

- Search rules implemented so far:
  - pullUpMethod
  - renameMethod
  - useSuperType
  - removeMethod
  - removeClass

- Approach support renamings and multiple refactorings applied over the same element

- The current implementation is a proof of concept that offers very promising results

## Future Work

- Improve visualization of results: to automatically generate the state space graph

- Extend the set of supported refactorings, implementing more refactoring searching rules

- Measure scalability over industrial-size systems

- Test other graph transformation tools as PROGRES

## Open Issues

- Our algorithm is only partially correct, thus it could not terminate in certain cases.
*Would we be able to assure a finite state space?*

- It is not possible to represent every refactoring with AGG.
*Can we find a more suitable graph transformation tool?*

- We address documenting software evolution and revealing functional equivalence between two system versions.
*To which real scenarios can our results be applied?*