Christian Colombo
(joint course design with Gordon J. Pace)
University of Malta

March 2018

RUNTIME VERIFICATION FROM THEORY TO PRACTICE AND BACK

A Guarded Command Language for RV

A Guarded Command Language

- We will now move on to a higher-level of abstraction, using a *guarded-command language* (GCL) to specify the properties.
- You are given a parser and code to manipulate GCL specifications, and…
- …your task is to:
  - write a translator from GCL to AspectJ
  - rewrite the specification of FiTS using GCL

S&S of the GCL

- A GCL script consists of three parts:
  - A part starting with a line containing just `VERIFICATIONCODE in` which Java code (typically in our case a class called `Verification`) to support the specification may be included.
  - A second part starting with a line just containing the word `PRELUDE` –consisting of a number of rules which will be added verbatim to the AspectJ code to be generated. This typically contains package name and imports.
  - A third part starting with a line containing just the word `RULES`, and in which are the specification rules.
- The script may include comments starting with //.

S&S of the GCL

- At their simplest level, specifications using GCL are of the form:

```
method(parameters) | condition -> { action }
```

- For example:

```
Shape.move(Integer dx, Integer dy)
   |  dx>=0 && dy >=0
   -> { System.out.println(“NE direction”); }
```

S&S of the GCL

- At their si
  using GC

```
    method(param        condition -> { action }
```

Before this method is called…

- For exampl :

```
Shape.move(Integer dx, Integer dy)
    |  dx>=0 && dy >=0
    -> { System.out.println(“NE direction”); }
```

S&S of the GCL

- At their simpl
  using GCL ar

```
    method(parameters)          tion -> { action }
```

…matching these parameter variables…

- For example:

```
  Shape.move(Integer dx, Integer dy)
     |  dx>=0 && dy >=0
     -> { System.out.println(“NE direction”); }
```

S&S of the GCL

- At th … tions usin

```
me … { action }
```

- For exam

```
Shape.move(Integer dx, Integer dy)
   |  dx>=0 && dy >=0
   -> { System.out.println(“NE direction”); }
```

… and if this condition holds (which may refer to the parameters)…

S&S of the GCL

- At their s … s using GC

```
    method(param   condition -> { action }
```

… then run this action (which may refer to the parameters)…

- For example:

```
  Shape.move(Integer dx  Integer dy)
     |  dx>=0 && dy >=0
     -> { System.out.println(“NE direction”); }
```

S&S of the GCL

- Multiple rules can be given in a single script:
- For example:

```
Shape.move(Integer dx, Integer dy)
   |  dx>=0 && dy >=0
   -> { System.out.println(“NE direction”); }
*.move(Integer dx, Integer dy)
   |  dx<=0 && dy >=0
   -> { System.out.println(“NW direction”); }
```

S&S of the GCL

- To capture the target of the method, you may follow the event name and parameters by the keyword `target` and give it a name and type.
- Example:

```
Shape.moveUp(Integer dy) target (Shape s)
       |  dy >=0
       -> {
             s.mark(“MOVED”);

       }
```

Syntactic Sugar in the GCL

- You may write (..) instead of specifying the parameters of an event (if you do not care about capturing them).
- You may leave the condition empty, in which case it is interpreted to be *true*. Note that the | is still necessary.
- Example:

```
Shape.move(..) target (Shape s)
      | -> { s.flip(); }
```

GCL Support

- **Rule.java**: A class with methods to enable the manipulation of a GCL rule. The important methods are:

```
ArrayList<String> getParameterVariables()
ArrayList<String> getParameterTypes()
String getTargetType()
String getTargetVariable()
String getEvent()
String getCondition()
String getAction()
```

GCL Support

- **Rule.java**: A class wi
  enable the manipulat
  The important methods

```
ArrayList<String> getParameterVariables()
ArrayList<String> getParameterTypes()
String getTargetType()
String getTargetVariable()
String getEvent()
String getCondition()
String getAction()
```

The first four may be set to *null* if not given in the rule

GCL Support

- **GCLScript.java**: A class with methods to enable the manipulation of a GCL script.
- The important constructor parses a GCL script from a file possibly throwing an exception:

  ```
  GCLScript (String filename)
  ```

- The important getter methods are:

  ```
  ArrayList<Rule> getRules()
  String getAuxiliaryCode()
  String getPrelude()
  ```

GCL Support

- **Main.java**: A simple class which reads a script and pretty prints it again to the standard output.

Exercises

1. Add a method `toAspectJ` to the classes `Rule` and `GCLScript`, which return AspectJ source code (as a string) implementing the code.
2. Rewrite the specification of FiTS using GCL.
3. Run your translator on the specification to obtain an AspectJ script which you can compile with FiTS.
4. Rerun the scenarios to check that you get the same results as before.

Example – Source

```
VERIFICATIONCODE
public class Verification {
         public static Boolean a_happened;
         public static void fail(String s)
         {        System.out.println(“ERROR: ”+s);
         }
         public static void initialiseVerification()
         {        a_happened = false;
         }}

PRELUDE
package fits;

RULES
*.a(..) target (UserInfo user)
  |  !(user.equals(SUPERUSER))
  -> { Verification.a_happened = true; }
*.b(Integer cost) target (UserInfo user)
  |  (!(user.equals(SUPERUSER)) && (cost > 1000) &&
(Verification.a_happened)
  -> { Verification.fail(”Property 1 violated"); }
```

Example – Verification.java

```
public class Verification {
       public static Boolean a_happened;
       public static void fail(String s)
       {
              System.out.println(“ERROR: ”+s);
       }
       public static void initialiseVerification()
       {
              a_happened = false;
       }
}
```

Example – Properties.aj

```
package fits;

public aspect Properties {

  before (UserInfo user):
    call(* *.a(..)) &&
    target(user) {
    if (!(user.equals(SUPERUSER)))
      { Verification.a_happened = true; }

  before (UserInfo user, Integer cost):
    call (* *.b(..)) &&
    target (user) &&
    args(cost) {
    if ((!(user.equals(SUPERUSER)) &&
        (cost > 1000) &&
        (Verification.a_happened))
       { Verification.fail(”Property 1 violated"); }
}
```

Eclipse and the GCL Tool

For the sake of this assignment:

1. Import the `MyRVTool` project.
2. Import `FinancialTransactions-03-GCL` – a clean FiTS project, with:
   a. A text file `specification.rules`
   b. A Java class `Verification.java`
   c. An AspectJ file `Properties.aj`
3. Right-clicking and choosing `Properties` on these files will give you their filename with the full path. Copy and paste them into the `Main.java` file in the `MyRVTool` project.
4. The workflow is to: (i) edit the the `MyRVTool` specification; (ii) running the `MyRVTool` project to generate the code in the `FinancialTransactions-03-GCL` project, which (assuming correct syntax is generated); (iii) compile and run the `FinancialTransactions-03-GCL` project.