Introducing Meta-Interfaces into Java

Peep Küngas, Vahur Kotkas, and Enn Tyugu

Software Department
Institute of Cybernetics at Tallinn Technical University, Estonia
{peep,vahur,tyugu}@cs.ioc.ee

In the present work we apply a formal program construction method with the aim of increasing the flexibility of Java classes. This work relates to recent results in the dynamic composition of software and component-based program development.

A meta-interface is a specification that introduces a collection of interface variables of a class and defines, which interface variables are computable from others under what conditions.

For instance, having a class $Triangle$ with methods $sinFindSide$, $sinFindAngle$ for computing based on theorem of sine: $\frac{a}{sin(A)} = \frac{b}{sin(B)} = \frac{c}{sin(C)}$, we can introduce interface variables for all angles (A, B, C) and sides (a, b, c) of the triangle and declare a meta-interface that will specify the methods that can be used. Such meta-interface could look as follows:

```
var a, b, c, A, B, C : any
rel a,A,B -> b {sinFindSide}
...
rel b,B,a -> A {sinFindAngle}
```

Here $sinFindSide$ and $sinFindAngle$ are implementations of the sine theorem in Java methods. The meta-interface just declares how these methods can be used.

After introducing an extension of the language that allows one to use equations, we can specify this meta-interface by using a shorter description and do not need the methods implemented by programmers:

```
var a,b,c, A,B,C : any
rel a/sin(A)=b/sin(B)
rel a/sin(A)=c/sin(C)
rel c/sin(C)=b/sin(B)
```

Note that the components specified in a meta-interface as of type **any** have to be actual components of the Java class and their type is determined by the Java declarations.

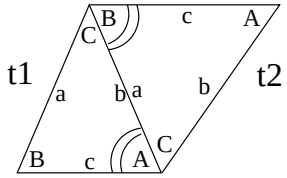
The meta-interface is used as follows: one writes a request for synthesis of a method with input $x_1, \ldots, x_m$, where $m > 0$, and output y, whereas $x_1, \ldots, x_m, y$ are variables specified in the meta-interface, for instance, in the form $x_1 \& \ldots \& x_m \rightarrow y$, and uses a prover to prove that this formula is derivable from the specification of the meta-interface.

The prover returns a sequence of rules applied in the proof, which from the synthesis point of view represents an algorithm we use to generate the program

code to solve the problem. Thus the algorithm is the co-result (or side-effect) of the proving process.

A meta-interface can be written for two different purposes. First, it may specify possible usage of the class, i.e. its derivable methods, like in our example. Second, it can be used as a specification showing how some application software should be composed from components supplied with meta-interfaces. In the latter case, a new class can be built completely from a specification of its meta-interface. The aim of introducing meta-interfaces is to make classes as components more flexible.

To illustrate the usage of meta-interfaces let us specify a class for solving a computational problem on two triangles that have one common side and one common angle (see following figure).



Values of some components of triangles are given, as can be seen from the following class code.

```
class Problem implements metaInterface {
    public static String[] metaInterface = {
        "var t1, t2 : any",
        "rel t1.b==t2.a",
        "rel t1.A==t2.B"};
    Triangle t1 = new Triangle(), t2 = new Triangle();
    public void main(String[] args) {
        Problem p = new Problem();
        p.t1.b = 5;
        p.t1.B = 70;
        p.t2.A = 40;
        String name = SSP.synthesize(p, "t1.a -> t2.b");
        SSP.exec(name, p, new Integer(4));
    }
}
```

As a result of the call $SSP.synthesize$ a new method will be synthesized (see method $xf17634$ below) that realizes the requested computational problem.

The method $SSP.exec$ executes the synthesized method and as a result modifies the object p by assigning proper value to the component $t2.b$.

```
    public void xf17634(Problem p, int i) {
        p.t1.a = i;
        p.t1.A = p.t1.sinFindAngle(p.t1.b, p.t1.B, p.t1.a);
        p.t2.a = p.t1.b;
        p.t2.B = p.t1.A;
        p.t2.b = p.t2.sinFindSide(p.t2.a, p.t2.A, p.t2.B);
    }
```