Behaviour Editing Environment: a simple agent-oriented DSL

Manuel Bonarrigo
manuel.bonarrigo@studio.unibo.it

Lorenzo Rizzato
lorenzo.rizzato@studio.unibo.it

January 2019

The agent-oriented framework which we implemented and employed towards the end of the Distributed Systems course has proven to be able to handle the creation, execution and communication between agents effectively. Still, its verbosity and complexity might discourage its usage for the implementation of concurrent and reactive systems of varying scale. The main goal of this project is to develop a Domain Specific Language which simplifies the definition of agents and behaviours, in order to let the users focus on the problem they are trying to solve, rather than on the underlying infrastructure.

1 Goal/Requirements

The DSL to be developed should provide a clean and concise way to define and instantiate a series of *agents*, whose execution should be carried out by the AOP framework developed during the course. Therefore, the compiler should be able to generate a series of (possibly) human-readable Java classes – one for each agent – written by means of the `Agent` and `Behaviour` abstractions introduced by the framework.

Consider this example:

```
agent Counter {
    int x = 0;

    bee inc {
        x++;
    }

    plan inc;
}
```

```
public class Counter extends Agent {
    private int x = 0;

    private Behaviour inc() {
        return Behaviours.of(() -> x++);
    }

    @Override
    public void onBegin() {
        inc().addToAgent(this);
    }
}
```

We would expect the compiler to take the agent specification on the left as input, and produce the Java class on the right.

Agents should also be able to express more complicated execution plans. For instance, one might want to write an agent that executes two behaviours in parallel, while waiting for both to finish:

```
agent Counter {
    int x = 0;

    bee inc {
        x++;
    }

    bee dec {
        x--;
    }

    plan inc and dec;
}
```

```
public class Counter extends Agent {
    private int x = 0;

    private Behaviour inc() {
        return Behaviours.of(() -> x++);
    }

    private Behaviour dec() {
        return Behaviours.of(() -> x--);
    }

    @Override
    public void onBegin() {
        Behaviours.allOf(
            inc(),
            dec()
        ).addToAgent(this);
    }
}
```

In general, the language should provide at least the following features:

- behaviour declaration and definition;
- nested behaviour invocation;
- the ability for an agent to specify its starting behaviour (or a combination of many of them);

- Java-like expression definition;
- generation of readable Java classes.

2 Work plan

1. Learning the *modus-operandi* of the XText framework and the XTend language;
2. Extrapolation of the redundant aspects from the AOP framework to be factorized by the language;
3. Definition of the DSL's grammar and compiler implementation through the following incremental steps:
   a) base language features (agent definition, behaviour definition, control flow statements, behaviour composition);
   b) send-receive agent interaction;
   c) LINDA coordination primitives;
   d) white- and yellow-pages discovery services;
   e) dynamic scoping.
4. Refactoring of the AOP framework in order to add support to the `IfElseBehaviour` and `DoTimesBehaviour`;
5. Creation of a suite of automatic tests for the following aspects:
   a) parsing;
   b) validation;
   c) code generation;
   d) formatting (optional).
6. *Optional*: deploying an Eclipse plugin to aid the definition, execution and deployment of programs written in the new language.

2.1 Division of labour

→ Rizzato: 1, 2, 3, 4

→ Bonarrigo: 1, 2, 3, 5

Item 6 will be potentially addressed by both participants.