The case for an object variable in rules

Joop Ringelberg 20-01-20 Version: 1

Introduction

The right hand side of a bot rule consists of a let* expression or of a series of assignments. The rule condition, the expressions in the bindings and those in the assignments of the let* are interpreted relative to the *current context*. For example, in this expression:

```
if SomeRole >> SomeProperty then
     bind AnotherRole to SomeRole
```

all the role identifiers are evaluated with respect to the context that the rule is defined in.

If one of the assignments sets a property value however, we assume the property is on the *current object:*

```
bot: for User
     perspective on: Role
        if AnotherRole >> AnotherProperty then
           SomeProperty = true
```

Here, `SomeProperty` is supposed to be carried by `Role` and, indeed, the system warns if this is not the case. `Role` is said to be the *object of the perspective*.

Problem statement

What if we were to use a Calculated Role as the object of the perspective? That introduces no problem. The value of the current object set is calculated as declared with that Calculated Role.

But next consider that the computation depends on state that we actually change in the rule's right hand side! We might, for example, change a Boolan property's value, causing the object set to become empty. Obviously, the mechanism responsible for executing the updates must capture the current object set as it exists at the start of the execution.

While this can be handled internally (because property assignment uses the current object set *implicitly*), the following case fails:

```
FilteredRole = filter Role with Criterium
bot: for User
     perspective on: filteredRole
        if true then
        Criterium = false
        bind FilteredRole to AnotherRole
```

Suppose an instance of `Role`, `Role_1`, has value `true` for its property `Criterium`. As soon as the first assignment of the rule’s effect has been carried out, FilteredRole will evaluate to a set that does no longer hold `Role_1`! `Role_1` will never be bound to `AnotherRole` by this rule.

Solution

With version v.0.2.0 we introduce a variable in rules that is automatically bound to the members of the current object set, *as it exists when the rule is triggered*. The variable is called `object` (notice that variables are lower case names). So, the above rule can be corrected as follows:

```
FilteredRole = filter Role with Criterium
bot: for User
    perspective on: filteredRole
      if true then
        Criterium = false
        bind object to AnotherRole
```

Similar cases

A similar argument can be made for the role instance that the query expression in a Calculated Property is applied to. We can imagine a variable `role` to be bound to that instance. However, use cases do not come easily to mind. There may be situations where we want to filter a value with reference to (properties of) the original role instance.

By analogy, we could argue for a variable `context` for Calculated Roles. However, while that variable would be convenient, it would not enable anything we cannot yet express, as explained with the following example:

```
MyCalculatedRole = let* context <- (SomeRole >> context) in …
```

As any role instance has just a single context, we can easily compute it and bind it in a `let*`.