Optimizing inverted query application

Joop Ringelberg 23-06-20, 31-01-22 Version: 2

Introduction

In this text we describe an optimization of the functionality that applies inverted queries. We apply inverted queries for two reasons:

1. To find resources whose state must be evaluated after a modification to the represented state;
2. To find users who must receive a Transaction with the said modification.

The optimization we describe does not change the functionality.

Problem statement

Referring to Figure 1, we see that `onRoleDelta_binder` of `r2` holds two inverted queries. As the first step of the inverted query is skipped because of the cardinality of the binder operation, those two are:

```
binder r5 >> binding context                    I
context                                         II
```

Now consider a `RoleDelta` with binder (id) equal to (an instance of) `r1` and binding equal to (an instance of) `r2`. It is obvious that only inverted query II should be applied (the new path will never lead to `c4`, as it must include the new binding between `r1` and `r2`).

The query evaluation mechanism currently is implemented in such a way that applying II to `r1` will give no results, so the semantics is preserved. We can safely skip this step, however.

Solution

We will solve the problem by storing queries in `onRoleDelta_binder` not as an Array, but as a map indexed by the type of the range of the original binder step (the step that is actually removed from the inverted query before it is stored in `onRoleDelta_binder`).

Runtime, we then use the type of the binder in the delta to index the queries in `onRoleDelta_binder` so we only apply the right inverted queries.

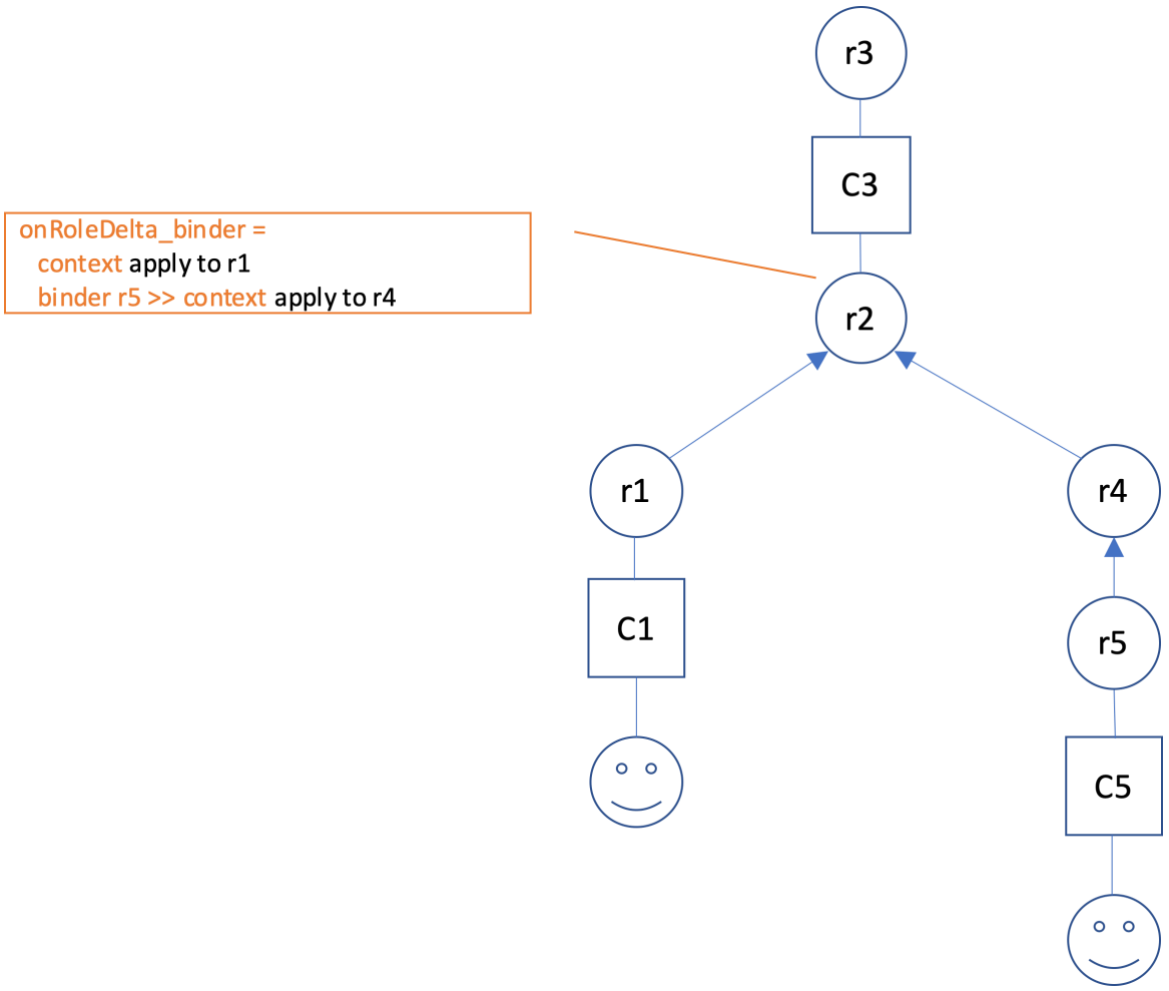

Figure 1. Two inverted queries stored with onRoleDelta_binder in r2.

Multiple types (Role Aspects)

In general, r4 may have multiple types[1]. We should index the collection of inverted queries stored with (type) r2 with all types of (instance) r4.

[1] We jump somewhat opportunistically from instance to type with respect to the roles in this example. Now we mean to understand c4 as an instance, having multiple types.