Naamgeving query-operatoren op instantie- en definitie niveau

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 23-05-18 | Versie: 1 |

# Introductie

Om een type (Context, Rol, Property, enz.) te kunnen beschrijven, hebben we een Context nodig[[1]](#footnote-1). Dat betekent dat “een Context” betrekking kan hebben op een *type*, maar ook op een *instantie van een type*. Hetzelfde geldt voor Rol, Property, enzovoort.

Een Query-operator is, in termen van programmeren, een *selector* die uit een bepaalde datastructuur een onderdeel neemt en als functioneel resultaat oplevert.

Perspectives heeft operatoren als rolInContext, binnenRol, buitenRol en context. binnenRol bijvoorbeeld, selecteert uit de datastructuur Context. Maar ‘Context’ kunnen we dus opvatten als *type* èn als *instantie*. In het eerste geval willen we dat binnenRol de *definitie van de BinnenRol* oplevert, in het tweede geval willen we de (instantie van) de BinnenRol van de Context(instantie).

Het instantieniveau is meer elementair dan het typeniveau. Oftewel, de implementatie van binnenRol op instantieniveau is een Purescript functie, terwijl de implementatie van binnenRol op typeniveau een samengestelde Query is[[2]](#footnote-2).

De modelleur werkt uitsluitend op typeniveau. In termen van Aangifte: hij werkt met het type Aangifte, het type Aangever, enzovoort. Maar als hij een berekende Rol opschrijft – d.w.z. een Query – beschrijft hij een pad op instantieniveau! Immers, de Query wordt runtime uitgerekend op instanties van Aangiftes, Aangevers, enzovoort.

Daar verbind ik de conclusie aan dat de primaire betekenis die we aan de standaard Query operatoren moeten geven, op instantieniveau ligt. Oftewel, als de modelleur opschrijft dat een berekende Rol de context is van de binding van de Aangever, dan interpreteren we de operator context als de context van een bepaalde binding van een bepaalde Aangever.

Onvermijdelijk moeten we dus voor de interpretatie van context, binnenRol, enz. op typeniveau àndere namen kiezen. Ik kies ervoor om de instantie-niveau namen het postfix Def te geven. Dus: contextDef, binnenRolDef, enz.

## Enkel- en meervoud

Een Rol of Property kan zowel enkel- als meervoudig zijn. In andere woorden: het kan een functie zijn, of een relatie. We kiezen ervoor om de naam van Query-operatoren daarmee in overeenstemming te laten zijn[[3]](#footnote-3). Dus: rollenInContext in plaats van rolInContext.

## Vuistregel voor Def

Het is niet altijd intuïtief meteen helder of een operator op type- danwel instantieniveau bedoeld is. Een simpele vuistregel: als de implementatie een Query is waarvan de laatste twee stappen binding >-> context zijn, moet de naam op Def eindigen!

## Closure

Een query die wordt samengesteld met de closure operator, geven we een postfix Closure. Bijvoorbeeld: een Rol (type niveau) heeft één of meerdere Aspecten. De Rol waarmee we een Rol een Aspect geven, heet aspectRol. De Query selector die dergelijke Aspecten selecteert, heet dus aspectRolDef. In het algemeen zijn we geïnteresseerd in de transitieve closure van die selector. De query die dat uitrekent, noemen we aspectRolDefClosure.

## Boolean Queries

Een Query-operator die een Boolean resultaat levert, kunnen we beginnen met is. Bijvoorbeeld: isFunctioneel. Maar we kunnen ook beginnen met has: hasLabel.

## M is for Memorizing

TripleGetters memoriseren hun functionele resultaat. ObjectsGetters doen dat niet. Een TripleGetter-naam eindigt daarom op een ‘M’.

# Speciale gevallen

We kunnen niet altijd de boven beschreven regels aanhouden.

## rolType en contextType

We hebben voor de twee datastructuren, PerspectRol en PerspectContext, ook twee selectoren nodig om de type component op te leveren. Overloading kan niet.

## iedereRolInContext, typeVanIedereRolInContext

De naam rolInContext wordt uiteraard gebruikt voor de selector die de rol met die naam oplevert. De selector iedereRolInContext levert alle rollen. Dus van een Aangifte zouden de instanties van de Aangever, Getuige, Verdachte, enz. allemaal opgeleverd worden. De selector typeVanIedereRolInContext levert het type van elk van die rollen.

## rolIsFunctioneel, propertyIsFunctioneel, rolIsVerplicht, propertyIsVerplicht

De berekening van deze Boolean Properties verloopt anders voor Rollen dan voor Properties.

## boundContexts

Deze operator levert de Contexten die zijn gebonden aan de rollen van een Context. Als een Rol een Rol bindt (die geen BuitenRol is), telt die Rol niet mee in het resultaat. Vaak gaat het hier om gebonden definities, maar dat hoeft niet.

1. Een Rol heeft onvoldoende structurele rijkheid om een type in te kunnen beschrijven. [↑](#footnote-ref-1)
2. Namelijk: neem (op instantieniveau) de rol ‘binnenRolDefinitie’, daarvan de binding, en daarvan de context. Het resultaat is een Context die de definitie is van de BinnenRol (van de Context-definitie waarmee we begonnen). [↑](#footnote-ref-2)
3. Als de modelleur de ariteit van een Rol verandert, moet hij ook de (display)naam van de Rol aanpassen. [↑](#footnote-ref-3)