Metatypen

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 21-11-17 | Versie: 1 |

# Introductie

Perspect is gebouwd op twee typen records, namelijk (Perspect)Context en (Perspect)Rol. Samen vormen zij het abstracte datatype (ADT) PerspectEntity. Hiermee is Perspect dus van boven tot onder gebaseerd op dezelfde conceptuele bouwstenen.

In deze tekst verken ik de manier waarop we in termen van dat basis-vocabulair de complexere concepten zoals Domain en Partij, Property en PropertyPerspectief en typen en instanties daarvan kunnen representeren.

# Rol, property

‘Property’ reserveren we voor de eigenschappen van een rol – en hier bedoelen we een instantie, geen type. Wanneer de modelleur een rol definieert, maakt hij een *type* rol. In de brontekst schrijft de modelleur zo’n type met een hoofdletter: Rol. Bijvoorbeeld :Aangever in de tekst hieronder[[1]](#footnote-1):

:ContextType :Aangifte

:RolInContextType :Aangever

In de expressie waarmee we een berekende rol beschrijven – met andere woorden, in een *query* – schrijven we, als we aangeven dat we van een context een bepaald type Rol selecteren, de naam van die rol met een *kleine letter*. Bijvoorbeeld :aangever in de query hieronder:

:ContextType :Aangifte

:RolInContextType :Aangever

:BerekendeRolType :AfgeleideVanAangever

query :aangever ...

We kunnen denken over de operator :aangever in de query als een ‘property van de context’. In die zin is een rol aan een context gebonden via zo’n property. We zullen in deze tekst echter de term property vermijden als we het over rollen van een context hebben en kortweg schrijven “de rol van de context”.

**Let op:** alle query-operatoren[[2]](#footnote-2) leveren een *sequentie* van rollen, of een sequentie van contexten. In de Purescript representatie is dat een Array. Dit geldt óók voor rollen met een ariteit van 1, zoals binding.

# Grafische notatie

De twee typen records kunnen we grafisch weergeven. Deze weergave sluit aan bij de grafische notatie van Perspect modellen en breidt die uit met property-waarde toekenning en ariteit van rollen.

ID

buitenRol

binnenRol

type

rolInContext

newtype PerspectContext = PerspectContext

{ id :: ID

, pspType :: ID

, binnenRol :: PerspectRol

, buitenRol :: ID

, rolInContext :: StrMap (Array ID)

, gevuldeRollen :: StrMap (Array ID) }

**Opmerkingen.**

1. Het pspType is als rol weergegeven (met de naam ‘type’), maar wordt niet met een gewone RolInContext gemodelleerd. Het is een property van het record met een enkelvoudige waarde. Dientengevolge kan een Context precies één type hebben.
2. Het type kan in een query opgevraagd worden met de operator type.
3. De query-operator rolInContext geeft alle rollen van een Context. We kunnen ons een rol met de naam rolInContext voorstellen die alle in de Context aanwezige rollen als waarde heeft. Uiteraard is de query-operator rolInContext geïmplementeerd als een functie die alle waarden levert van de stringmap die de waarde van de property ‘rolInContext’ van de context is.
4. de query-operatoren binnenRol en buitenRol zijn geen gewone rollen maar kunnen wel zo worden gebruikt. Ook deze operatoren hebben een specifieke implementatie. Overigens zal de modelleur deze operatoren zelden hoeven te gebruiken, omdat
   1. hij een property van een context kan opvragen (waarmee impliciet de binnen- en buitenrol bevraagd worden)
   2. hij een context binden aan een rol (waarmee impliciet de operator buitenRol gebruikt wordt).
5. elke rol-naam (beginnend met een kleine letter!) kan dus als query-operator gebruikt worden

De modelleur beschrijft een type context volgens de regels van de Perspect grammatica, zoals het voorbeeld in de vorige paragraaf.

Een rol wordt als volgt getekend:

Type

context

binding

prop1 = v1

prop2 = v2

newtype PerspectRol = PerspectRol

{ id :: ID

, pspType :: ID

, binding :: Maybe ID

, context :: PerspectContext

, properties :: StrMap (Array String)

, gevuldeRollen :: StrMap (Array ID) }

**Opmerkingen.**

1. De ID van een rol tekenen we zelden. Het is de identificatie van een bepaalde rol-instantie; niet de naam van het type rol! Indien nodig, tekenen we die in de rol.
2. Het type van de rol geven we aan met het label buiten de rol zelf.
3. De query-operator context geeft van een rol zijn context. Deze operator is anders dan gewone rol-properties, want die kunnen alleen een eenvoudige waarde (simpleValue: Number, Boolean, Date of String) hebben. Deze operator is de omgekeerde van rolInContext.
4. De naam van elke property van een rol kan als query-operator gebruikt worden (vergelijk dit met de rolnamen bij een context). Het resultaat is een simpleValue.
5. De query-operator binding geeft van een rol de rol of context waarmee de rol gevuld is.

## Cardinaliteit

Zowel rollen als properties kunnen in principe nul, één, of meer waarden hebben. Let op: een Rol heeft altijd maar één binding, maar een context kan wel meerdere instanties van dezelfde Rol hebben (een Aangifte met meerdere Aangevers, bijvoorbeeld). In tekst geeft de modelleur dat door twee properties van de definitie van een rol een waarde te geven:

* isFunctional
* isVerplicht

(beide met een boolean waarde).

Cardinaliteit is een concept dat alleen relevant is voor *types*. In de grafische notatie, die alleen voor instanties is bedoeld, komt het dus niet voor. Maar omdat we bij een rol wel propertie-assignments kunnen intekenen, kunnen iets van cardinaliteit wel in de tekeningen terug laten komen.

In plaats van de volle naam van de properties uit te spellen, gebruiken we ook een verkorte notatie:

|  |  |  |
| --- | --- | --- |
| isFunctioneel | isVerplicht | Verkort |
| True | True | 1 |
| True | False | 0-1 |
| False | True | 1-n |
| False | False | 0-n |

Grafisch:

MetaType

isVerplicht=false

isFunctioneel=false

TD

MetaType

0-n, TD

# Instanties, typen en metatypen

Een context heeft als type ContextType en die heeft als type MetaType. Zo zijn er ook RolInContextType, PropertyType, enzovoort. In deze tekst geef ik hun definitie en hun gebruik.

Elk MetaType wordt gerepresenteerd met een Context met Rollen. Deze Rollen hebben vaste properties:

1. isFunctional, een boolean property;
2. isVerplicht, ook een boolean property.

Samen coderen ze de ariteit van Rollen en Properties.

Taalgebruik:

* ‘een MetaType’ is een instantie van MetaType. Er is maar een beperkt aantal MetaTypen, namelijk: ContexType, RolInContextType, PropertyType, enzovoort. De naam van al deze instanties eindigt op “Type”.
* ‘een ContextType’ is een instantie van ContextType en definieert dus een type Context. Er is in beginsel een onbegrensd aantal ContextTypen. Een voorbeeld is ‘Aangifte’.
* ‘een Aangifte’ is een instantie van Aangifte. Het is een context.

# MetaType

MetaType is het meest abstracte concept. Het is een context met twee rollen. De belangrijkste is RolInContext. De modelleur mag deze rol zo vaak gebruiken bij een MetaType als hem goeddunkt. De tweede rol is Range en die wordt gebruikt om bij definities van een property een type voor de mogelijke waarden te geven. MetaType heeft geen properties.

MetaType

isVerplicht=false

isFunctioneel=false

TD

RolInContextType

isVerplicht=false

isFunctioneel=true

PA

Range

range=simpleValue

Merk op dat we de typenaam van de rollen niet ingetekend hebben, om de tekeningen overzichtelijk te houden. Alle rollen in de MetaTypes zijn van het type RolInContext.

De definitie van MetaType luidt:

:MetaType :MetaType

:RolInContextType :RolInContextType

isFunctioneel = false

isVerplicht = false

pragma = typeDeclaration

:RolInContextType :Range

isFunctioneel = true

isVerplicht = false

pragma = propertyAssignment

:ContextType :Range

range = simpleValue

Door te specifieren dat de rol niet verplicht is en evenmin functioneel, geven we aan dat we hem in instanties van MetaType willekeurig vaak mogen inzetten.

**De bijzondere behandeling van ‘Type’**. Uiteraard heeft bijvoorbeeld elk ContextType (zoals Aangifte) een type. In de teksteditor schrijft de modelleur een *type declaratie.* Dit is een (Meta)Type naam (zoals ContextType), gevolgd door de naam van het type dat hij gaat beschrijven. Er is nooit een moment dat hij ‘type’ intikt. ‘type’ is impliciet in de syntax.

De typering is natuurlijk expliciet in de representatie in contexten en rollen. Vandaar dat de rol “Type” in de afbeeldingen voor moet kunnen komen. Maar omdat alle MetaTypen het type MetaType hebben, heb ik “Type” weggelaten om de tekeningen te vereenvoudigen.

# ContextType

De context waarmee we het type van context-typen definiëren is ‘ContextType’. ContextType is een MetaType en heeft vier rollen in context (waarbij we type weglaten).

ContextType

RolInContextType

BinnenRolType

BuitenRolType

BerekendeRolType

0-n

TD

0-1

TD

0-1

TD

0-n

TD

:MetaType :ContextType

:RolInContextType :RolInContextType

isFunctional = false

isVerplicht = false

pragma = typeDeclaration

:RolInContextType :BinnenRolType

isFunctional = true

isVerplicht = false

pragma = typeDeclaration

:RolInContextType :BuitenRolType

isFunctional = true

isVerplicht = false

pragma = typeDeclaration

:RolInContextType :BerekendeRolType

isFunctional = false

isVerplicht = false

pragma = typeDeclaration

# Het nuttig gebruik van MetaTypen

Als de modelleur in de (tekst)editor heeft ingetikt:

ContextType :Aangifte

dan laat het systeem hem zien dat op de regel daaronder vier verschillende soorten componenten kunnen worden ingevoegd: BinnenRolType, BuitenRolType, RolInContextType of BerekendeRolType. Het vervolg zou dus kunnen zijn:

ContextType :Aangifte

RolInContextType :Aangever

Oftewel: het systeem voert een query uit op ContextType die als resultaat de contexten levert die de rol RolInContext vullen. Elk van die contexten levert een kandidaat om de definitie van Aangifte mee te vervolgen. Daarbij vergelijkt het systeem de cardinaliteitsproperties van de rol die de kandidaat vult met wat de modelleur al heeft geschreven. Deze query luidt:

ContextType ## rolInContext ~> binding

of, equivalent:

ContextType ## rolInContextType ~> binding

Hier zijn beiden gelijk omdat alle rollen van het type RolInContextType zijn!

# RolInContextType

De context waarmee we het type van een RolInContext definiëren is RolInContextType. RolInContextType is een MetaType en heeft zes rollen.

PropertyType

PropertyPerspectief

MogelijkeBinding

RolInContextType

0-n

TD

0-n

TD

1

RB

IsVerplicht

range=boolean

isFunctioneel

1

PA

1

PA

range=boolean

pragma

range=pragma

0-1

PA

:MetaType :RolInContextType

:RolInContextType :PropertyType

isFunctioneel = false

isVerplicht = false

pragma = typeDefinition

:RolInContextType :PropertyPerspectief

isFunctioneel = false

isVerplicht = false

pragma = typeDefinition

:RolInContextType :MogelijkeBinding

isFunctioneel = true

isVerplicht = true

pragma = roleBinding

:RolInContextType :isVerplicht

isFunctioneel = true

isVerplicht = true

pragma = propertyAssignment

:RolInContextType :isFunctioneel

isFunctioneel = true

isVerplicht = true

pragma = propertyAssignment

:RolInContextType :pragma

isFunctioneel = true

isVerplicht = false

range = pragma

Merk op dat Type niet voorkomt in de naam PropertyPerspectief. Dit is omdat een PropertyPerspectief niet geïnstantieerd wordt. *Instanties* van RolInContextType – dus: types – moeten specificeren of die specifieke rol functioneel is of verplicht.

# Instructies voor semantische controle

RolInContextType heeft één rol in context die gevuld wordt met de context MogelijkeBinding. Deze context heeft verder geen rollen. Als de modelleur een specifiek type rol in context beschrijft (b.v. :Aangever), is het de bedoeling dat hij de rol mogelijkeBinding daarvan *bindt aan* een rol- of contexttype:

:ContextType :Aangifte

:RolInContextType :Aangever

:MogelijkeBinding => :Gebruiker

(een binding-expressie is te herkennen aan de =>).

Maar bij het uitbreiden van een context-type, in deze situatie:

:ContextType :Aangifte

als de modelleur kiest voor uitbreiding met een rol in context:

:ContextType :Aangifte

:RolInContextType

dan is het de bedoeling dat hij een *type declaratie* expressie maakt:

:ContextType :Aangifte

:RolInContextType :Aangever

(in een type-declaratie-expressie staat het type vooraan en de nieuwe instantie van het type erachter).

Er zijn dus twee situaties:

1. de modelleur schrijft een binding-expressie;
2. de modelleur schrijft een type-declaratie-expressie.

We willen dat het systeem hem daarbij leidt of corrigeert. Dat willen we baseren op de metatypes RolInContextType en MogelijkeBinding. Maar die twee beiden contexten en spelen beiden een rol in een context (resp. ContextType en RolInContextType). Hoe moet het systeem bepalen of een binding of een type-declaratie moet worden gemaakt?

We maken dit gedrag mogelijk door een extra property toe te voegen aan de (buitenrol van de) contexten RolInContextType en MogelijkeBinding: pragma.

Pragma is een begrip dat gebruikt wordt in de theorie en praktijk van compilers. Een pragma instrueert een compiler (om een bepaald gedrag uit te voeren). We onderscheiden de volgende mogelijke waarden voor pragma:

* roleBinding (RB)
* propertyAssignment (PA)
* typeDeclaration (TD)
* privateContextProperty (PRP)
* publicContextProperty (PUB)
* Query (Q)

Elk staat voor een bepaald type expressie, respectievelijk:

mogelijkeBinding => :Gebruiker

isFunctional = false

RolInContextType :Aangever

# Pragma’s voor metatypen

Onderstaande tabel geeft voor elk besproken metatype het pragma. Deze termen vormen de mogelijke waarden van de pragma-property van RolInContextType .

|  |  |  |
| --- | --- | --- |
| **MetaType** | **Pragma** | **Voorbeeld expressie** |
| ContextType | typeDeclaration | :Aangifte :aangifte1 |
| RolInContextType | typeDeclaration | :Aangever :aangever1 |
| BinnenRolType | privateContextProperty | private :betrouwbaarheid |
| BuitenRolType | publicContextProperty | Public :urgentie |
| BerekendeRolType | Query | Query :aangever binding |
| PropertyType | typeDeclaration | :PropertyType :urgentie |
| PropertyPerspectief | typeDeclaration | :PropertyPerspectief :xx |
| MogelijkeBinding | roleBinding | :mogelijkeBinding => :Gebruiker |
| IsVerplicht | propertyAssignment | isVerplicht = true |
| isFunctioneel | propertyAssignment | isFunctioneel = true |
| Range | propertyAssignment | Range = Number |

# PropertyType

De context waarmee we het type van een property definiëren is PropertyType. PropertyType is een MetaType en heeft drie rollen.

IsVerplicht

range=boolean

isFunctioneel

Range

PropertyType

1

PA

1

PA

1

PA

range=boolean

range=boolean

:MetaType :PropertyType

:RolInContextType :isFunctional

isFunctioneel = true

isVerplicht = true

:RolInContextType :isVerplicht

isFunctioneel = true

isVerplicht = true

:RolInContextType :range

isFunctioneel = true

isVerplicht = true

Merk op dat de property ‘range’ hier gebruikt wordt. Dit is omdat deze rollen een *property van het PropertyType* definiëren. En voor Properties moet de modelleur een range opgeven.

# Het nuttig gebruik van PropertyType

Dit MetaType wordt anders gebruikt dan ContextType. De rollen suggereren hier geen kandidaat-componenten van een context of rol, maar van een property. Maar een PropertyType heeft slechts drie kenmerken, namelijk isFunctioneel, isVerplicht en range. De modelleur geeft voor elk PropertyType aan wat de waarde is van elk van deze drie.

We zien dat in de definitie van PropertyType terug in het feit dat elk van de drie rollen verplicht en functioneel is. Dat betekent dat als de modelleur een PropertyType definieert, hij alle drie móet gebruiken. Hij moet van een PropertyType aangeven of die property functioneel en/of verplicht is, en wat de range is.

Bij rollen zagen we geen range. Een rol heeft een mogelijke binding. Maar bij properties moeten we een range opgeven. De range van deze drie properties is óók aangegeven in de definitie van het MetaType, met de woorden ‘boolean’ en ‘simpleValue’ In de programmering van Perspect is vastgelegd wat de keuzen voor simpleValue zijn: boolean, number, string en date. Hier eindigt dus de metabeschrijving van Perspect (deze informatie is in de metatypen niet te vinden).

Bijvoorbeeld. Als de gebruiker heeft ingetikt:

ContextType :Aangifte

RolInContextType :Aangever

PropertyType :betrouwbaarheid

dan suggereert het systeem op de volgende regel dat hij een waarde geeft voor één van de drie properties:

ContextType :Aangifte

RolInContextType :Aangever

PropertyType :betrouwbaarheid

isFunctioneel = true

# PropertyPerspectiefType

De context waarmee we het type van een property perspectief definiëren is PropertyPerspectief. PropertyPerspectief is een type en heeft één rol.

PropertyPerspectief

PropertyReferentie

Volgnummer

range=Number

0-n, TD

1, RB

PropertyType

1, PA

:ContextType :PropertyPerspectief

:RolInContextType :PropertyReferentie

isFunctioneel = false

isVerplicht = false

pragma = typeDefinition

:RolInContextType :PropertyType

isFunctioneel = true

isVerplicht = true

:RolInContextType :Volgnummer

isFunctioneel = true

isVerplicht = true

Een propertyperspectief is niet verplicht (bij een rol), en er mogen meerdere perspectieven bij één rol gedefiniëerd worden. PropertyPerspectief heeft één soort rol, de PropertyReferentie. Een specifiek propertyperspectief kan meerdere van deze rollen hebben. Bijvoorbeeld:

:PropertyPerspectief :AdresGegevens

:PropertyReferentie => :Straat

volgnummer = 1

:PropertyReferentie => :Huisnummer

volgnummer = 2

:PropertyReferentie => :Postcode

volgnummer = 3

:PropertyReferentie => :Woonplaats

volgnummer = 4

# Berekende rollen

Een berekende rol ontstaat door een query, uitgevoerd op een context. Met een berekende rol haal je een rol uit een andere context naar voren, naar de context waar de berekende rol aan wordt toegevoegd. Een typisch voorbeeld is de transitive closure van binding, waarmee je van een betrokkene-rol het begin van de rol-telescoop ophaalt. Toegepast op het onderwerp van een actie geeft dat de gebruiker. De context van de gebruiker-rol is het systeem en daar kunnen we dan een peer-rol van opvragen[[3]](#footnote-3). Zoiets:

query onderwerp (closure binding) context peer

Oftewel, van rechts naar links lezend: de peer(s) van de context van de transitieve closure van de binding van het onderwerp (van de actie, die de (hier onzichtbare) context vormt). Maar wat zijn functies, in Perspect?

## Functie als context

In Perspect representeren we een functie als een context met een resultaat-rol, en nul of meer argument rollen.

f

g

h

Application: g( h( f( a ) ) )

a

In dit diagram zien we hoe functie f toegepast wordt op argument a, hoe vervolgens g toegepast wordt op het resultaat daarvan en uiteindelijk h op het resultaat dáárvan.

We kunnen ook een compositie maken van functies:

f

g

h

>>>

Composition: f >>> g >>> h

>>>

De compositie van twee functies is óók een functie, dus de representatie van de compositie-functie heeft vier rollen: twee voor de samen te voegen functies en dan een argument- en resultaatrol. In het onderstaande diagram toont de vulling van rollen de ‘dataflow’ door de compositie:

f

g

h

>>>

Composition: f >>> g >>> h

a

>>>

Deze diagrammen zijn eerder conceptueel dan een technische specificatie. In Perspect hebben we geen expliciete compositie-operator in query-expressies. Het systeem interpreteert de AST (Abstract Syntax Tree) die uit ontleding van de query-tekst resulteert, door middels compositie uit basis-operatoren een functie samen te stellen. Elke stap in de query wordt bij de voorafgaande gevoegd (links-associatief).

Dat betekent dat we kunnen volstaan met de representatie van compositie, en basis-operatoren.

>>>

binding

>>>

binding

binding

Bovenstaand diagram geeft de compositie van driemaal de basis-operator binding. Binding is een simpele functie van rol naar rol.

De operator rolGetter moet partieel worden toegepast voordat een functie van context naar rol ontstaat:

>>>

rolGetter

binding

Aangever

Dit is de AST die de parser maakt van de expressie:

query :aangever :binding

De operator rolGetter is dus impliciet. De operator propertyGetter werkt op vergelijkbare wijze, maar dan op properties van rollen.

De operator filter neemt ook een argument. Deze operator moet expliciet worden geschreven:

query filter :isUrgent :aangifte

(filter de aangiftes die de rol :aangifte spelen in de (verder niet genoemde) context door alleen degene over te houden wier property :isUrgent de waarde true heeft).

De AST is:

>>>

filter

propertyGetter

isUrgent

rolGetter

Aangever

## Query syntax

Een query-expressie is een compositie van functies. Elke rol- en propertynaam wordt geschreven met een kleine beginletter en kunnen we opvatten als een functie van een rol of context. De syntax is links-associatief:

filter :isUrgent :aangifte

leest als:

(filter :isUrgent) :aangifte

Dit betekent dat we haakjes nodig hebben in deze expressie:

:dossier (filter :isUrgent :aangifte)

Immers, we willen de aangiftes uit het dossier filteren; niet het filter van het dossier nemen.

1. De dubbele punt voor :Aangever staat voor de default prefix, die op zijn beurt staat voor de Perspect Namespace. [↑](#footnote-ref-1)
2. Uitgezonderd de operatoren die een property van een Rol opleveren, natuurlijk. [↑](#footnote-ref-2)
3. Het voorbeeld is wat kunstmatig. [↑](#footnote-ref-3)