Tekst representeren

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 28-12-17 | Versie: 1 |

# Introductie

Op het moment van schrijven van dit stuk (28-12-17), wordt tekst in Perspectives op een bepaalde manier gerepresenteerd. In dit stuk vergelijk ik die met een andere vorm van representeren.

Met ‘tekst’ bedoel ik hier: een tekst die contexten en rollen beschrijft.

# Huidige representatie

Tekst wordt niet als zodanig gerepresenteerd, maar afgeleid uit de interne representatie van contexten en rollen. We gebruiken een ‘prettyprinter’ om geïndenteerde tekst te produceren uit de Purescript representatie.

Desalniettemin hebben we in deze interne representatie ruimte opgenomen voor commentaar. Dat is de plek waar commentaar boven en achter een coderegel terechtkomt. Bij het pretttyprinten wordt dit commentaar weer ingevoegd.

Indentering is niet expliciet gerepresenteerd. De prettyprinter werkt met een toestand waarin de indentering wordt bijgehouden.

Deze representatie heeft drie bekende nadelen:

1. commentaar is onderdeel van de contexten en rollen zelf. Het is daardoor niet mogelijk om in de ene tekst bij een context of rol ander commentaar op te nemen dan in de andere tekst;
2. een context die in de lexicale context van een andere is gedefinieerd, zouden we in een tekst in zijn geheel kunnen opnemen, maar ook als verwijzing. De representatie geeft ons niet de mogelijkheid om dit op te slaan, laat staan om dit in de ene tekst anders te doen dan in de andere;
3. heel veel contexten die in de database van Perspectives wordt opgeslagen, worden nooit als tekst gepresenteerd. Desondanks reserveren we voor elk van hen ruimte voor commentaar[[1]](#footnote-1);
4. de volgorde van rollen in een context gaat verloren (en daarmee de volgorde van definities in een tekst);
5. de volgorde van property-assignments in een rol gaat verloren;
6. de prefixes gaan verloren.
7. Een specifieke schrijfwijze van een naam gaat verloren (prefixed versus impliciet namespaced, bijvoorbeeld).

Punt twee kunnen we uitbreiden naar andere aspecten van weergave, zoals bijvoorbeeld in- en uitklappen (dit wordt ondersteund door de Ace editor) en echte vormgevingsfacetten zoals font en fontgrootte.

# Expliciete representatie van tekst

In zijn huidige vorm is de syntax van Perspectives sterk regel georiënteerd (aangevuld met de betekenis van inspringen). Van een tekstregel zouden we willen kunnen vastleggen:

* de code zelf, al dan niet gestructureerd.
* het commentaar ervoor en erachter
* de indentering (als het een context-declaratie of rolbinding betreft)
* of een rolbinding als een referentie naar een context moet worden geschreven, of dat de definitie van deze context zelf opgenomen moet worden
* of een ingesprongen blok opgeklapt moet worden weergegeven.

Al deze informatie kan in termen van eenvoudige waarden (strings, getallen, booleans) worden vastgelegd. Dat betekent dat we met een rol met properties zouden kunnen volstaan (waarbij regelpositie óók als property vastgelegd kan worden).

Het is handig om een regel met daaronder een ingesprongen blok van regels als tekstuele eenheid te beschouwen. Er zijn twee soorten regel-expressies die door zo’n blok gevolgd (kunnen) worden:

1. een context-declaratie
2. een rolbinding

Beiden lenen zich dus voor representatie met een context, waarbij de verschillende regels evenzovele rollen zijn. Een context-declaratie-context heeft de volgende soorten rollen:

* de context-declaratie zelf;
* een public property
* een private property
* een rolbinding

Een rolbinding-context heeft de volgende soorten rollen:

* de rolbinding zelf
* een rol property

Verder kunnen we de tekst als geheel representeren op vergelijkbare wijze als een context, maar met een iets andere syntax:

* de Context <name> as <prefix> regel
* een regel voor elke import expressie
* alle definities.

Of, beter nog, we kunnen import expressies toestaan in elke context en tevens een prefix toestaan in elke context declaratie.

## Vergelijking met diagrammen

Om contexten en rollen in diagrammen weer te geven, hebben we op zijn minst ook positie-informatie nodig. Verder zullen we in diagrammen bepaalde onderdelen bewust weglaten, zoals sommige rollen. Vergelijk deze selectieve presentatie met het al dan niet uitschrijven van een lexicaal geneste context, of met het al dan niet inklappen van geïndenteerde tekst.

In principe zouden we deze informatie op kunnen nemen in de contexten en rollen zelf (zoals we met commentaar hebben gedaan). Maar daarmee vergroten we de nadelen van de huidige representatie alleen maar. Zeker bij diagrammen zal het belangrijk zijn om in het ene diagram andere informatie van dezelfde context of rol te tonen, dan in het andere.

Zonder hier in detail te treden kunnen we ons voorstellen dat we een diagram op vergelijkbare wijze in contexten en rollen kunnen representeren als tekst.

## Voor- en nadelen van expliciete representatie van tekst

De zeven nadelen van de huidige representatie gelden niet voor de expliciete representatie. Maar merk op dat het voordeel ‘commentaar is een functie van context èn de tekst’ ook als nadeel kan worden gezien. Als de modelleur een context opneemt in een tekst, kunnen we hem dan ook de mogelijkheid bieden een al bestaand commentaar over te nemen? Soms hoort een commentaar onder alle condities bij de code!

Door tekst expliciet te representeren, kunnen we eenvoudig een scheiding aanbrengen tussen de fase waarin de modelleur een tekst maakt, en die waarin hij zijn beschreven contexten en rollen toevoegt aan zijn bubble.

Een nadeel is dat we de expliciete tekst en de interne representatie van de beschreven contexten en rollen altijd met elkaar in overeenstemming moeten houden. Een dubbele administratie is altijd lastiger dan een enkelvoudige.

# Van expliciete tekstrepresentatie naar contexten en rollen

1. Dit nadeel verdwijnt zodra we de representatie van contexten verbergen achter een foreign functionele interface en in javascript *just in time* onderdelen aanmaken. [↑](#footnote-ref-1)