Aspects and role filling

Joop Ringelberg 02-02-22 Version: 1

Introduction

In this text we explore the consequences of the introduction of Aspect Roles for the role filling relation.

An Aspect Role is one that is taken, in *design time*, from one context and added to another. As an example, think of a generic role Driver that can be added to various types of transport, e.g. a transport of ill persons to hospitals (PatientTransport) and ordinary taxi rides (PrivateTransport). Instances of such roles will be stored with their 'host' context type, e.g. a PatientTransport instance, indexed by their original, fully qualified name. So, supposing Driver is a role of the generic Ride context, a PatientTransport context instance would have Ride$Driver instances. In contrast, a Patient role modelled on PatientTransport would have instances typed by PatientTransport$Patient.

The takeaway is that a role type may be used in many contexts. This is important when it comes to queries, which after all are nothing but descriptions of paths through the web of Context and Role instances, expressed in terms of types.

An example will make this clear. Suppose a Taxi company wants to compile a list of Patients driven by a particular Employee. A query like this would seem to do the job:

```
Employee >> fills Driver >> context >> Patient
```

However, the compiler should say no to this query! Why? Because Driver is a role in the context of the generic Ride context and that context has no Patient role. Instead, we should express the query like this:

```
Employee >> fills Driver in PatientTransport >> context >> Patient
```

This underlines the fact that Driver *as such* does not identify a role type; we must combine it with a context type. Let me hasten to add that we will interpret 'Driver' as meaning: the role Driver in its syntactical context, i.e. Ride. Let's introduce a Purescript data type for this combination:

```
newtype RoleInContext = RoleInContext {context :: ContextType, role ::
EnumeratedRoleType}
```

Representation of Queries

We represent queries with a `QueryFunctionDescription` that gives us the domain, range, function, and some meta-properties and the argument expressions that supply values to be bound to function parameters. Domain and range are constructed as an Abstract Data Type (ADT) of a base type and role domains are based on `EnumeratedRoleType`s. This we need

to change to `RoleInContext`, because a path through type-space is not along context- and role types; it is along context types and *roles in a particular context* – not necessarily their lexical context!

Additions to the language

As the example in the previous paragraph illustrated, we need new constructs in our language. We need an extra qualification when describing what we fill a role with. We also need to be able to say what context we end up in when we traverse the fills relation. There is only one more way for a query to end up in a role, and that is when we move from a context to a role. The context type is obvious in that case[1].

FilledBy in role definition

First of all, we should be able to define a role type's filler in terms of the *combination* of another role type and a context:

```
user Attendant filledBy Driver in PatientTransport
```

Extending our example, suppose the hospital requires the person delivering a patient to register as Attendant, we see we cannot do with just `Driver`, but need to extend that to `Driver in PatientTransport`. Again, when no context type is given, the system interprets this as that the syntactically embedding context of the role type is meant.

Fills (in queries)

Secondly, we need to be able to qualify a role with a context type in queries when using the `fills` keyword:

```
Employee >> fills Driver in PatientTransport >> context >> Patient
```

FilledBy (in queries)

Finally, we can qualify the role with a context when traversing the fills relation in the other direction:

```
User >> filledBy Driver in PatientTransport
```

Compiling these expressions to QueryFunctionDescriptions

How do we construct the `RoleInContext` structures in the domains and ranges of the query function descriptions for the steps that take us to a role instance?

[1] However, when moving from context type C to role type A, we should check whether A is available in C. It will be when A is in the namespace C; and it will be when C has A's context as Aspect.

Role step

The `RoleInContext` that must be the range of the role step is conceptually easy: it is just the combination of the `ContextType` that is the domain of the step, combined with the `EnumeratedRoleType` that is its range.

However, in compile time the domain is an Abstract Data Type (ADT) constructed from `ContextType's`. We construct the `ADT RoleInContext` from it by traversing the `ADT ContextType` with a function

```
ContextType -> RoleInContext
```

that tacks on the `EnumeratedRoleType` onto each `ContextType`.

Fills step