The Operational Meaning of a Perspective

Joop Ringelberg 10-12-20 Version: 1

Introduction

We have four categories of use for a perspective:

1. We use perspectives when creating screens so the user can consult his data. This usage ranges from creating screens automatically, to informing components on the client side about what properties are available on a role, to the user.
   Part of this use is to keep the results of queries made by a client up to date with respect to changes to the underlying data.
2. We also use perspectives to enable the user to change data and prevent him from making unauthorised changes.
3. Perspectives guide us when a change to a context, role or property occurs (is made by the user). They tell us which peers should be informed of that change (a process called synchronisation). A variant on this theme is when a peer is added to a context and we need to send him all its details (a process called serialisation).
4. Finally, when we receive a change (as a delta packed in a transaction), we use perspectives to judge whether the author of that change was authorised to do so before we apply the delta to local data.

In analysing the usage of perspectives, it is useful to distinguish roles from properties. We will use the words role-perspective and property-perspective.

Perspectives have an object. These objects are, invariably, roles. It is useful to analyse perspectives on Enumerated roles apart from perspectives on Calculated roles.

In our analysis we'll be mainly concerned with usages 3 and 4 above: synchronisation and authorisation, respectively.

Perspective on an Enumerated Role

In Perspectives, we treat an instance of an Enumerated role and its binding as a unit with respect to properties. By that we mean that a perspective on a role provides access to all properties of the role **and** those of its binding (if not limited by a View).

So, for a user U having a perspective on an Enumerated Role ER:

- the role-perspective means:
  - an instance E of ER should be synchronised for U;
  - an instance B of type BR that is the binding of ER should also be synchronised for U;

    - U can legally Create, Delete and Bind an instance of ER (depending on the verbs in the perspective);
    - However, U cannot legally Bind, Create or Delete an instance of BR.
- the property-perspective (possibly limited by the view of the perspective) means:
    - all values for properties of E and B should be synchronised for U;
    - U can legally Create, Delete and Change values of properties of E and B.

Compound binding: Binding Role Graph (BRG)

A complication arises because the binding of a role may be an expression constructed out of multiple Role types with the operators `AND` (for multiple bindings of a single instance) and `OR` (for alternative binding types of a single instance). The type of such an expression is represented as an `ADT EnumeratedRole`, using the constructors `SUM` (for expressions with `OR`) and `PRODUCT` (for expressions with `AND`).

So, in general, we say that the binding of an `EnumeratedRole` is specified as an `ADT EnumeratedRole`, where the simple case is `ST <EnumeratedRole>`.

This complication works out differently for role-perspectives than for property-perspectives.

The role-perspective applies to all role types in the ADT. So if the binding of a role is compound, we have to synchronise the relevant instances of **all** types in the binding ADT. However, as a user does not have the right to Bind, Create or Delete an instance of the binding of a role just by virtue of having a perspective on that role, we have to reject any (incoming) modification to instances of the types in the binding ADT (when justified solely by the perspective on the role that has that binding).

The property-perspective is different and depends on the constructors in the ADT.

- when role types are combined with `SUM`, the perspective applies just to the property types in the *intersection* of the property types of the members of the `SUM`;
- instead, on combining with `PRODUCT`, it applies to the *union* of those properties.

To further clarify the role-perspective: it applies to all types, **regardless** of the constructor (`SUM` or `PRODUCT`).

Perspective on a Calculated Role

A Calculated role is defined by a path expression. As with bindings, the basic building blocks of those expressions are Role types (Calculated and Enumerated alike). However, the operators are different: they represent traversals of the graph consisting of roles and contexts.

A simple path leads from a Context type to a Role type, often in another context[1]. Either that role is Enumerated, or it is Calculated, leading (recursively, in the end) to an Enumerated Role itself (it is an error to define two Calculated roles in terms of each other). So we see that a Calculated role in one context really is an Enumerated role in another context.

This means that we can express the type of a Calculated Role in terms of Enumerated Roles, using the constructors of ADT. For a simple path-based Calculated Role, its type is ST <some enumerated role>.

However, we can construct expressions with operators that combine paths: union and intersection. The type of a Calculated role constructed with union or intersection is an ADT constructed with SUM.

We can apply the insight gained from analysing compound bindings to this case. We've seen that for role-perspectives, irrelevant of the constructor used (`SUM` or `PRODUCT`), the perspective applies to all role types occurring.

However, for property-perspectives, the perspective just applies to the intersection of the properties of the individual types.

Notice that we have no path operator that constructs expressions with an `PRODUCT` type.

Serialisation for Calculated Roles

In the analysis above we've focused on the endpoint(s) of a Calculated role expression, treating a Calculated role as a remote Enumerated role. This falls short when it comes to serialisation (in the third category of usage of perspectives). If we introduce a peer in a context where he has a perspective on a remote Enumerated role, we cannot suffice with just sending the context and its roles. We also have to provide the remote role, its context, **and all roles and contexts on the path to it**.

[1] Though we cast the object of a Perspective on a role in the same context as its user role, as a (singleton) path as well for the sake of uniformity of representation and analysis.