Who authors?

Joop Ringelberg 29-07-20 Version: 1

Problem statement

Changes to Perspectives State (or, equivalently, the Perspectives Universe) come into being by executing functions in one of these modules:

- the `Perspectives.Assignment.Update` module,
- the `Perspectives.Instances.Builders` module or
- the `Perspectives.SaveUserData` module.

All these functions return results in (monads based on) the monad `MonadPerspectivesTransaction`.

To accept changes coming from another user, they have to be authorized. We achieve this by sending deltas that are signed by the other user, and that explicitly represent (contain) the roles in which users made their changes.

In this text I treat the question: how do we compute the role that is authorized to make the changes, in the modules listed above? The answer consists of two parts, because changes are either instigated either directly by the user, mediated by the GUI, through the API; or are executed by bots behaving on his (the user's) behalf.

Bots

A bot appears in model source code as a rule, that is given for a particular user. For example:

```
    bot: for User
      -- This rule creates an entry in IndexedContexts if its model has 
been taken in use.
      perspective on: UnconnectedIndexedContext
        if exists UnconnectedIndexedContext then
          bind object to IndexedContexts
```

This bot (taken from `model:System`) does some bookkeeping for the `User` of `PerspectivesSystem`.

As the example shows, we must stipulate the user role a bot works on behalf of. This source code translates to an Action and the user (i.e. the type) shows up as the Subject member of that Action.

In the module `Perspectives.Actions` we compile bot actions to executable code. Part of that code is a function that computes a Unit result in MonadPerspectivesTransaction, a so-called Updater:

```
type Updater s = s -> MonadPerspectivesTransaction Unit
```

This Updater executes the effect described in the right hand side (RHS) of the bot rule. Now, just prior to executing this RHS, we add the subject of the action to the state of `MonadPerspectivesTransaction`. This state is a `Transaction`: it contains a member `authoringRole` and we bind the subject to it.

As soon as the RHS has finished, we restore the previous value of `authoringRole` (if any) in the `Transaction`. In other words, we *wrap* the Updater in a state that holds the authoring user role. State changes may trigger bot rules; this mechanism makes sure that all functions in the aforementioned three modules can just take the author role right from state.

API calls

The end user can only change the Perspectives Universe through the Graphical User Interface. Now this interface consists of screens that are built from parts that represent a particular user role's perspective on a context. When the user navigates to another context, just prior to displaying such a part, the GUI code asks the PDR for the role that the user plays in that context.

With each and every call that changes Perspectives State, the GUI returns that role. It is the authoring role. In the API, we put that role in the Transaction in which the change is computed, thus ensuring that all functions in the three modules mentioned above are informed correctly about the authoring role[1].

[1] If no authoring role is provided by the API caller, we take it to be the System User.