Become

Joop Ringelberg 07-10-20 Version: 1

Introduction

This text is a proposal for a new verb to be used in the Perspectives Language.

Consider the user role `Chatter` in `ChatApp` in `model:SimpleChat`. It has a perspective on the context role `Chats` (filled by `Chat`). But it also fills the role `Initiator` in `Chat`:

```
case: ChatApp
  context: Chats = callExternal … returns: Chat$External
  user: Chatter (mandatory, functional) filledBy: sys:..$User
    perspective on: Chats
case: Chat
  user: Initiator (mandatory, functional) filledBy: Chatter
```

(simplified version).

A very common use case based on this model is to create

- a new instance of the context role `Chats`
- a new instance of the context `Chat`
- and in it a new instance of the user role `Initiator`
- filled by the user role `Chatter`.

Because this pattern of creation is so common, we have a verb for it: `Become`. By using this verb in a perspective, we indicate that we want an action that performs all these four steps in a single operation. It looks like this in a model text:

```
  user: Chatter (mandatory, functional) filledBy: sys:..$User
    perspective on: Chats
    perspective on: Initiator in Chats Become
```

Notice the bit of new syntax here: `Initiator in Chats`. We combine a user role (`Initiator`) with a context role (`Chats`). To use this legally, the model must comply to these conditions:

1. obviously, both roles must exist;
2. the context role must filled by a context type that is indeed equipped with the user role;
3. this latter user role must be filled by the user role that has the perspective.

Assignment operator

The new verb corresponds to a new assignment operator to be used in the right hand side of bots:

```
make <userrole1> a <userrole2> in <contextrole>
```

`<userrole1>` is to be substituted by the user role that has the Become perspective. `<userrole2>` is to be substituted by the user role in the embedded context, while `<contextrole>` is the context role in the embedding context. So we have the following constraints:

1. the value of the `filledBy` clause of `<contextrole>` must be the embedded context, let's say EC;
2. `<userrole2>` must be a role in EC;
3. `<userrole2>` must have a `filledBy` clause with the value `<userrole1>`.
4. All three placeholders must be replaced with a role identifier; there cannot be another expression. Specifically, the `<contextrole>` cannot be replaced with an expression selecting some context.

Why `make` and not `become`? Obviously, the syntax would not read as naturally with `become` substituted for `make`. But it is also quite natural when we consider that it is *the bot* that actually 'makes' X a Y.

The `make` operator suggests a wider range of use cases, e.g. for non-user roles. However, the current proposal does not support that!

Usage in screens

To use this new action from within a user role screen, use the component `Become`:

```
<Become
   fillingrole="Chatter"
   filledrole="model:SimpleChat$Chat$Initiator">
   { props => <Button
                  onClick = { e =>
                     props.become()
                     .then(erole => setSelectedChat([erole]))}
               >Start a chat</Button>
   }
</Become>
```

This code example will lead to a simple button with the text "Start a chat". Clicking it will open a window on the newly created Chat instance.

Notice that the function become returns a promise that resolves to the external role of the new context. Usually we want to provide access to that context: in the example, the code will immediately open it.