# The Body-with-Account Pattern

Joop Ringelberg 27-08-21 Version: 1

## Introduction

The concept of membership, or *account*, occurs frequently in real world situations. It involves a *body* (like an organisation, but it may also be more like a service, such as a particular course given by an educational institution) and people that are its members, or, in other words, haven an account with it.

This text describes this pattern in terms of context- and role types that can be used as Aspects in models that apply to concrete situations. This pattern can be thought of as an elaboration of the *Guest Pattern*.

### Features of the pattern

The account should be private, while the Body should be publicly accessible, in order for anyone to be able to apply. An Account obviously needs a perspective on itself, including the possibility to retract from the Body (to end the contract, so, in effect, to no longer have an account).

An application may be in one of three states: waiting for approval, accepted and rejected.

## The notion of Contract

One way to implement this pattern is to have the Visitor create a new, private Context that has himself filling the role of Contractant and a representative of the Body as Contracter.

The Body itself could have a calculated role that drags the Contractant into it. These roles then would have perspectives on the features of the Body hidden to non-members.

What variable information goes into a contract?

- the parties
- begin- and possibly end date
- a fee, maybe

Further, the Contractant could have credentials to access resources provided by the Body.

We could put all this information on the Account role itself in the form of properties:

- begin- and end date
- fee
- credentials.

So we do not really need a contract.

# Implementing the pattern

In order to represent the various information, we either need an embedded context or an enumerated role. Since we have rejected the notion of contract, we will use an enumerated role. However, the context being public, it must be an *unlinked* role (otherwise visitors would receive a representation of the context with all account roles, disclosing, at least, the number of accounts).

To protect the privacy of Accounts, its perspective on itself needs to be selfOnly.

In order to be able to see the context at all before an end user has established an Account, we need a Calculated role. This role must have a perspective that allows it to create an Account, that is, to initiate the process that leads to either a full account or its rejection. This we achieve with a Guest role that computes sys:Me.

Depending on the preference for enumerated roles over calculated roles, as soon as the Guest has created an instance of Accounts that is (ultimately) filled with himself, the Accounts type will turn up as the type of the role played by sys:Me in the Body context.

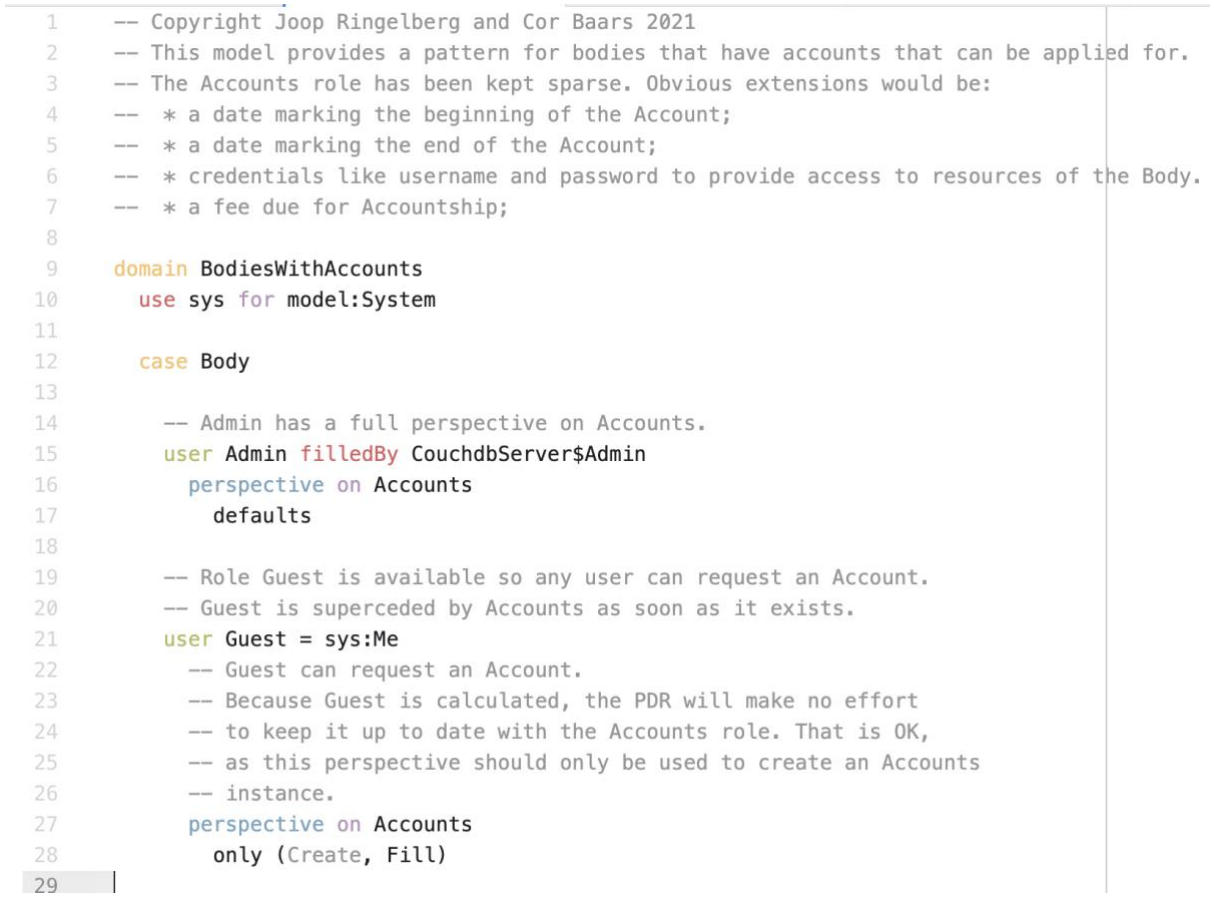
We then have three states on Accounts:

- Waiting
- Rejected
- Accepted

# The Model

Figure 1 shows part of the model BodiesWIthAccounts. The Admin role is quite simple, just providing access to the Accounts. Guest is a calculated role that has a perspective that allows it to create an instance of Accounts.

Figure 2 shows the Accounts role.



```
1   -- Copyright Joop Ringelberg and Cor Baars 2021
2   -- This model provides a pattern for bodies that have accounts that can be applied for.
3   -- The Accounts role has been kept sparse. Obvious extensions would be:
4   --  * a date marking the beginning of the Account;
5   --  * a date marking the end of the Account;
6   --  * credentials like username and password to provide access to resources of the Body.
7   --  * a fee due for Accountship;
8
9   domain BodiesWithAccounts
10    use sys for model:System
11
12    case Body
13
14      -- Admin has a full perspective on Accounts.
15      user Admin filledBy CouchdbServer$Admin
16        perspective on Accounts
17          defaults
18
19      -- Role Guest is available so any user can request an Account.
20      -- Guest is superceded by Accounts as soon as it exists.
21      user Guest = sys:Me
22        -- Guest can request an Account.
23        -- Because Guest is calculated, the PDR will make no effort
24        -- to keep it up to date with the Accounts role. That is OK,
25        -- as this perspective should only be used to create an Accounts
26        -- instance.
27        perspective on Accounts
28          only (Create, Fill)
29
```

*Figure 1. The model BodiesWithAccounts, showing the Admin and Guest role*

```
30        -- User Accounts should stored in private space.
31        -- By making the role unlinked, there are no references from the context to
32        -- Accounts role instances (but there are references the other way round).
33        -- This allows us to create a screen to browse through Accounts without
34        -- loading all references at once with the context.
35        -- Specialisation may restrict their fillers.
36        user Accounts (unlinked)
37          property IsAccepted (Boolean)
38          property IsRejected (Boolean)
39          state Root = true
40            -- Use this state to inform the applicant that his case is in
41            -- consideration.
42            state Waiting = not IsRejected and not IsAccepted
43              perspective on Accounts
44                -- Account can see he is not yet rejected,
45                -- but not accepted either.
46                props (IsRejected, isAccepted) verbs (Consult)
47                selfOnly
48
49            -- Use this state to inform the applicant that he will not become
50            -- a member.
51            state Rejected = IsRejected
52              perspective on Accounts
53                -- Guest can see his request for an account is rejected.
54                props (IsRejected, isAccepted) verbs (Consult)
55                selfOnly
56
57            -- Use this state to provide perspectives on the various
58            -- things that are hidden for non-members.
59            -- This has to be done in the specialisations of this User role.
60            state Accepted = IsAccepted
61              -- An Account can see just himself and can decide to be no longer
62              -- a member, by removing himself from the role.
63              perspective on Accounts
64                only (RemoveFiller)
65                verbs (Consult)
66                selfonly
```

*Figure 2. The Accounts role of BodiesWithAccounts.*