Booting a domain

Joop Ringelberg 28-07-20 Version: 1

Introduction

When a user adds a Model to his PDR, he automatically adds some context- and role instances, too. Collectively, we call them the *base model instances*. Each Domain contains a context that describes the model (it is a specialisation of `sys:Model`). Moreover, each Domain (may) contain a context that serves as the root of other instances (it is a specialisation of `sys:RootContext`). Such a root context invariably has a user role that is a specialisation of `sys:RootContext$User`.

Technically, these instances are constructed from a CRL file that is included with the Model. As with all modifications of the data that is part of the Perspectives Universe, these role- and context instances should contain signed deltas detailing what user, in what capacity (role), constructed them[1]. However, CRL is not a format that supports such information (and, indeed, we will see that it would not be appropriate for this file to carry it). This text explains how these deltas come into being.

Who constructs the base model instances?

On first sight, it might appear that the author of the model must be the author of the base model instances. This certainly holds true for some instances, such as the model description. However, it would be inappropriate for *indexed contexts* (and roles). An indexed context is one that is on the one hand unique for each user, but on the other hand is identified *by the same name* by those users – `sys:MySystem` being the prime example. Such contexts should be constructed by the users themselves – not by the author of the model.

When the PDR parses the CRL file and constructs the instances it describes, it must find the appropriate role that is authorized to construct each of them. In the general case, that may be an undecidable problem in the sense that there may be multiple roles that are authorized to construct a particular context – and then how should the system choose?

Instead, we arrange for things such that all these instances can be created by the `User` role of the `PerspectivesSystem`. We do this by modelling some Aspects in the system model and by stipulating that modellers add these Aspects to their types. This limits modellers in the instances they can ship with their models. However, we think that will be no problem. Moreover, there seems to be an escape for the modeller who really needs to add instances of types that cannot use the aforementioned Aspects: he can introduce a

[1] The reason being that whenever the PDR ships these instances to the PDR of another user, that PDR will want to check that they were constructed by authorized roles.

`Guest` role and have it filled with the system `User` role, then give that role the necessary perspectives.

Aspects

sys:Model

The type `sys:Model` should be used as an aspect of the type that describes the model. This means that each Domain has a unique type that describes it. These types can be empty, otherwise; we just need a unique type for other reasons.

To give an example: in `model:SimpleChat` we have the following type:

```
  case: Model
    external:
      aspect: sys:Model$External
    aspect: sys:Model
```

The modeller then includes the following instance:

```
  chat:Model usr:SimpleChatModel
    extern sys:Model$External$Name = "Simple Chat"
```

(only part of the type is given).

sys:RootContext

The type sys:RootContext should be used as an aspect of the type that functions as the root context for a model. Again, in `model:SimpleChat` we have:

```
  case: ChatApp
    external:
      aspect: sys:RootContext$External
    aspect: sys:RootContext
```

We also have a role Chatter that has an aspect `sys:RootContext$User`:

```
    user: Chatter (mandatory, functional) filledBy:
sys:PerspectivesSystem$User
      aspect: sys:RootContext$User
      perspective on: Chats
```

Perspectives on Aspect roles

These aspects should be used because we have provided the `sys:PerspectivesSystem$User` role with perspectives on them. This allows us to have the `User` role legally construct instances of models and root contexts.

Instances of models

To start with instances of models:

```
user: User (mandatory, functional)
  perspective on: ModelsInUse
```

Being a default perspective, it includes the verb CreateFiller. That allows `User` to construct *instances of the binding of* `ModelsInUse`. As this latter role is restricted to fillers of type `Model`, we see that `User` now has the authority to construct instances of all types that are models – hence the requirement that modellers give their model description that Aspect.

Root contexts

Next, let's consider root contexts. These are special cases. They are contexts that are not bound in another context, or need not be bound. Consequently, our semantics for creating contexts fail: there is no context role that a user can have a perspective on, that allows that user to create the context and its external role. A Database Query Role gives an escape route, but we don't want to tie the two together.

Hence, we just make an exception for root contexts. The `sys:PerspectivesSystem$User` role is by decree authorized to create such contexts and their external role. So, for example, the context type `model:SimpleChat$ChatApp`, having the aspect `sys:RootContext`, can be instantiated by `sys:PerspectivesSystem$User`.

Users in root contexts

`ChatApp` has a user role `Chatter`. This role – filled by `User` – is authorized to construct new chats. We have to make it legal for the `User` role to construct and fill the `Chatter` role. Again, like with root contexts, we make an exception. It is legal, by decree, for the `sys:PerspectivesSystem$User` role to create and fill roles that have the aspect `sys:RootContext$RootUser` (and notice that, because each user is in the end a `sys:PerspectivesSystem$User`, everyone can create these roles). By giving `Chatter` that aspect, all is well.

Special case: the external role of Model

The representation of the external role of sys:Model is part of the model file (DomeinFile). This is because a repository provides a list of such roles as its inventory. It also allows us to ship the relevant role representations from the repository to the PDR of the users consulting the repository. Thus, these external roles can be verified to be constructed by their model authors.