Aspect states

Joop Ringelberg 02-11-21 Version: 1

Introduction

In this text I describe a feature that is not implemented, but that may be important to add. It is about that a modeller might want to *extend the state definition* of an Aspect of one of her types. Consider a role type R that has Aspect A, where A has a state S. Now it is quite natural to model R like this:

```
thing R
     aspect A
     on entry of A$S
        do
          …
```

So while defining R, we refer to *a state of A*, and we add to the actions that should be carried out automatically. This is perfectly understandable; probably we want the original automatic actions (those of S, if any) to be carried out first and then those we provide in the definition of R.

Problem statement

However, implementation runs into the problem: how and where do we store the extension, if A is defined in another model than R? This is a common case; we use a model to provide Aspects, like a pattern library.

Extension of the state S obviously has to be restricted in use to the type R! We do not want to extend S for all other situations where A is used as an aspect, so we cannot modify the model that contains A.

Design

A possible solution would be to extend the internal definition of Enumerated roles of a model M (as the only role types that can have states) with a new member: a table that maps Aspect state names to *state extension names* within model M. Also, we'd add a section of *state extensions* to the DomeinFile (the Purescript type that holds all definition in a model). So M would contain state extensions and its Enumerated Role types may refer to those extensions (the same would hold, mutatis mutandis, for context types).

We can model a state extension like this:

```
type StateExtensionRecord =
     -- the key in these maps is the subject the effect or notification or
perspective is for.
```

```
    { notifyOnEntry :: EncodableMap RoleType Notification
    , notifyOnExit :: EncodableMap RoleType Notification
    , automaticOnEntry :: EncodableMap RoleType Action
    , automaticOnExit :: EncodableMap RoleType Action
   , perspectivesOnEntry :: EncodableMap RoleType
StateDependentPerspective
    , subStates :: Array StateIdentifier
    }
```

Using state extensions

Module Perspectives.RoleStateCompiler should use those extensions, for example in the function `compileState`. This function compiles the automatic actions that should be carried out on entry and exit of a particular state.

The StatefulObject of the state that is compiled is a reference to (in the case of the RoleStateCompiler) an EnumeratedRoleType. `compileState` should find all Aspects of that role type that extend the state that is being compiled and add the extensions to the various members of that state.

Automatic Actions

It should follow this algorithm to add extensions in the right order:

1. Starting with the StateFulObject type, do a depth-first Aspect walk;
2. Remember the path to the root;
3. Keep each path that ends in the Aspect that provides the state that is being compiled (if any);
4. Then, in an arbitrary order for all paths found:
5. Execute actions in a path in the order from leaf (Aspect) to root (type of StateFulObject).

Notifications

The notifications contained in an extension should probably be executed in the same order as described for Automatic Actions, but as there are no side effects, there will be no effect on the state of the system if they are executed in arbitrary order. However, the user experience is probably better if notifications of extensions come after those of the original Aspect.

Perspectives

State dependent perspectives are used to synchronise part of the local network of contexts and roles with users that have acquired a wider perspective. The order in which this happens is probably of no concern.

Substates

An important facet of extending Aspect states is the ability to refine an Aspect state into substates. I can find no objection against executing these substates in an arbitrary order, as long as substates are executed later then their parent states.

Work around for the current implementation

As we have not yet implemented this feature, what is a modeler to do, who wants to extend an Aspect state?

The simple solution is to copy the state definition of the Aspect to the type in one's model. Now both the Aspect state and the Role state will be executed, albeit in an unspecified order.