OPCOACH
FORMATION ET EXPERTISE ECLIPSE

How to merge application models

Auteur	Olivier Prouvost
Date	11/8/14
Référence	OPC 10 ECL ME4 EXP 01
Version	A

Historique

Date	Version	Auteur	Observations
11/8/14	A	Olivier Prouvost	Create document

Table des matières

1 INTRODUCTION

The goal of this document is to describe a global policy to merge a master E4 application model with its sub fragments.

1.1. The problem

The modularity with E4 application is limited today by the model fragment mechanism. Only two ways are possible :

1. extending the model by model fragment
2. extending the model by processor .

1.1.1. Model fragment

A model fragment is a piece of model described using a new syntax. A fragment can be created using the 'extract fragment' from a master model.

This approach is ID centric : you must know the ID of an object to contribute in. For instance, to contribute in the addon list of a 'master' model, you need to know the ID of the application modmel which contains the addon list.

So, if the JDT for instance provides a model fragment to be included in any application it will not work unless the ID of the application is definitively fixed. But the problem will appear also for perpective stack , shared elements, part descriptors, bindings, and so on…

The second problem with the model fragment is its description. Although we have a great model editor to describe hierarchically the application, the description in model fragment is always linear and no hierarchy can be defined.

This is not an efficient solution.

1.1.2. Processor

The processor method is a different way to initialize an existing model. It just execute a piece of code where it is possible to inject the main application. In this case we don't care about ID and we can write the code to add or remove some pieces of model in the main application.

But the problem is that is is a pain to write this code while we have a cool and powerful model editor…

2 PROPOSING A THIRD WAY TO CONTRIBUTE TO THE MODEL

Why could not we use another model to contribute to a model ? This solution would offer a lot of benefits :

- model editor could be used directly
- the model contribution could be tested as a full application
- model could be merged whatever the parent model is
- it would be similar with the extension mechanism but with a hierarchical description.

2.1. The problem to solve

If we merge two models one must be considered as the master model. The master model will define the concepts and the master object of the expected applicaiton. For instance, if this model contains a perspective stack it will be kept and filled with the contents of the other perspective stack present in sub models.

The order of merge could be also considered. As for extensions in Eclipse 3 we must consider that merging models must give the same result whatever the order of treatment. Of course, the result model could be sorted differently but it is up to the developper to propose a generic sort when data are displayed. For instance, if several sub models contribute to the perspective stack, a perspective switcher could display the perspectives in the alphabetical order.

The last problem is to solve the merge policy for each object. Actually merging a sub model in a master model should not create a new application model, because the master is the target. But for a perspective stack, it should be created if none is defined in master. Several policy could be found :

- **add if not exist** : the merge will create the same object in the master. If it already exists nothing will be created (for instance for a command already defined). The test to know if object already exists depends on the object nature.
- **add** : the merge will add the current object in the corresponding parent in master. No test is done to know if object is already present (not sure this policy will be useful).
- **nothing to do** : the merge consist to merge nothing because object is unique in master

To know if an object already exist in master we must test different fields depending on the cases. For instance, to check if a command is already defined in master (org.eclipse.ui.edit.copy), we must search for a command with the same ID. On the other hand, testing if an handler is defined for a command must be tested using the command relation and then check if the ID is the same.

If we consider the different kinds of objects to be merged, there are not so much levels to manage and the merge should be a quick operation : for instance, merging a perspective will just change the parent of the sub perspective to the master perpective stack and its content will be the same. Only binding to existing objects should be recomputed (commands, handlers…).

The following array tries to define the different policies for each object found in the sub model and how to merge it in the master model.

How to merge application models

type object policy test for exist operation rebind to do message if not merged

Order	*Object in model*	*Policy for merge*	*Test for exist*	*Merge Operation*	*Child rebinding*	*Message if not merged*
1	Application	Nothing to do	Instance not null in master	None	None	
2	Addon	Add if not exist	elementID	Add addon	None	Addon with ID xxx already exists in master application. Not merged
6	Binding Contexts	Add if not exist	ElementID	Add Binding context	None	Error if two bindings with same ID have a different parent (with different ID)
7	Binding Table	Add if not exist	Element ID	Add binding table	Rebind commands	Error if two binding tables have same ID but different context ID
5	Handlers	Add if not exist	Element ID	Add handler	Rebind command	Error if 2 same handlers have different contents (commands, persisted stated...)
4	Commands	Add if not exist	Element ID	Add command	Rebind category	Error if 2 same cmds have different contents
3	Commands Category	Add if not exist	Element ID			

To deal with the composition dependencies, objets must be merged in this order :

How to merge application models

Application, Addons, Commands categories, Commands, Handlers, Binding Context, Binding Table

3 OSGI CONSIDERATIONS WHEN A MODEL IS USED AS A FRAGMENT :

When a model is used as a fragment some OSGi conditions must be checked

3.1. Master plugin must have access to sub models

When the main model is created its plugin must define the relation to its sub models. It can be done by hand if the main eap plugin is defined.

In the case of the legacy model application it can be a tricky issue because there is no access to the legacy model. May be a solution would be to have its own master model which depends on the legacy and the other features...

3.2. Packages containing bundleclasses :: must be exported

If a model uses a class with the bundleclass :: annotation, it must export the package where this class stands.