

COMET - Requirements model to service architecture model transformation

<!-- -->

Table of contents

1 Requirements model to architecture model transformation.....	2
1.1 Transformation specification.....	2

1. Requirements model to architecture model transformation

We see a possibility of transforming the requirements model to a first draft of the architecture model, for example by generating a UserService for each actor (which we see is quite common the case). However, the transformation is a one-way transformation from requirement model to architecture model. It will constrain the models unnecessarily, especially the requirement model, to maintain a round trip transformation between the requirements model and the architecture model.

The following transformations are possible:

1. Map each actor to a UserService and create an Interface as a provided interface of the actor. For each of the use cases that the actor relates to add a corresponding operation to the Interface. Naming conventions: Use the Actor name to name the UserService and the same for the interface prefixed with an "I". The operations should be named according to the use cases.
2. Map each <<Manage>> use case to a BusinessService providing CRUD operations as well as find and collection operations for the Resource(s) that are related to the <<Manage>> use case. Naming conventions: use the resource name prefixed with "Manage"

Possible way to handle extends and includes relationships:

1. Include -> reuseable UserService with interface providing the "include" service or an operation in the interface of the extended UserService.
2. Extends-> Operation in the interface of the extended UserService

1.1. Transformation specification

MOF have defined a metamodel for modeltransformations below we have specified the transformation rules for the Actor to UserService mapping based on this scheme. Figure 1 describes a simple transformation from a UML use case model to an architecture model with UserService components (classifiers) with interfaces.

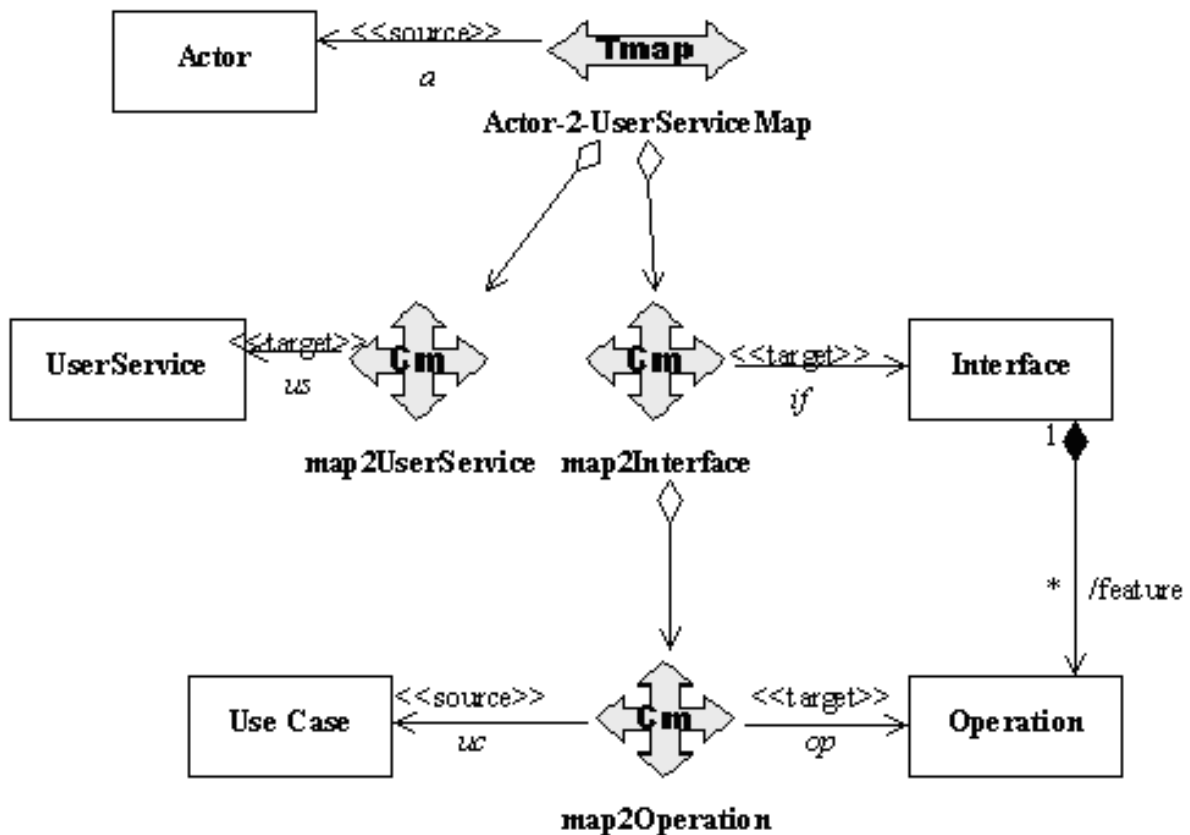


Figure 1: Example – from use case models to architecture

The transformation map ‘Actor-2-UserServiceMap’ has one type of source object, an Actor. This signifies that the extent of actors in the source model is input for this transformation (one at a time). Source and target objects are modelled in terms of associations to types from the source and target metamodels, respectively. Source and target associations can be named, which signify named references to the source/target objects. A target association signifies the creation of one instance of the target type. From the example, the target ‘UserService’ signifies a creation of a UserService object. The references to these objects have global scope and can be used in, for example, FeatureMaps in the specification.

The classifier maps are either part of the transformation map or other classifier maps, defined through aggregate associations. An association defines a transformation path that implicitly (or explicitly, through OCL statements) carries source objects. When no constraint is specified, the sources of the owner become sources for the contained element (classifier map).

Figure 2 depicts a detailing of the transformation, which includes simple feature maps. A feature map is modelled as an attribute and may contain assignments or action specifications.

An assignment typically maps basic properties of the source, to basic properties of the target, e.g. their names. An action specification is a more complex operation that e.g. adds a realisation dependency to a target object.

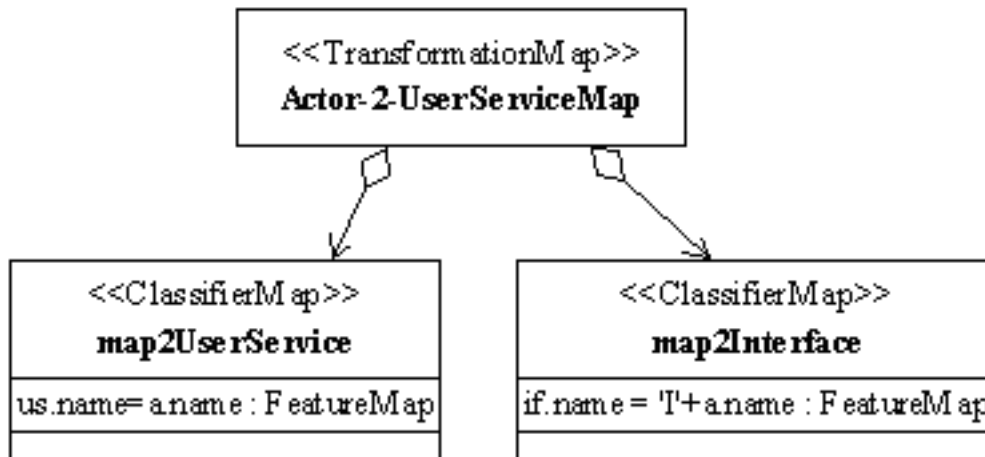


Figure 2: Example – feature map

OCL constraints are used to constrain the input domain for a transformation class, based on the context of the owner. Typically, these define selection criteria for the set of objects to be handled by the receiving transformation class. OCL constraints are assigned to the associations between transformation classes. In the example in Figure 3, the OCL constraint on the association selects all use cases an actor communicates with.

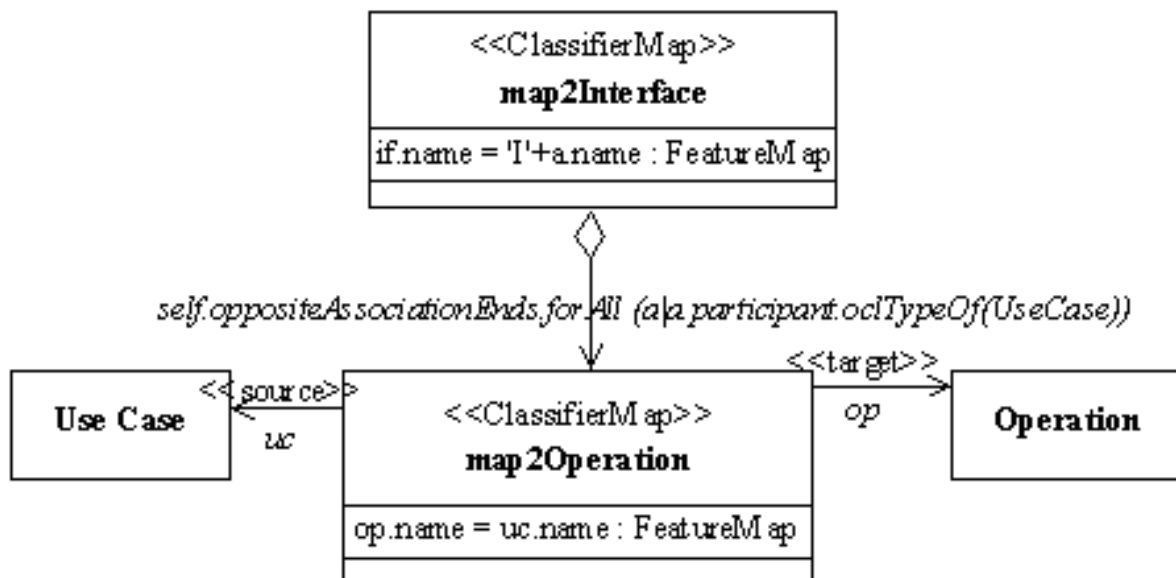


Figure 3: Example – OCL constraints

In addition to transforming simple properties within FeatureMaps, more complex operations are needed to transform relational features, like adding an operation to an interface. Two different methods are used to support this. The first one is implicitly performed, by adding properties by referring relations from the metamodel, as in Figure 4.

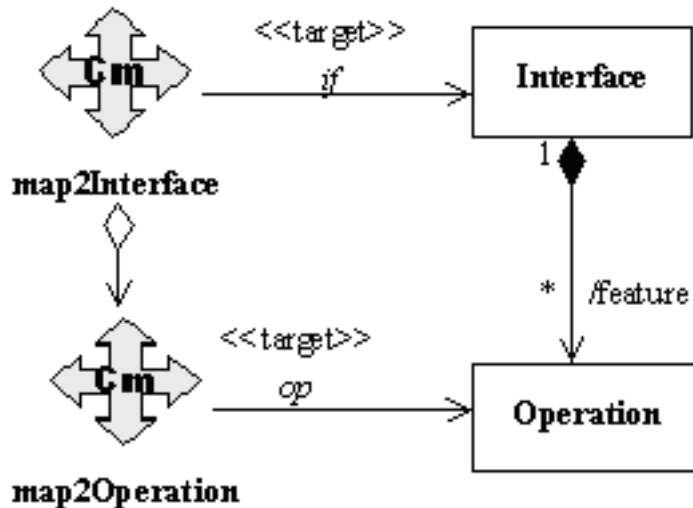


Figure 4: Referring to metamodel relations

The second way is to specify feature maps that add a relationship to the target object, as shown in Figure 5.

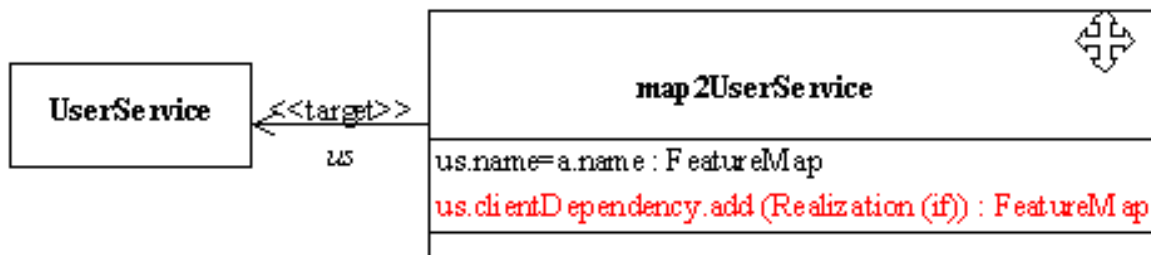


Figure 5: Adding properties in a FeatureMap

The feature map accesses the properties of the target object and imposes an addition of a feature, e.g. a new realisation, as in the example. The add operation is assumed to be handled by the target object.