Construction of Models Needs Idempotent Transformations

Clementine NEMO, Mireille BLAY-FORNARINO
Universite de Nice - Sophia Antipolis
Laboratoire I3S (CNRS-UNSA), Equipe Modalis
Batiment Polytech'Sophia - Dept. SI, 930 route des Colles
B.P. 145 F-06903 Sophia Antipolis Cedex
(nemo,blay)@polytech.unice.fr

I. Challenge

Model transformations play a critical role in Model Driven Development because they automate recurrent software development tasks. Some of these transformations are refinement of models by adding or retracting elements to produce new models conforming to additional constraints. For example, such transformations are used to integrate non functional properties. But modifications of the resulting model can break the conformity to those functional properties. Our challenge is to detect and restore this conformity applying the same transformation again. In this paper, we defend that model transformation is the key concept to (i) validate and (ii) restore models and we establish a system to define idempotent transformations.

II. Guideline Example

Figure 1 depicts a component diagram. On the initial model (m0) the component `Server` provides an interface `ManageData` with an operation `addData`.

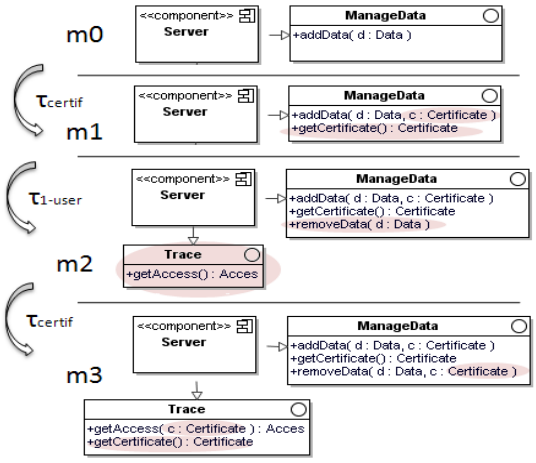

Fig. 1. Steps to restore a model by re-application:
1) Initial model (*m0*) is transformed by applying transformation T_{certif}.
2) The user modifies the resulting model *m1* by adding a *Trace* interface.
3) The transformation T_{certif} is applied again. It adds a *certificate* parameter to *getAccess* operation.

To improve the model m0, we add a security concern. We include it to certify the access to the `Server`. So we apply a model transformation T_{certif} defined by the following constraint C_{certif}: all operations of all provided interfaces of the *secured component* have a `Certificate` parameter as input. The application of the transformation T_{certif} adds or retracts elements to create a model that verifies the constraint C_{certif}, *i.e.* the model conforms to T_{certif}.

In order to apply T_{certif} on m0 the role *secured component* [1], used in the constraint expression, is bound to the `Server` component of the model m0. It ensures that (a) all operations of all interfaces provided by the *secured* component require a *certificate* parameter and (b) each provided interface defines a *getCertificate* operation. The result of the application is depicted o n the model m1 and the modifications are red highlighted.

III. Application and re-application of transformations

A. Motivation

In our example, the transformation T_{certif} aims to build models supporting certification process. All models resulting from the T_{certif}'s application satisfy the constraint C_{certif}.

However model evolutions can lead to constraint violations, *e.g.* adding an interface `Trace` with an operation `getAccess` without the `Certificate` input parameter (result on the model m2) breaks the conformity of m1 to T_{certif}.

Expressing these constraints using OCL is possible but it will not automatically correct the model. The *Re-application* of a transformation should be a way to restore the conformity of the model to a given transformation [2]. Re-applying the transformation T_{certif} must only add the needed parameter and operation. The modifications of the transformation re-applications are red highlighted on the model m3.

Nevertheless not all transformations can be applied several times on a same model without an unexpected side-effect, *e.g.* adding a `Certificate` input parameter or a `getCertificate` operation even if they already exist, or adding a `Certificate` input parameter to the operation `getCertificate`. Only transformations with the idempotent property can be safely applied several times to a model.

B. Idempotency, the hidden property

Some approaches support several applications of the same transformation on a model, but they do not associate the idempotent property to the transformations themselves. The QVT transformation language [3] guarantees to update the necessary elements by the use of the *Check-Before-Enforce* mode. Other transformation languages limit the number of applications of a transformation rules [4], they specify the direction of the application [2], propose operators of composition to factorize the same actions [5] or maintain the target context by graph analysis [6]. AGG supports rules with negative application conditions often used to avoid the creation of duplicated elements [7].

The common point between these approaches is support the definition of transformations that can be applied several times. The developer is in charge of ensuring that the transformations have the idempotent property. Whereas the writing of the initial transformations is easy, the addition of conditions to return them idempotent is complex and error-prone. Hence we propose to define transformations in a way that they automatically have the idempotent property, facilitating by this means the design of transformations and enforcing the role of transformations in model-driven software development.

IV. Idempotent Transformations

A. Definition and application of transformations

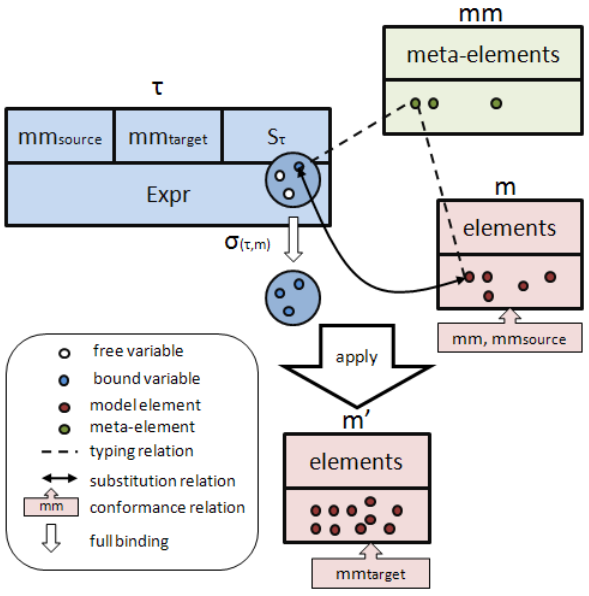

Fig. 2. Transformation application : $\tau_\sigma(m) = m'$

a) *A transformation* τ is defined by a quadruplet (mm_{source},mm_{target},$Expr$,S_τ). mm_{source} and mm_{target} are the metamodels which define the spaces of the transformations, $Expr$ is the transformation expression, and S_τ is a set of free variables of $Expr$. These variables are used to bind the transformation to the model. In the guideline example, the set of free variables refers to the role *secured component*.

b) *The conditions to apply* a transformation τ on a model m are: *(condition 1)* the model has to conform to the metamodel source ($m \in mm_{source}$) and *(condition 2)* all the variables of S_τ have to be bound to an element of the model m according to their typing [8]. The binding is a substitution $\sigma_{(\tau,m)} = \{(var, e) | var \in S_\tau, e \in m\}$. We note $\tau_\sigma(m)$ the application of the transformation on the model m according to the substitution σ.

c) *The conformity of a model to a transformation* results from the application of the transformation. The model must satisfy the constraints defined by the target metamodel ($m' \in MM_{target}$). We say that m' *conforms to the transformation* T.

Figure 2 depicts these definitions graphically.

B. Definition and properties of idempotent transformations

According to a given substitution $\sigma_{(\tau,m)}$, a transformation τ can be re-applied on the resulting model $m' = \tau_\sigma(m)$ if :

1) the transformation is *quasi-endogenous*, *i.e.* it preserves the conformity to the source metamodel, *i.e.* $mm_{target} \subseteq mm_{source}$[1].
2) the transformation is *substitution-preserving*, *i.e.* $\forall (var, e) \in \sigma_{(\tau,m)} \Rightarrow e \in m'$

Transformations satisfying these properties are idempotent if they conform to the following definition.

Definition (Idempotent transformation) Let τ a *quasi-endogen* and *substitution-preserving* transformation.
The transformation τ(mm_{source},mm_{target},S_τ) is idempotent if: $\forall m \in mm_{source}, \tau(m) = m' \Rightarrow \tau(m') = m'$

Our goal is to support the definition of idempotent transformations in a transparent way for the user. The user only specifies actions to transform the initial model in a model conforming to the target metamodel. The user can then modify this model and apply the same transformation several times on that model to restore it. If any modification is performed, the model conforms to the transformation. // We call such transformations *IT*. We now define them.

C. IT : Idempotent Transformation by construction

An *IT* is a transformation whose $Expr$ is composed of three parts : a *selection part* that selects elements, an *identification part* that identifies the elements to be created, and a *modification part* that modifies the model.

The selection part: includes the elements necessary to modify the model. It is defined as a sequence of *selection actions*. Each selection action is idempotent, *i.e.* in the same context, the same elements are selected. These actions are only based on positive literals such as existence of an element. Moreover, the selection actions do not select the elements created or valuate by the transformation application itself, *e.g.* in the guideline example, the transformation T_{certif} will never

[1] $\forall$me$\in mm_{target}$, me$\in mm_{source}$

select the *getCertificate* operation because it was created by this transformation application. We note $select_{\tau_\sigma}(m)$ the set of selected elements by $\tau_\sigma(m)$.

The selection part does not check if the model conforms to the source metamodel. This conformity is supposed to be checked before to apply the transformation. When the set of selected elements is empty, it only means the transformation has nothing to do. It means the model conforms to the transformation.

The identification part: points out in a unique way each element (*e.g.* for a component, its name; for a reference, its name, the identifiers of the source and target). We note $ident_{\tau_\sigma}(m)$ the set of identifiers for $\tau_\sigma(m)$. The identification actions return identifiers according to the elements selected by selection actions. Identifiers do not depend on the number of selected elements.

The modification part: is a sequence of idempotent *basic actions* closed by $select_\tau(m)$, $ident_\tau(m)$ and $\sigma_{(\tau,m)}$ (e.g creating an element, setting a value, deleting an element). These actions do not depend on the number of selected elements and identifiers.

D. Why ITs are idempotent ?

*IT*s are idempotent if the basic actions are executed on the same subset of elements. Let us demonstrate that : $select_\tau(m') \subseteq select_\tau(m)$.

Let τ an *IT*, let m a model like $m \in mm_{source}$, and let $\sigma_{(\tau,m)}$ a binding between τ like $\tau_\sigma(m) = m'$ and $select_\tau(m) = \{e | e \in m\}$.

By contradiction we suppose that $\exists e' \in select_\tau(m'), e' \notin select_\tau(m)$. It means that:

- e' is selected because is created by $\tau(m)$. It is a contradiction because in the definition of the selection part, the created element are not selected. So $e' \in m$.
- e' is selected according to new elements or to updates values. For the same reason, this is a contradiction because of the definition of the selection part. So $e' \in m$.
- e' is selected according to the non existence of an element. It is a contradiction, by definition, because we forbid to select element according to negative literals.

Consequently there is no way to select an element that was not in $select_\tau(m)$. However since elements could have been deleted there are elements in $select_\tau(m)$ that are no more in $select_\tau(m')$. So we proved that $select_\tau(m') \subseteq select_\tau(m)$.

Since identifiers and basic actions are computed according to the selected elements therefore $ident_\tau(m') \subseteq ident_\tau(m)$ and $action_\tau(m') \subseteq action_\tau(m)$. Since all the actions are idempotent and executed on the same elements (no action depends on the element number) therefore they do not modify the initial model.

We now illustrate our approach with a system supporting definition of *IT*.

V. A SYSTEM TO WRITE IDEMPOTENT TRANSFORMATIONS

We define a model as a set of named and typed elements. A value can be associated with some elements. An element is identified in a unique way (by the identification action) in the model. The applications of transformations are identified

```
element(server, component, server, ap0).
element(manageData_ap0, interface, manageData, ap0).
element(server_provides_manageData_ap0, reference,
               provides, ap0).
hasForValue(server_provides_manageData_ap0,,
              [server, manageData_ap0], ap0).
```

Fig. 3. Partial Prolog definition of the model m0 in figure 1

and the identifier is associated to each element, value or meta added. This system has been implemented in Prolog (cf. Fig 3).[2]

The actions to select elements are based on : the element identifier, the type, the name, and the application identifier. We can find all the elements with a given name, or typed by a given type and so on. These selection actions will not select an element which was created or assigned by a previous application of the transformation (cf. Fig 4).

```
existElem(Id,Meta,Name,Ap,NotAp) :-
    element(Id,Meta,Name,Ap),
    (NotAp == null ; Ap \== NotAp).
```

Fig. 4. A selection action to select elements

The basic actions to modify a model are : *creation* of a new element with a given identifier, type and name; *set a value* to a given element; *add a meta* to a given element; *delete* a given element. Each of these actions are executed according to a given transformation application identifier.

```
trCertif([Secured],Ap) :-
 doForAll(Ap,
   [existReference(RefId,provides),
    directlyLinked(RefId,Secured,Interface),
    existReference(RefOp,operation),
    directlyLinked(RefOp,Interface,Operation) ],
   [getParameterID(Operation,c,IdPar),
    getReferenceID([Operation,parameter,IdPar],IdRef),
    getReferenceID([IdPar,type],IdType) ],
   [ createParameter(IdPar,IdRef,IdType,Operation,
                                      c,certificate)]).
trCertif([Secured],Ap) :-
 doForAll(Ap,
   [existReference(RefId,provides),
    directlyLinked(RefId,Secured,Interface)
   ],
   [
    ...
    ],
   [
    createOperation(IdGetCertificate,
          IdRefGetCertificateInterface2Operation,
          Interface,getAccess),
    createParameter(IdPar,IdRef,IdType,IdGetCertificate,
          return,certificate)]).
```

Fig. 5. Partial definition of *IT* corresponding to T_{Certif} in figure 1

[2]Source code is available at : http://modalis.polytech.unice.fr/~clementine/PHD/policies.xhtml

In order to facilitate the design of a component model, we defined a set of more elaborated actions using the basic actions. The following *IT* (cf. fig 5) uses these composed actions and describes the transformation T_{certif} of the guideline example (cf. Fig. 1).

Independently of the transformation language, if the aim is to re-apply the T_{Certif}, the transformation implementation has to express additional information, such as :

- Select operations defined in provided interfaces but not the operation *corresponding* to *getCertificate*,
- Do not create a parameter whose the name is *c* if it already exists,
- Do not create an operation *getCertificate* if it's already defined.

The advantage to the user is noticeable: less effort and less number of errors. This complexity is tackled in the implementation of the actions and the engine to execute ITs. We are now able to apply several times the same transformation on a model and restore it.

VI. Conclusion

Re-application of transformation restores a model if the transformation has the idempotency property. In this paper we define a system to easily design idempotent transformations by action composition. Now, our challenge is to construct a model conforming to a set of transformations. We use idempotent property of ITs to apply them several times. This process is repeated until a fixed point is reached (the model is not modified any more by transformation applications) or a cycle is detected (same sequence of modification actions).

References

[1] R. France, K. Dae-Kyoo, S. Eunjee, and S. Ghost, *Using Roles to Characterize Model Families*, 2003.

[2] P. Stevens, *Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions*, Jan. 2010.

[3] OMG, *MOF QVT Final Adopted Specification, OMG Document ptc/2005-11-01*, Object Modeling Group, Jun. 2005. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/2005-11-01

[4] A. Lajmi, S. Cauvin, M. Ziane, and T. Ziadi, "A Multi-View Model-Driven Approach for Packaging Software Components," in *25th Annual Symposium on Applied Computing(SAC 2010)*. ACM, Mar. 2010.

[5] J. Sanchez and J. Garcia, *Approaches for Model Transformation Reuse: Factorization and Composition*. Springer, 2008.

[6] I. Rath, G. Bergmann, A. Okros, and D. Varro, "Incremental Pattern Matching in the VIATRA Model Transformation System," in *International Conference on Model Transformation(ICMT'08)*. Springer, Jul. 2008.

[7] T. Mens, G. Kniesel, and O. Runge, "Transformation dependency analysis - A comparison of two approaches," in *Langages et Modèles à Objets(LMO2006)*, Mar. 2006.

[8] J. Steel and J.-M. Jézéquel, "Model Typing for Improving Reuse in Model-Driven Engineering," in *International Conference on Model Driven Engineering Languages and Systems(MODELS/UML)*, ser. LNCS, vol. 3713. ACM/IEE, Oct. 2005, pp. 84–96.