Combining Preoccupations with Models

Jean Bézivin, Marcos Didonet Del Fabro, Frédéric Jouault, Patrick Valduriez

ATLAS Group, INRIA and LINA - University of Nantes
2, rue de la Houssinière, BP 92208 - 44322 - Nantes cedex 3, France
{jean.bezivin, marcos.didonet-del-fabro, frederic.jouault}@univ-nantes.fr, patrick.valduriez@inria.fr

1 Introduction

Information systems are composed by complex and interrelated preoccupations. Each preoccupation should be created independently from each other to handle with a single aspect. Once created, these preoccupations must be combined to form the complete information system.

Model Driven Engineering (MDE) and Aspect Oriented Programming (AOP) are two ways to achieve separation of preoccupations in information systems. Separation of preoccupations in AOP is well documented [2]. We propose here to study separate handling of preoccupations in MDE. Each preoccupation is captured by a model. This is expressed by the *repOf* relation. The precise nature of the model is expressed by a metamodel. We say that a model conforms to a metamodel.

Problem statement: Given two models *Ma* and *Mb* capturing different preoccupations of a system, how to combine them into a new model *Mab,* conforming to a new metamodel *MMab.*

Analyzing primitive operators over models, we verified that it is not possible to have a single operator ⊕ capable of combining any kind of models, such that *Mab = Ma* ⊕ *Mb*. For example union and intersection operators are purely syntactic. They do take into account different combination semantics, for instance how to concatenate elements from different models or how to order combined elements.

In this paper we present a functional notation for combining different preoccupations. We define a variable combination function capable of representing different combination semantics. We propose how to apply this function and related elements inside a model driven environment. We validate this approach using it in our prototype called ATLAS Model Weaver (AMW).

2 Combining Preoccupations

Consider the combination of two different preoccupations in a system. These preoccupations are represented respectively by a set of elements *Pa* and *Pb*. A combination function, denoted by *CF*, defines how *Pa* and *Pb* are combined to form a new set *Pab.* We write:

CF: Pa X Pb → Pab **(1)**

We may also write: *Pab = CF (Pa, Pb)*. *Pa* and *Pb* are the function arguments. *Pab* is the result and *CF* is the function definition. Every set in the function *CF* must have a type. We write the type equation as follows:

CF: typePa X typePb → typePab.

The function also has a type. It is defined by the entire type equation. The type of *CF* is *(typePa X typePb → typePab)*. *CF* contains the function definition (also said function body). The function definition follows specific rules, called the function definition type, or *typeCF*.

However a unique function *CF* is not capable of computing different combination semantics. We initially add a parameter *Cs* into the combination function. The set *Cs* contains elements defining correspondences between elements from *Pa* and *Pb.* The combination semantics of these correspondences are specified in *typeCs*. We write:

CF: Cs X Pa X Pb → Pab **(2)**

Pab = CF (Cs, Pa, Pb)

The type equation is:

CF: typeCs X typePa X typePb → typePab

The parameter *Cs* has a different purpose then *Pa* and *Pb*. The function definition *CF* must compute every different semantic defined in *Cs*. But in this case it is also passed as parameter in every function execution. We avoid this by currying *Cs* into a function *CF'*. We write:

CF': Cs → CF **(3)**

CF = CF' (Cs)

For the type equation we have:

CF': typeCs → typePa X typePb → typePab. The definition of *CF'* is specified in *typeCF'*.

Note that *CF'* returns a *function* that takes *Pa* and *Pb* as parameters, e.g., *CF*. This way we have a different combination function *CF* for every different combination semantics specified in *Cs*:

CF: (CF': Cs → (Pa X Pb → Pab)) **(4)**

Pab = (CF' (Cs)) (Pa, Mb)

To create *Cs* there is a matching function *Match* specifying how the elements from *Pa* are put into correspondence with the elements from *Mb.* The function body is defined over *Pa* and *Pb* types.

Match: typePa X typePb → Cs **(5)**

Cs = Match (typePa, typePb)

Match: typeOftypePa X typeOftypePb → typeCs.

The combination function is symmetric, meaning that neither *Pa* nor *Pb* have the specific status of models, like in existing aspect oriented solutions. It depends on the semantics captured by *typeCs*. A combination is not commutative, e.g., combining *Pa* and *Pb* may be different from combining *Mb* and *Ma*.

3 Combining Preoccupations as Models

A model is a set of elements and associations. The structure of a model, e.g., how these elements and

associations are organized are defined in a metamodel. It acts like a typing system. A model conforms to a metamodel. Models are transformed into other models using transformations. A transformation is itself a model. It conforms to a transformation metamodel.

In our solution the preoccupations are represented by models. We have models *Ma* and *Mb*. The preoccupation types are the preoccupation metamodels *MMa* and *MMb*. The combined preoccupation is represented by a model *Mab*, conforming to metamodel *MMab*. The body of the combination function *CF* is represented by a transformation model *Mt*. It conforms to *MMt*.

We define a metamodel capturing the semantics for combining models, (corresponds to *typeCs)*, called *weaving metamodel*. The weaving metamodel is minimal, capable of representing correspondences and links between model elements, which we argue as being the common structures for many combination scenarios. It is further extended with different elements according to application needs. This allows capturing complex composition semantics.

A weaving model *Mc* (corresponding to *Cs*) is created to link metamodel elements, for instance from *MMa* and *MMb*. It is passed as parameter to a curried transformation model *Mt'*. It conforms to *MMt'*. We have one *Mt*' for each weaving metamodel. For every model *Mc* passed as parameter, it produces a transformation model *Mt*. It conforms to *MMt,* which may be equal to *MMt'*. The difference between *MMt* and *MMt'* allows generating transformations in different languages, for instance in ATL [6] or XSLT. The transformation takes the models *Ma* and *Mb* as parameters to perform the combination.

We implemented a Model Weaver prototype that coupled with the ATL transformation engine is capable of combining models. It allows having extensible weaving metamodels. The user interface adapts to these different weaving metamodels. It is implemented using Eclipse EMF API [3]. In the screen shot of the Model Weaver in Figure 1 there are three panels: a left metamodel, a weaving model and a right model, respectively. In this case it weaves geographical information with elections results. The example is explained in details in [4].

4 Related Work

C-SAW [1] is an aspect oriented model weaving solution. It defines an embedded constraint language which inserts the preoccupations from an aspect model into a primary model containing the business logic. It defines a set of combination operations such as inheritance and implementation of interfaces. It does not separate the operation definition and execution like in our solution. A merge operator is defined in [5] for combining models in a general way. It could be used as a basis for implementing generic model combinations.

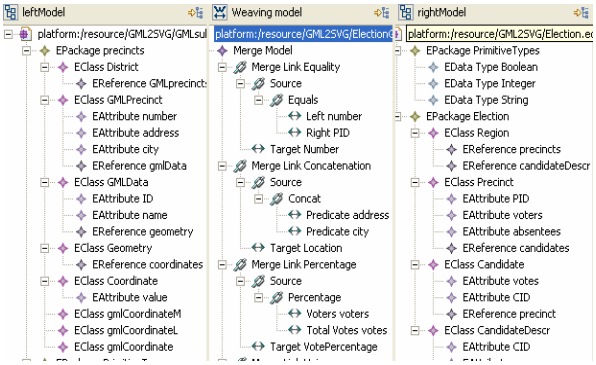

Fig 1. The AMW prototype

5 Conclusion

In this paper we have proposed a model driven solution for combining preoccupations by the means of combination functions. The preoccupations are represented by models; the combination functions are considered models as well. Reifying functions allows adding variability to it, with different weaving metamodels specifying the combination semantics. We have implemented a prototype to represent the weaving model and metamodel. The weaving prototype coupled with a transformation facility enabled us to validate the ideas proposed above.

As future work different weaving metamodels will be specified to handle different combination scenarios. Different matching functions as well, enabling to create weaving models semi-automatically.

6 Acknowledgments

This work was partially supported by a grant from Microsoft Research, Cambridge, UK, and ModelWare. IST European project 511731. The prototype was built with the help of Erwan Breton and Guillaume Gueltas as part of collaboration with the Sodifrance Company.

References

[1] Zhang J., Gray J., Lin Y., A Model-Driven Approach to Enforce Crosscutting Assertion Checking First International Workshop on the Modeling and Analysis of Concerns in Software (MACS), held at ICSE, St. Louis, MO, May 2005.

[2] Aspect Oriented Software Development community. http://aosd.net.

[3] Eclipse Modeling Framework - www.eclipse.org

[4] Bézivin J., Didonet Del Fabro M., Jouault F., Valduriez P. Generic Data Mapping using Model Weaving, 2005 (submitted for publication)

[5] Pottinger R., Bernstein P., Merging Models Based on Given Correspondences. VLDB 2003

[6] ATLAS Transformation Language. Reference site: http://www.eclipse.org/gmt (May 2005)