Model Integration with Model Weaving:

a Case Study in System Architecture

Albin Jossic[1] Marcos Didonet Del Fabro[2] Jean-Philippe Lerat[1]
Jean Bézivin[2] Frédéric Jouault[2]

[1]Sodius SAS
6, rue de Cornouaille,
BP 91941, 44319 Nantes Cedex 3 France
{ajossic, jplerat}@sodius.com

[2]ATLAS Group, INRIA & LINA
2, rue de la Houssinière,
BP 92208, 44322 Nantes Cedex 3 France
{marcos.didonet-del-fabro, jean.bezivin,
frederic.jouault}@univ-nantes.fr

Abstract

Complex metamodels are often decomposed into several views, in particular for architecture framework metamodels such as DoDAF (Department of Defense Architecture Framework). Designing models that conform to this kind of metamodels implies data integration problems between the different views. Usually, different views from a same metamodel share a common core. This is the reason why those views are interrelated. The common core is identified with the creation of mapping links.. Within MDE (Model Driven Engineering) approaches, these links may be captured by weaving models. Using MDE principles, we automate this data integration process by generating transformations between these views. For this, we generate weaving models between different views using matching heuristics and then we produce model transformations from this weaving model. We have applied this method to DoDAF metamodels. In this architecture framework, SV-5 (a system view) relates to OV-5 (an operational view) and SV-4 (another system view) with the aim to provide a traceability matrix about system architectures.

1 Introduction

System Architecture (SA) is becoming an important concern in the management of complex computer systems. A general definition of SA may be found in [1]. Many frameworks have been proposed for dealing with SA. One of them is DoDAF (Department of Defense Architecture Framework) [2] [3]: DoDAF is a framework for the development of system architectures for military or enterprise organizations. This framework enables to specify these architectures following three different views, called Operational Views (OV), System View (SV), and Technical Standards View (TV). There are different relationships between the concepts of these views. For example, the Traceability Matrix in SV-5 (a subset of SV) is used to capture the relationships between the Operation Activities contained in OV-5 (a subset of OV) and System Functions in SV-4 (another subset of SV). Each one of these views is constructed conforming to a common set of architecture data entities and relationships between these entities. This data model is called DoD Core Architecture Data Model (CADM). Since they are created using similar data entities (concepts), there are many common entities between these different views.

In this paper, we study how to generate the traceability matrix that contains the overlapping concepts between these different views, most particularly from OV-5 and SV-4 views into SV-5 view. All these views come from architecture descriptions defined within DoDAF. This traceability matrix can be generated by manually coding model transformations. However, the OV-5 and SV-4 metamodels contain several elements and this task may be quite complex. It should be possible to automate this process as much as possible.

We propose a model driven engineering (MDE) approach that automates the process of defining the traceability matrix. We concentrate on the automation of the data integration from a model to another one. This process concerns the entities that are the same and also their common properties and relationships in the different metamodels.

Our solution is divided into different steps. First, we propose to represent the different views (OV-5, SV-4 and SV-5) in KM3 metametamodel [4]. Second, we capture the relationships (i.e., mapping links) between these views using a weaving model [5]. A weaving model is a model that specifies different kinds of mappings between metamodel elements. The weaving model is created by a set of transformations that are executed sequentially. Each

transformation calculates similarity values between the concepts of the views, based on different matching heuristics. Last, we intend to implement an ATL [6] transformation from SV-5 to itself, to calculate the traceability matrix. This is a future work.

This paper is organized as follows. Section 2 describes our case study. Section 3 presents our solution to automate the data integration process. Section 4 concludes the paper.

2 Case Study

In our case study, we choose views OV-5 and SV-5, SV-4 and SV-5 in the DoDAF context. DoDAF [3] is an architecture framework designed by the United States Department of Defense. DoDAF framework supersedes the C4ISR (Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance) Architecture Framework. There are existing derivative frameworks based on DoDAF like MoDAF (United Kingdom Ministry of Defense) or NATO-AF (North Atlantic Treaty Organization). First, we introduce the possible solutions and then we explain the intersection between DoDAF views.

2.1 Possible solutions

We have distinguished two choices to generate a traceability matrix in a SV-5 view for DoDAF architecture descriptions. The first possibility is to implement a transformation from the OV-5 and SV-4 metamodels to the SV-5 metamodel. This transformation translates the data from the input models (OV-5 and SV-4 views) into the output model that conforms to SV-5 metamodel and that calculates the traceability matrix. In this case, we must code the transformation manually. However, creating this transformation by hand is a complex process that should be automated.

The second possibility is to automate the data integration from the input to the output metamodels. In this case, we propose to create a model that captures mapping links between OV-5 and SV-5, SV-4 and SV-5. From there, we are able to generate the two transformations that translate the necessary data into a model that conforms to SV-5. Finally, we implement a transformation from SV-5 to itself with the aim to create the traceability matrix.

2.2 Overlapping parts between DoDAF views

To build the SV-5 traceability matrix for a given architecture description, we need views OV-5 and SV-4. The reason is because the Operational Activity to Systems Function Traceability Matrix (SV-5) contains the relationships between the *Operational Activities* from the Operational Activity Model (OV-5) and the *System Functions* from the Systems Functionality Description (SV-4). Figure 1 illustrates the relationships between DoDAF views. In this class diagram, *System Function* elements inherit from *Process Activity; Task* is another term for *Operational Capability*.

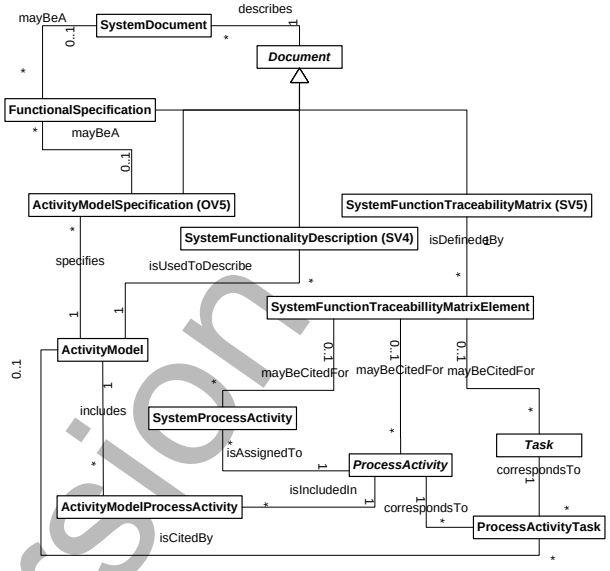

Figure 1: Relationships between DoDAF views

Due to the fact that SV-5 view for an architecture description must be able to capture many parts of the OV-5 and the SV-4 views, we found several common entities and common relationships between these entities. Moreover, according to the inheritance tree, in the three views we found common parent entities, such as *Document*, because all the views on DoDAF metamodel are based on the same core. Another example is the set of elements and their relationships composed of *Activity Model*, *Activity Model Process Activity*, and *Process Activity*. These elements exist in the three DoDAF views.

The three views have an average of 100 entities, and approximately thirty percent of common entities between OV-5 and SV-5, SV-4 and SV-5. The majority of the properties from the common entities are the same, in all the views. This is the reason why we propose to partially automate the data integration of OV-5 and SV-4 into a SV-5 model. After the integration is done, we are able to calculate the traceability matrix into the SV-5 view.

3 Model Integration

We present in more details the steps to automate the integration of DoDAF views. First, we show how to represent DoDAF views with MDE practices. Second, we define the process that creates a weaving model between both metamodels. This process is called *matching*. Last, we

describe how to use the weaving model to generate the transformation that executes the data integration.

3.1 Representing DoDAF views in MDE

We translate OV-5, SV-4, and SV-5 as metamodels for DoDAF architecture descriptions. In MDE practices, a metamodel is a formal definition of a model. A model conforms to a metamodel. With aim to use the AMMA platform [7] [8] to implement weaving models and transformation, we define OV-5, SV-4, and SV-5 view metamodels in KM3 (Kernel MetaMetaModel).

We first define the CADM (Core Architecture Data Model) for each DoDAF view. The CADM defines structured representations of the architecture data elements using the IDEF1X notation [9]. This representation is the data model for DoDAF views with entity relation diagram without composition relationships between data entities. With the aim to give more expressivity for metamodel views, we translate some relationships between the data entities as composition relationships between classes of the KM3 metamodels. For example, we translate the relationship *includes* between an *Activity Model* and its *Process Activities* as a composition relationship. This situation occurs in the three views.

3.2 Matching

The process that defines the mappings between two different metamodels is called *matching*. The matching process is encapsulated in a model management operation called *Match* [10]. A *Match* operation takes two models *Ma* and *Mb* as input and produces a mapping *Map* between the elements of both models as output:

Map = Match (Ma, Mb).

Usually, with the *Match* operation, it is not possible to automatically define exact mappings for every model elements. In this case study, we want to define only equivalence links between entities of both metamodels. Our goal is to generate, as much as possible, the exact mapping between the entities of OV-5 and SV-5, SV-4 and SV-5 metamodels. This mapping is a representation of the intersection between the models.

The *matching* process produces a weaving model that captures a set of mapping links. A mapping link is defined by the element references in their respectively models and a similarity value that characterizes the equivalence between them. To define the exact similarities between entities, we assign a similarity value using a matching technique that sets a similarity value to one (1) if the entities or the structural features in the two metamodels are exactly the same. The default similarity value is zero (0). Finally, we choose all links with a similarity value equal to one. These mappings with exact equivalence are saved in a weaving model. The weaving model conforms to a weaving metamodel that is an extension of a core weaving metamodel [5]. We describe the different steps of the matching operation in the following sections.

3.2.1 Creating weaving models

A weaving model supporting different kind of mappings is created by a model management operation called *CreateWeaving*, which is defined below:

Mw = CreateWeaving (Ma, Mb).

The operation takes the two models as input (*Ma* and *Mb*) and it produces a weaving model as output (*Mw*). We propose to implement this operation using model transformations. Consequently, we may say that the two input models are transformed into one output model that contains the mappings between them.

At this stage, the transformation does not create the exact mappings between the elements of models *Ma* and *Mb*, but it matches all the elements of *Ma* and *Mb* (the Cartesian product *Ma* × *Mb*), and it creates equality mappings for every pair of elements.

The transformation creates a weaving model that conforms to an extension with equality mappings, as illustrated in Figure 2 (in KM3 [4]).

```
class Equal extends Equivalent {
   attribute similarity : Double;
}

class Equivalent extends WLink {
   reference left container subsets end: Element;
   reference right container subsets end: Element;
}

class Element extends WLinkEnd {}
```

Figure 2: Weaving metamodel extension

We show in Figure 3 how a transformation rule written in ATL [6] matches all the classes from a left and a right model (conforming to KM3) and produces an equality mapping between a left and a right element. The *from* part indicates that the transformation matches all the classes of a left model with all the classes of a right model. The *to* part creates the output element, which is an equality mapping conforming to the metamodel extension of Figure 2.

```
rule EqualityMapping {
   from
        left : KM3L!Class, right : KM3R!Class
   to
        anode : AMW!Equal (
          left <- <the left element>
          right <- <the right element>
       )
}
```

Figure 3: Matching rule in ATL

We use an ATL transformation to create the weaving model between the two models. In this transformation, we first generate the Cartesian product between all entities of both models and then we assign the similarity value with the previous algorithm. The Cartesian product contains several imprecise relationships. To refine the Cartesian product, we filter the relationships based on the entities types. In fact, the Cartesian product is done only between elements of the same type (i.e., Class-to-Class, Attribute-to-Attribute, and Reference-to-Reference). This prevents from creating a large model with unnecessary relationships, for example Class-to-Attribute relationships. This transformation takes as input the left and the right models, and the weaving model.

The rule that creates the Cartesian product is illustrated in Figure 4. This rule applies three actions: it creates a new node in the mapping model defined by the concatenation of the left and the right entity names; it saves the concerned entity references from the both input model; and, it sets the similarity value between the left and the right elements.

```
rule PairWise {
   from
      left : KM3L!ModelElement,
      right : KM3R!ModelElement
      (
        (left.oclIsTypeOf(KM3L!Class) and
         right.oclIsTypeOf(KM3R!Class))
       or
        (left.oclIsTypeOf(KM3L!Attribute) and
         right.oclIsTypeOf(KM3R!Attribute))
       or
        (left.oclIsTypeOf(KM3L!Reference) and
         right.oclIsTypeOf(KM3R!Reference))
      )
   to
      anode : prop_g!Node (
         name <- left.name+'_'+right.name,
         model <- thisModule.aModel,
         leftRef <- left,
         rightRef <- right,
         similarity <- left.similarity(right),
      )
}
```

Figure 4: ATL Rule for Cartesian product

3.2.2 Calculating Equivalence

To create the weaving model that captures all the equivalence links between elements, we must calculate the similarity value between each entity of both models. In our case, we have only two values for the similarity. In fact, the similarity is set to one if both elements are the same according to different criteria (explained later). Otherwise, the similarity is set to zero (i.e., default value). We illustrate the similarity algorithm in Figure 5. This algorithm is implemented for all existing types of metamodel elements: *Class*, *Attribute*, and *Reference*.

```
helper context KM3L!ModelElement
   def:similarity(right:KM3R!ModelElement):
   Integer=
   if self.similarityName(right)
   and self.similarityType(right)
   then
      if (self.oclIsTypeOf(KM3L!Attribute) and
          right.oclIsTypeOf(KM3R!Attribute))
      or (self.oclIsTypeOf(KM3L!Reference) and
          right.oclIsTypeOf(KM3R!Reference))
      then
         if self.owner.similarity(right.owner)=1
         and self.similarityUpper(right)
         and self.similarityLower(right)
         then
            1
         else
            0
         endif
      else
         1
      endif
   else
      0
   endif;
```

Figure 5: Similarity algorithm

In the following section, we explain in more details the similarity algorithm for Class-to-Class and Structural Feature-to-Structural Feature mapping. In particular, we define characteristics to specify the equivalence link for each entity type.

3.2.2.1 Class-to-Class equivalence

According to the similarity algorithm (Figure 5), Class-to-Class similarities are calculated taking one element from the left metamodel and one element from the right metamodel and analyzing their internal features. We apply two methods to calculate the similarities:

- *String similarity*: the names of the model elements are considered as *Strings*. The names are compared using normal string comparison. This is implemented in *similarityName* helper.

- *Type similarity*: the types of the model elements are extracted with the OCL function *oclType* and are compared. This is implemented in *similarityType* helper.

There are other methods that can be used to calculate the similarities. For example, it is possible to check if the containing elements of the classes from the left and the right models are equivalent. This way, the children elements are also equivalent. Another possible method is to compare the similarity of the inheritance trees of these elements.

The heuristics described determine the strength of the equivalence concept that we want to apply (it ranges from zero to one [0-1]). The mapping link is considered as equivalent only if all conditions are satisfied. In this case the similarity value is set to one.

3.2.2.2 StructuralFeature-to-StructuralFeature equivalence

A structural feature is an internal feature of a class. It can be an attribute or a reference. Structural Feature-to-Structural Feature similarities are calculated taking one structural feature from the left metamodel and another one from the right metamodel. In fact, we treat only Attribute-to-Attribute and Reference-to-Reference elements because we choose not to calculate equivalence value for elements with different types. To calculate the equivalence between two structural features, we reuse the similarity algorithm (Figure 5) and we add three specific methods:

- *Owner Entity similarity*: owner classes of the given structural features are compared using the similarity algorithm.
- *Structural Feature Type similarity*: the types of the model elements given by the structural features are compared using the similarity algorithm.
- *Cardinality similarity*: the *upper* and the *lower* cardinality of the structural features are compared. Cardinalities are the same only if the upper and the lower cardinalities are the same in both structural features. This is implemented in *similarityUpper* and *similarityLower* function.

To calculate Structural Feature-to-Structural Feature similarities, we take advantage of the knowledge about the KM3 metametamodel [4]. For instance, we take into account the cardinalities and the referenced types.

3.2.3 Filtering the result

The weaving model contains a set of links with many similarity values. However, this is not the final weaving model. It is necessary to select only the links with the highest similarity values. We filter the weaving model using another transformation that chooses the equivalence mappings. In this operation, we are only interested in the mappings with a similarity value equal to one. This transformation takes as input the previous weaving model and produces another weaving model without the links with lower similarity values. This allows to keep only the relevant mapping links.

3.3 Generating transformations

In this step, we use the weaving model to generate the transformations that implement the data integration between metamodels. According to our motivating example, we produce the transformations from OV-5 to SV-5 and from SV-4 to SV-5.

We have implemented an ATL transformation to generate these transformation models. This transformation takes as input the following models: the left model, the right model, and the weaving model created by the filtering process. This transformation produces as output a transformation model that conforms to the ATL metamodel. Finally, the transformation model is extracted into an ATL file.

In the following schema (Figure 6), we show that *AMW2ATL* transformation produces *MML2MMR* transformation from the left metamodel *MML*, the right metamodel *MMR* and the weaving model *MW*. *AMW2ATL* is a Higher Order Transformation (HOT). A HOT is a transformation that takes transformations as input and/or produces transformations.

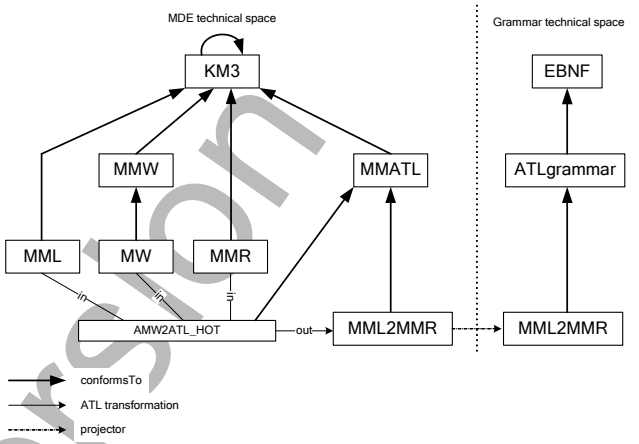

Figure 6: Producing MML2MMR Transformation

In our case, *MML* is OV-5 or SV-4 metamodels and *MMR* is the SV-5 metamodel. From this, we generate OV-5 to SV-5 and SV-4 to SV-5 transformations, which are implementations of data integration from OV-5 and SV-4 to SV-5 metamodels. To reverse the data integration process, it is necessary to invert *MML* and *MMR* metamodels. According to our example, we can also produce SV-5 to OV-5 and SV-5 to SV-4 data integration.

4 Conclusions

In this paper, we have presented a practical MDE-based solution to automate the integration of different models. We used DoDAF as case study. In our solution, we produced a weaving model between metamodels using ATL transformations. This weaving model represents equivalence mapping links between the metamodels' elements. The similarity algorithm that calculates the equivalence mappings can be improved by using other criteria or can be made more permissive. From this step, we are able to produce the transformation model that implements the data integration from the left to the right metamodels. This transformation is generated based on a weaving model. In the last step, we can export the transformation model into an ATL code source.

This process is particularly adapted for complex metamodels composed of several views which are based on

a same core. In fact, it is possible to exchange data between models that conforms to two different parts with a generated transformation between them. This exchange concerns only the exactly common part, in other words the metamodels intersection.

As future work, we plan to implement a transformation that produces the traceability matrix from a SV-5 view. After the data integration, this SV-5 model about an architecture description contains all the Operational Activities of the OV-5 model and all the System Functions data of the SV-4 model. The re-factoring transformation will refine the SV-5 model to obtain the same model with the calculated traceability matrix.

Acknowledgments

This work is being partially supported by Modelplex, European Integrated Project (FP6-IP#034081).

References

[1] *Systems Architecture* online definition from Wikipedia website. Available at http://en.wikipedia.org/wiki/System_architecture

[2] *DoDAF* online definition from Wikipedia website. Available at http://en.wikipedia.org/wiki/DODAF

[3] *DoDAF 2004 Volume II: Product Description*, 4/2/2004. Available at http://www.defenselink.mil/nii/global_Info_grid.html

[4] Jouault, F., Bézivin, J.: *KM3: a DSL for Metamodel Specification*. In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037, Bologna, Italy, pp. 171-185. 2006

[5] Didonet Del Fabro, M., Bézivin, J., Jouault, F., Valduriez, P.: *Applying Generic Model Management to Data Mapping*. In proc. of Base de Données Avancées (BDA 2005), October 17-20, 2005, Saint-Malo, France

[6] Jouault, F., Kurtev, I.: *Transforming Models with ATL*. In proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica

[7] Bézivin, J., Jouault, F., Touzet, D.: *An Introduction to the ATLAS Model Management Architecture*. Research Report LINA, (05-01). 2005

[8] Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: *Model-based DSL Frameworks*. In: Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, OR, USA. ACM. 2006

[9] *IDEF1X, Integration Definition For Information Modeling*, Federal Information Processing Standards Publication 184, December 21, 1993

[10] Bernstein, P., A.: *Applying Model Management to Classical Meta Data Problems*. In proc. of the 1st Biennial Conf. on Innovative Data Systems Research (CIDR 2003), pp 209-220