TECHNICAL TRANSACTIONS | **CZASOPISMO TECHNICZNE**
FUNDAMENTAL SCIENCES | NAUKI PODSTAWOWE
1-NP/2016

PIOTR ZABAWA*

NAMEDELEMENT REVISITED IN AN ASPECT-ORIENTED APPROACH

NOWE SPOJRZENIE NA NAMEDELEMENT W PODEJŚCIU ZORIENTOWANYM NA ASPEKTY

Abstract

In this paper a novel concept of adding structural responsibilities to meta-model classes for decreasing the meta-model complexity is introduced. This mechanism is supported by a combination of new Context-Driven Meta-Modeling Paradigm (CDMM-P) and its implementation in the form of the Context-Driven Meta-Modeling Framework (CDMM-F) with aspect-oriented paradigm and its AspectJ implementation supporting functionality and structure enrichment. The concept presented in the paper confirms the openness of CDMM-P and CDMM-F on the applicability of the aspect-oriented approach. It is also crucial for the process of generalization of notions introduced into the meta-model when a new modeling language is designed. It also helps to restructure the meta-model from the perspective of reusability. The NamedElement, known from many Object Management Group's (OMG) standards, was chosen.

Keywords: aspect oriented design, *aspect oriented programming*, *model, meta-model*, *meta-meta-model*, *responsibility*, *cross-cutting concern*, *dependency injection*, *inversion of control*

Streszczenie

W artykule wprowadzono nową koncepcję dodawania odpowiedzialności strukturalnych do klas metamodelu służącą zmniejszeniu jego złożoności. Mechanizm ten jest wspierany przez zestawienie nowego paradygmatu Context-Driven Meta-Modeling Paradigm (CDMM-P) i jego implementacji w postaci frameworku Context-Driven Meta-Modeling Framework (CDMM-F) z paradygmatem aspektowym i jego implementacją AspectJ wspierającą wzbogacanie funkcjonalności i struktury. Koncepcja prezentowana w artykule stanowi potwierdzenie otwartości CDMM-P i CDMM-F na możliwość stosowania podejścia aspektowego. Jest ona również kluczowa dla procesu uogólniania pojęć wprowadzanych do metamodelu podczas projektowania języka modelowania. Pomaga ono także w restrukturyzowaniu metamodelu z perspektywy ponownego użycia. Został wybrany NamedElement znany z wielu standardów Object Management Group (OMG).

Słowa kluczowe: projektowanie zorientowane na aspekty, *projektowanie aspektowe*, *programowanie zorientowane na aspekty*, *programowanie aspektowe*, *model*, *metamodel*, *metametamodel*, *odpowiedzialność*, *zobowiązanie przekrojowe*, *wstrzykiwanie zależności*, *odwrócenie sterowania*

DOI: 10.4467/2353737XCT.16.136.5715

* Piotr Zabawa (pzabawa@pk.edu.pl), Department of Physics, Mathematics and Computer Science, Cracow University of Technology.

1. Introduction

The paper is addressed to the NamedElement meta-model or meta-meta-model element, which is common to many well-known (meta-)meta-models. For convenience, the following notions are applied further in the paper:

- (meta-)meta-model or (m)mm denotes meta-meta-model or meta-model respectively
- mm denotes meta-model
- (m)mm denotes meta-meta-model
- (meta-)model or (m)m denotes meta-model or model respectively
- m denotes model
- (m)m is an instance of (m)mm, that is m is an instance of mm and mm is an instance of (m)mm
- s suffix denotes plural number of each notion above

The NamedElement can be met for example in (m)mms, like Meta-Object Facility (MOF) and its different realizations as well as in (m)ms, like Unified Modeling Language (UML) and Business Process Model and Notation (BPMN2). This (m)mm element is specific, because the responsibility it introduces into (m)mm affects many (m)mm elements. So, the nature of such common responsibility can be named cross-cutting structural responsibility or cross-cutting structural concern. Its responsibility is to enrich many (m)mm elements by the name represented in the form of a string. This way for example instances of classes or meta-classes or relationships may store their names in (m)ms.

Traditionally, the NamedElement class is introduced to (m)mms via a generalization relationship. However, this relationship is not supported directly by Context-Driven Meta--Modeling Framework (CDMM-F) [10–12] based on Context-Driven Meta-Modeling Paradigm (CDMM-P) introduced in [9] as the framework is located in data-layer. That is why the paper was introduced, just to explain how this kind of (m)mm elements may be introduced in the context of CDMM-F with the help of an aspect-oriented approach. The way of the element is introduced impacts on the features of the (m)mm as the whole.

The analogy between functional responsibilities and structural responsibilities which is referenced in the paper results from the observation that both responsibilities have a dual nature. Moreover, the problem of functional responsibilities is widely discussed in [1, 2, 7, 8] while the problem of structural responsibilities is almost completely ignored. However, it is crucial for meta-modeling domain as well as to data layer design. This paper is focused on the meta-modeling domain only.

2. Traditional Approach to NamedElement

As mentioned in section 1, the NamedElement is represented in the form of the class that contains one field of type string to store the name of the instance of this class. The NamedElement class is related to other mmm elements via a generalization relationship. If the NamedElement class is not abstract, then, in the case of the mm, the NamedElement instance is the model element which contains the name of model class. In the case of the mmm the NamedElement instance is the mm element which contains the

name of the mm class. Otherwise, if the NamedElement class is abstract, then, in the case of the mm, the NamedElement instance is the instance of the nearest concrete subclass of NamedElement class. This instance constitutes the model element which contains the name of the model class. In the case of the mmm the NamedElement instance is the instance of the nearest concrete subclass of NamedElement class. This instance constitutes the mm element, which contains the name of the mm class. NamedElement is represented in (m)mm by abstract classes.

The main problem with the traditional approach to creating (m)mms is the fact that their elements are interrelated at compile-time. So, the (m)mm graph is created during compilation process and not at run time. Thus, the relationships are static. This kind of interrelating mm elements influences the change introduction ease significantly. Different relationships interrelate classes differently. The weakest static relationship is UML dependency relationship, the stronger relationships are associative relationships, association (the weakest from this set), aggregation and composition (the strongest from this set). Unfortunately, the generalization relationship is the strongest one. And just this relationship is applied not only for NamedElement but for many other (m)mm classes. The popularity of this relationship was originated by knowledge modeling, where generalization is one of the most important relationships – it helps to build generalization hierarchies. However, in the software engineering domain the application of this relationship should be limited. And it is limited in target applications in many ways, like for example by application of design patterns. Nevertheless, in the meta-modeling domain it is promoted.

The approach discussed in the paper is different than the one presented above – it helps to interrelate (m)mm classes with their additional static responsibilities dynamically by injecting static responsibilities to (m)mm classes. The mechanism of injecting this kind of responsibilities is supported by aspect-oriented approach while (m)mm classes are managed by CDMM-F. The injecting concept and its applicability to (m)mms construction is discussed in section 3 while the role the CDMM-F plays in (m)mms definition is explained in section 4.

3. Structural Responsibility Injection

In contrast to the static compile-time concept of interrelating (m)mm classes presented above, this section is focused on application of the concept of interrelating (m)mm classes dynamically. Some consequences of the dynamic injection of relationships into (m)mms are also briefly discussed below.

The static responsibilities help to construct top hierarchies for (m)mms. They can be injected to (m)mms in order to address two modeling language designer needs – to introduce cross-cutting structural responsibilities for many existing (m)mm classes or to perform an mmm restructurization/refactorization process. The first need is typically planned from the very beginning of (m)mm defining process while the second need is usually involved by the observations made during the process of (meta-)modeling language definition. The characteristics of each need and its possible solution is presented in succeeding subsections below.

3.1. Cross-Cutting Structural Responsibility

The notion of cross-cutting concern is well known in Aspect-Oriented Design (AOD) and Aspect-Oriented Programming (AOP) [1–8], but it is related there to the functional concerns and is handled there by pointcuts and advices. However, at the same time AOD and AOP introduce the concept of enriching existing class hierarchies by classes interrelated statically to these hierarchies. Thus, for symmetry, the concept of enriching hierarchies statically can be seen from the perspective of static/structural concerns. Per analogy we have core structural concern, that is (m)mm to be enriched statically and other concerns. As the aspect-oriented approach fits to the concept of inversion of control (IoC) architectural pattern, all concerns both functional and static may be injected to the core concerns both functional and static respectively in the form that does not impact core concern's source code in any form. Moreover, the concerns are orthogonal, which means that one concern does not influence other any concern. In consequence, the static responsibilities can be added to (m)mm independently of each other.

It is worth noticing that there is also a notion of cross-cutting concerns in AOP. The cross--cutting concern is such a concern that crosses core-concern in a significant number of places. The more such places can be encountered, the more useful the IoC architectural pattern is. However, this pattern was applied so far for adding functional concerns, like error handling, system activity logging, auditing and many others. In this paper the same approach is applied to adding cross-cutting structural concerns. The NamedElement is a good example for such the cross-cutting concern as having the name is very common feature of (m)mm elements. Cross-cutting concerns are usually identified well before the meta-modeling process starts. However, they can be also added during this process in the case when they are recognized late. In section 3.2 the last cross-cutting structural responsibility addition is presented.

3.2. Meta-Model Refactorization

This section is focused on the (m)mm refactorization problem. A special case of the refactorization is involved by late recognition of a cross-cutting concern – this problem was characterized in section 3.1. But, usually the scope of the (m)mm refactorization for the purpose of structural responsibility addition is limited. As the consequence, both kinds of refactorization can be handled in the similar way although they have different purposes in the (meta-)modeling language design process. So, the same mechanism of adding structural responsibilities as the one described in section 3.1 can be applied for both forms of refactorization.

A simple example of such refactorization is presented in Figures 1 and 2. The UML class diagram for the mm structure before refactorization is presented in Figure 1 while the model after refactorization is depicted in Figure 2. The refactorization is limited to generalization of the fact, that both classes B1 and B2 have the same data field. The data field is thus moved to the new class T, which is aggregated both in B1 and B2.

Figure 2 presents a conceptual UML diagram, as it is informal for AOP. Nevertheless it reflects the fact of sharing common data field from class T well.

Fig. 1. Sample meta-model before structural refactorization

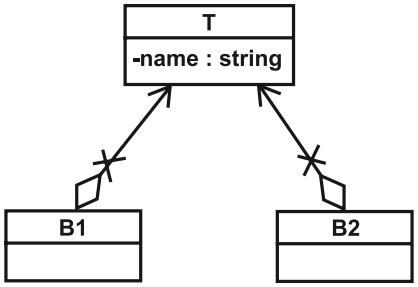

Fig. 2. Sample meta-model after structural refactorization in AOP approach – conceptual UML class diagram

The example from Figures 1–2 can be implemented in Java/AspectJ technologies in the form presented in Listings 1–2 respectively.

```
package pl.edu.pk.pz.aop.mm;

public class B1 {
  String name;
}
public class B2 {
  String name;
}
```

Listing 1. Java implementation of sample meta-model before refactorization

The classes that constitute structural responsibilities are located in the aspects layer (aspects) while the hierarchies to be enriched are placed in the class-object layer (classes). All these constructs are already available in AspectJ in the form of inter-type declarations (for modifying class hierarchies) and aspects (containers for all elements introduced by AspectJ to Java language).

In Listing 2 just the AspectJ AOP implementation was used to inject the T class as the default implementation of its IT interface. And this is a typical approach for this technology – classes are injected in the form of the relationships constructed from @DeclareParents annotation arguments.

One step more may be done in AOP – the aspects layer may be moved to the Spring framework and combined with AspectJ. However, the most important limit in the application of AOP to meta-modeling this way is connected to the fact that aspects are not instantiable (their lifecycle is synchronized with the lifecycle of the appropriate class instance in the best case, so they cannot exist without the class instance). As the result, the relationships represented by aspects do not have their instances. In consequence the relationships cannot

be differentiated, used or re-used between different (m)mms as separate entities. The CDMM approach is different as the relationships may have their instances as they are represented by classes. The core concept of CDMM-F is also based on the same mechanism as the one shown in Listings 1–2. However, the aspects in CDMM-F are used to interrelate (m)mm graph nodes represented by (m)mm entity classes by (m)mm edges represented by (m)mm relationship classes.

```
// meta-model classes
package pl.edu.pk.pz.aop.mm;

public class B1 {}
public class B2 {}

// top hierarchy package
package pl.edu.pk.pz.aop.th;

public interface IT {
  public String getName();
  public void setName(String name);
}

public class T implements IT {
  private String name;
  public String getName(){return name;}
  public void setName(String name){this.name = name;}
}

// aspects layer
package pl.edu.pk.pz.aop.aspect;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.DeclareParents;

import pl.edu.pk.pz.aop.th.IT;
import pl.edu.pk.pz.aop.th.T;

@Aspect
public class B1 {
  @DeclareParents(value="pl.edu.pk.pz.aop.mm.B1",defaultImpl=T.class)
  public IT t;
}
@Aspect
public class B2 {
  @DeclareParents(value="pl.edu.pk.pz.aop.mm.B2",defaultImpl=T.class)
    public IT t;
}
```

Listing 2. Java/AspectJ implementation of sample meta-model after refactorization

The NamedElement class can be injected in place of class T to the class layer or to the classes defined in CDMM-F. However, the classes from the example in this section have different names than the ones presented in the context of CDMM in order to underline significant differences between AOP approach and CDMM approach. The specifics of the CDMM approach from the refactorization perspective is explained in section 4.

4. Structural Responsibility Injection in CDMM

It was shown in section 3 that direct application of AOP to the meta-model classes introduces an important limit – relationships are not represented in the form of classes but in the form of aspects. In consequence, (m)mms can be built in this approach from classes located in (m)mm graph nodes interrelated by relationships located in (m)mm edges, which are represented in the form of an aspect. So, this approach just supports the concept of modeling relationships between classes in the form of references. Moreover, this feature introduces asymmetry to this approach. As the result of this asymmetry, (m)mm graph nodes are reusable (classes and their instances) while (m)mm graph edges are not reusable (aspects without instances).

In contrast to the typical AOP approach presented above, relationships in CDMM are represented in the form of classes which are reusable. CDMM approach allows for injecting relationships as classes that represent relationships in place of injecting relationships into classes in the form of aspects. In consequence, the relationships play the role of structural responsibilities of the interrelated classes. This approach is symmetrical and more general than the one based on naive application of AOP paradigm. In the CDMM approach both classes and relationships exist independently of each other and they are interrelated at run--time by Spring application context XML file [12]. So, (m)mm graph node classes as well as (m)mm graph edge classes are subject of reuse between different (m)mms. Thus, the (m)mms constructed according to CDMM approach may be easily customized, changed, designed from scratch and each part of them can be easily reused.

The paper is focused on handling the problem of NamedElement handling in CDMM. It is worth noticing that the structural responsibilities can be injected with the help of aspect oriented approach to the CDMM based (m)mm. The same technique can be used for injecting NamedElement into (m)mm graph. The NamedElement may be seen as just another structural responsibility of the (m)mm graph node or edge classes – the responsibility of (m)mm element name storage.

The concept of introducing NamedElement into CDMM (m)mm graph with the help of aspect orientation is presented below in the form of the example similar to the one presented above. However, this example refers to Spring notions like beans and application context and it is related to CDMM-F (m)mm.

The same (m)mm as the one presented in Figure 1 was chosen to represent the (m)mm before refactorization. The result of the refactorization of this (m)mm is presented in Figure 3.

Both Figure 2 and Figure 3 contain conceptual diagrams – they are not formal as since 2001 the AOP is out of scope of the UML standard. The CDMM approach makes it possible to define any (meta-)modeling language and to generate the self-organizing MDA-like modeling tool for this language. This way the concept of automatic model-driven aspect-oriented software generating can be achieved without standardization of the (meta-)modeling language. This CDMM characteristic feature applies for any technology which is in scope or out of scope of MDA standards.

The most important elements of the source codes for the example from Figure 1 that implement (m)mm in CDMM are presented in Listing 3 and Listing 4. Java source code

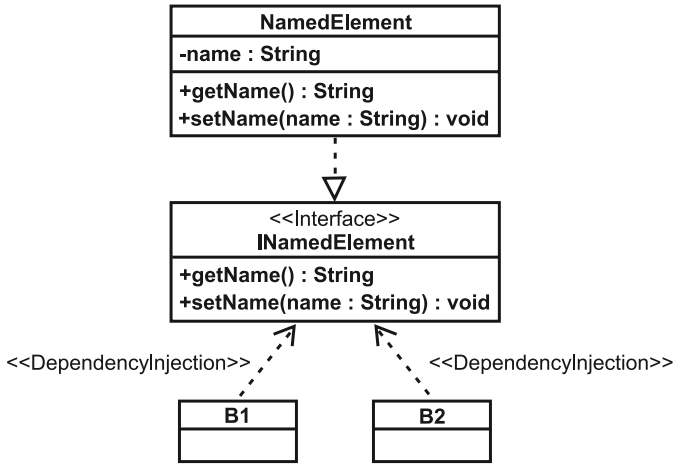

Fig. 3. Sample meta-model after structural refactorization in CDMM approach – conceptual UML class diagram

for two (m)mm classes is shown on Listing 3. The most important part of the CDMM-F's application context file for the (m)mm from Figure 1 is presented on Listing 4.

```
package com.componentcreator.metamodel.coremetamodel.domainsimpl;

public class DB1 extends BaseMetamodelCore implements IDB1 {
  String name;
}
public class DB2 extends BaseMetamodelCore implements IDB2 {
  String name;
}
```

Listing 3. Meta-model elements before refactorization

Now, a new (m)mm element is introduced. As a consequence of its introduction the name field migrates from B1 and B2 CDMM (m)mm classes to the new (m)mm element. This new element is just NamedElement and its definition is presented on Listing 5.

The structural responsibility represented in the paper by NamedElement can be added to some (m)mm elements via inclusion of the application context file presented in Listing 6 in the application context file from Listing 4. It was already stated that each such cross--cutting structural responsibility like NamedElement is orthogonal to the core concern – (m)mm and to other cross-cutting concerns. This orthogonality is represented by independence between responsibilities injected this way and by inclusion of extra application context.

The example presented above shows how to apply aspect oriented approach to enrich (m)mm defined in CDMM-F structurally. This approach to adding new concerns to (m) mm does not impact classes from (m)mm as long as this addition does not result from (m) mm refactorization. However, even in this case, the (m)mm change does not result from addition of new structural responsibility by aspects, but from the nature of the refactorization process itself. In the case of designing (m)mm and defining cross-cutting responsibilities

for it in advance, the addition of these responsibilities may be done in separation from the (m)mm's definition.

```
<beans>
  <!-- Root -->
  <bean class="com.componentcreator.metamodel.coremetamodel.root.RootMetamodelCore"
    id="root" scope="singleton"></bean>
  <!-- Root direct neighbours (collections) -->
  <bean class="com.componentcreator.metamodel.coremetamodel.domainsimpl.DB1"
    id="generalization" scope="prototype"></bean>
  <bean class="com.componentcreator.metamodel.coremetamodel.domainsimpl.DB2"
    id="class" scope="prototype"></bean>
  <!-- Responsibility implementations -->
  <!-- Root direct neighbours (collections of CPoliOMulti type) -->
  <bean class=
    "com.componentcreator.metamodel.coremetamodel.responsibilitiesimpl.RCollectionCPOM"
    id="collectionImplForRoot">
      <constructor-arg>
        <list>
          <value>com.componentcreator.metamodel.coremetamodel.domainsimpl.DB1</value>
          <value>com.componentcreator.metamodel.coremetamodel.domainsimpl.DB2</value>
        </list>
      </constructor-arg>
  </bean>
  <!-- Responsibility injections -->
  <aop:config>
    <aop:aspect id="holderA" ref="holderAAspect">
      <aop:declare-parents
        types-matching=
          "com.componentcreator.metamodel.coremetamodel.root.RootMetamodelCore"
        implement-interface=
          "com.componentcreator.metamodel.coremetamodel.responsibilities.IRCollectionCPOM"
        delegate-ref="collectionImplForRoot"/>
    </aop:aspect>
  </aop:config>
</beans>
```

Listing 4. Meta-model graph before refactorization

```
package com.componentcreator.metamodel.coremetamodel.metaontologies.namedelement;

public interface INamedElement {
  public void setName(String name);
  public String getName();
}
public class NamedElement implements INamedElement {
  private String name;
  @Override
  public void setName(String name) {
    this.name = name;
  }
  @Override
  public String getName() {
    return name;
  }
}
```

Listing 5. New meta-model element to be injected into meta-model graph (NamedElement)

```
<beans>
 <!-- NamedElement responsibility injections -->
 <aop:config>
  <aop:aspect>
   <aop:declare-parents
    default-impl=
     "com.componentcreator.metamodel.coremetamodel.metaontologies.namedelement.NamedElement"
    implement-interface=
    "com.componentcreator.metamodel.coremetamodel.metaontologies.namedelement.INamedElement"
    types-matching="com.componentcreator.metamodel.coremetamodel.domainsimpl.DB1" />
  </aop:aspect>
  <aop:aspect>
   <aop:declare-parents
    default-impl=
     "com.componentcreator.metamodel.coremetamodel.metaontologies.namedelement.NamedElement"
    implement-interface=
     "com.componentcreator.metamodel.coremetamodel.metaontologies.namedelement.INamedElement"
    types-matching="com.componentcreator.metamodel.coremetamodel.domainsimpl.DB2" />
  </aop:aspect>
 </aop:config>
</beans>
```

Listing 6. Meta-model graph after refactorization

5. Conclusions

This paper shows that the CDMM concept may be joined with other concepts applicable in software engineering domain. More specifically, it illustrates how the (meta-)meta--model graph implemented in CDMM-F can be enriched structurally by the application of AOP approach. The examples have shown that the most important advantages of AOP are preserved when the CDMM approach is used for (meta-)meta-model creation. Moreover, the combination of CDMM and AOP can be applied both for (meta-)meta-model refactorization as well as for the initial (meta-)meta-model design decisions.

The fact that AOP-oriented structural responsibilities can be injected into (meta-) meta-model results in very important feature of the presented combination of technologies – it introduces independence of life-cycles. The CDMM based (meta-)meta-model can be changed in large extent independently from changes introduced into AOP based structural responsibilities and vice versa. This feature helps to simplify and manage the process of designing modeling languages (meta-models) or designing languages used to define modeling languages (meta-meta-models).

References

[1] Filman R.E., Elrad T., Clarke S., Aksit M., *Aspect-Oriented Software Development*, 2004

[2] Gradecki J.D., Lesiecki N., *Mastering AspectJ: Aspect-Oriented Programming in Java*, First Edition, Wiley 2003

[3] Huang Sh.Sh., Smaragdakis Y., *Easy Language Extension with Meta-AspectJ*, Proceeding ICSE'06 Proceedings of the 28th International Conference on Software Engineering, p. 865-868, ACM, New York, NY, 2006.

[4] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W.G., *An overview of AspectJ*, In ECOOP'01: Proceedings of the 15th European Conference on Object-Oriented Programming, p. 327-353, London, UK, 2001, Springer-Verlag.

[5] Kiczales G., Lamping J., Menhdhekar A., Maeda C., Lopes C., Loingtier J.-M., Irwin J., *Aspect--oriented programming*, [in:] Akşit M., Matsuoka S., editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, p. 220-242, Springer-Verlag, Berlin, Heidelberg and New York, 1997.

[6] Kojarski S., *Third-Party Composition of AOP Mechanisms*, Ph.D. thesis, Graduate School of Northeastern University, ProQuest LLC, 2008.

[7] Laddad R., *AspectJ in Action*, Second Edition, Enterprise AOP With Spring Applications, Manning Publications, Greenwich, 2010.

[8] Miles R., *AspectJ Cookbook*, First Edition, O'Reilly Media, 2004.

[9] Zabawa P., Stanuszek M., *Characteristics of the Context-Driven Meta-Modeling Paradigm (CDMM-P)*, Technical Transactions of Cracow University of Technology, 2014, vol. 111, No. 3-NP, p. 123-134.

[10] P. Zabawa, *Context-Driven Meta-Modeling Framework (CDMM-F) – Internal Structure*, 2016, submitted for publication.

[11] Zabawa P., Nowak K., *Context-Driven Meta-Modeling Framework (CDMM-F) – Simple Horizontal Case-Study*, 2016, submitted for publication.

[12] P. Zabawa, *Context-Driven Meta-Modeling Framework (CDMM-F) – Context Role*, Technical Transaction 1-NP/2015, p. 105-114, DOI: 10.4467/2353737XCT.15.119.4156