Using DEMO to Objectify Metamodel Evolution

Nuno Silva[1], José Tribolet[1], Miguel Mira da Silva[1] and Carlos Mendes[2]
[1]*Instituto Superior Técnico, Universidade de Lisboa, Avenida Rovisco Pais 1, Lisboa, Portugal*
[2]*INOV Inesc Inovação, Rua Alves Redol 9, Lisboa, Portugal*

Keywords: Metamodel Evolution, Coupled Operations, EO, DEMO, ATD, OFD.

Abstract: A metamodel is an important aspect of defining a modeling language. It specifies the language's syntax through a set of constructs as well as how the language models are ought to be composed. Modeling languages, and thus their metamodels, are subject of constant evolution due to changing language requirements as consequence of business changes. Therefore, perceiving the essential aspects responsible for altering the structure of metamodels when a change requirement arises can become an issue. The Enterprise Ontology theory and its methodology (DEMO) provide ontological knowledge about organizations resulting in organizational self-awareness. Applying this methodology to the context of metamodeling can be a starting point for uncovering the essential aspects, i.e., the ontological knowledge regarding metamodel evolution. For that purpose, we modeled two diagrams using the DEMO methodology. The input for both diagrams was a set of coupled operations defined in Herrmannsdörfer's evolutionary metamodeling research. In the end we stated the main conclusions of our work and themes for future work.

1 INTRODUCTION

Model-based development provides an increase in the abstraction level of today's software development through the use of models (France and Rumpe, 2007; Pretschner et al., 2007). These models are built through the use of adequate modeling languages (Bézivin and Heckel 2006) that help users to directly express the abstractions from their problem domain (Guizzardi 2005).

A modeling language is defined through its abstract syntax, concrete syntax and semantics. The abstract syntax defines the set of valid models and is placed in the center of a modeling language definition (Kleppe 2008) from which both the concrete syntax and semantics are defined. Metamodeling is therefore the act of modeling the abstract syntax of modeling languages. For that purpose, innumerous languages for defining abstract syntax were defined such as the Meta Object Facility (MOF) from OMG (Object Management Group 2013), Kernel Metametamodel (KM3) from INRIA (Jouault and Bézivin, 2006), amongst others. The metamodel defines the constructs that the modeling language provides as well as how to compose them to models.

Like software changes over time (due to new requirements), also modeling languages are prone to change when new requirements arise (Favre 2005). These changes are usually a consequence of one or more business changes and reasons such as Perfective Maintenance, Corrective Maintenance, etc. are the main drivers of these changing requirements. Therefore, the language engineers evolve the modeling language to a new version by first adapting its metamodel to the additional requirements. However, metamodel adaptation may invalidate existing artifacts that depend on the metamodel (Sprinkle, 2003). Most importantly, existing models may no longer conform to the adapted metamodels. The existing artifacts need to be migrated to conform to the metamodel again, so that they can be used with the evolved modeling language. This can become a difficult task when the language engineers do not possess any ontological knowledge nor self-awareness regarding the metamodel and its core structural elements.

In this paper we propose to use DEMO, namely the ATD and OFD diagrams, to illustrate how a metamodel is structured in terms of its elements, which are the actor roles behind each metamodel structural change, what are the core transactions types responsible for altering the metamodel

Silva, N., Tribolet, J., Silva, M. and Mendes, C..
Using DEMO to Objectify Metamodel Evolution.
In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - Volume 2: KEOD*, pages 493-500
ISBN: 978-989-758-158-8

structure, and what kind of relationship constraints exist between metamodel elements, thus obtaining an ontological awareness regarding metamodel evolution. The provided input for each diagram was based on Herrmannsdörfer coupled operations (Herrmannsdoerfer, 2011).

DEMO (Design & Engineering Methodology for Organizations) is a methodology for modeling, (re)designing and (re)engineering organizations and networks of organizations. The theory used as foundation for this methodology is called Enterprise Ontology (EO) (Dietz, 2006) that, by itself, is based on the speech act theory. DEMO is composed of four aspect models that express the ontological knowledge of an enterprise in an easily accessible and manageable way (Dietz, 2006).

The remainder of this paper is structured into three main sections. The Theoretical Background section describes some important theoretical aspects concerning the topic of metamodel evolution, and Dietz Enterprise Ontology theory as well as the DEMO methodology. The Metamodels and DEMO section illustrates and explains the reasoning behind each of the two proposed DEMO diagrams. Finally, the Conclusions section states the main conclusions of our work and some future work topics.

2 THEORETICAL BACKGROUND

This section is divided into two sub-sections. The first one considers some of the relevant aspects behind the concept of metamodels and their evolution (metamodeling, reasons for evolution of modeling languages, and coupled operations). The second describes the theory behind Enterprise Ontology (a branch of the Enterprise Engineering field) and the methodology that supports it called DEMO.

2.1 Metamodel Evolution

In this section we describe important aspects of metamodel evolution concerning modeling languages such as the metamodeling process, i.e., the metamodeling languages and constructs for creating and changing metamodels, and the required operations for performing a metamodel change covered by Herrmannsdörfer (Herrmannsdoerfer, 2011).

A modeling language is not a static, immutable thing. It may evolve due to a multitude of reasons. Those can be Perfective Maintenance, Corrective Maintenance, Preventive Maintenance, and Adaptive Maintenance (Herrmannsdoerfer 2011). All these reasons are valid for evolving the abstract syntax of a modeling language, i.e., for evolving the language's metamodel.

2.1.1 Metamodeling – Languages and Constructs

A modeling language is defined through its abstract syntax, concrete syntax and semantics. The abstract syntax is what defines the set of valid models and is placed in the center of a modeling language definition (Kleppe, 2008) from which both the concrete syntax and semantics are defined.

Metamodeling is the act of modeling the abstract syntax of a modeling language. For that purpose, innumerous languages for defining abstract syntax were defined (Meta Object Facility – MOF – from OMG (Object Management Group, 2013), Kernel Metametamodel – KM3 – from INRIA (Jouault and Bézivin, 2006), amongst others.

MOF is the most widely applied metamodeling language. This is due to MOF being a standard, therefore not being proprietary to a certain metamodeling tool, and because all metamodeling languages are conceptually similar (all represent models as graphs) with the only difference being the provided metamodeling constructs.

The E-MOF (being one of MOF's compliance point) is based on the object-oriented paradigm and defines a hierarchy of models grouped into layers called meta hierarchy (Bézivin, 2005). The **model layer** is the bottom layer containing all the models that are specified according to a modeling language. All models from this layer conform to a metamodel defined in the next upper layer. The **metamodel layer** is the middle layer that contains all the metamodels defining the abstract syntax of modeling languages. Each metamodel in this layer conforms to the metametamodel in the next upper layer. Finally, the **metametamodel layer** (being the upmost layer) contains the metametamodel that defines the abstract syntax of a metamodeling language. The metametamodel in this layer conforms to itself.

The E-MOF metametamodel provides a set of constructs for defining metamodels. A **Package** consists of a number of *types* and *sub packages*. In order to distinguish a package from other, each one uses a name feature. The name of a sub package needs to be unique among all packages belonging to a super package and so on. **Type** is a common abstract super class of *Class* and *PrimitiveType*. In

the complete E-MOF metametamodel, all nodes must be instances of Types. In order to distinguish types, they also use a name. The name of a type has to be unique among all types associated to the same package.

A **Feature** is a common abstract super class of *Reference* and *Attribute*. To be able to distinguish features from each other, a Feature has a name. Like with the Package and Type constructs, the name of the feature needs to be unique within all features of the class (including the ones inherited from the super types). An Attribute can be similar to a Reference since both are represented as edges in the graph. However, when one compares both, Attributes have a *PrimitiveType* as type. As a consequence, edges representing instances of Attributes target nodes representing instances of primitive types. The same analogy can be applied to PrimitiveType and Class. Both represent nodes in the graph, however, when compared to Classes, **PrimitiveTypes** do not define references. Hence, nodes that are instances of PrimitiveType do not possess outgoing edges, thus being the terminal nodes in graphs. Each PrimitiveType can be specialized into DataTypes or Enumerations.

A **DataType** represents predefined primitive types such as Boolean, Integer and String. Data types may define an infinite number of possible literals, e.g. String. However, a model can only use a finite number of literals of a data type, since the set of nodes is finite. An **Enumeration** can define a finite set of literals. Each **Literal** that is used in the model is represented by exactly one node in the model and has a name to be able to distinguish it from the other literals.

2.1.2 Coupled Operations for Metamodel Evolution

Herrmannsdörfer (Herrmannsdoerfer, 2011) defined a library of 61-coupled operations for metamodel evolution. A *coupled operation* combines metamodel adaptations with model migrations. It allows information attachments about how to migrate corresponding models in response to a metamodel adaptation. Formally, it corresponds to a tuple *(adm, mig)* with a metamodel adaptation *adm*, and a model migration *mig*. Each operation was grouped into two major groups and respective sub-groups.

The first major group is the *primitive* operations group, which perform an atomic metamodel evolution step. Within these primitive operations we can distinguish between structural and non-structural primitives. *Structural* primitives create and delete metamodel elements whereas *non-structural* primitives modify existing metamodel elements (Herrmannsdoerfer, 2011). Then, we have composite operations. These can be decomposed into a sequence of primitive operations. These operations are grouped according to the metamodeling techniques they address as well as their semantics.

#	Operation Name	Classific.			Modelw.			OO dataware					APIw.		BMW	Evaluation				
		Language preservation	Model preservation	Inverse	[Wachsmuth, 2007]	[Becker et al., 2007]	[Ciccheti et al., 2008]	[Banerjee et al., 1987]	[Bréche, 1996]	[Fons, 1997]	[Claypool et al., 2000]		[Fowler, 1999]	[Digand Johnson, 2006]	FLUID	TA-Gen	PCM	GMF	Unicase	Quamoco	TTC
1	Create Package	r	p	2s		•										•	•				
2	Delete Package	r	p	1s		•										•					
3	Create Class	c	p	4s	•	•	•	•							•	•	•	•	•	•	
4	Delete Class	d	u	3u	•	•	•	•	•				•		•		•		•	•	
5	Create Attribute	c	s	7s	•	•	•	•							•	•	•	•	•	•	
6	Create Reference	c	s	7s	•	•	•	•							•	•	•	•	•	•	
7	Delete Feature	d	u	5/6u	•	•	•	•							•	•	•	•	•	•	•
8	Create Opposite Reference	d	u	9u		•						•	•					•	•	•	
9	Delete Opposite Reference	c	p	8s		•						•	•			•					
10	Create Data Type	r	p	11s		•															
11	Delete Data Type	r	p	10s		•														•	
12	Create Enumeration	r	p	13s		•										•	•	•	•	•	
13	Delete Enumeration	r	p	11s		•													•		
14	Create Literal	c	p	15s		•													•		
15	Merge Literal	d	u	14u		•											•				

Figure 1: Structural primitives table (Herrmannsdoerfer 2011).

Those are operations of *specialization/ generalization*, *inheritance*, *delegation*, *replacement*, and *merge/split* (Herrmannsdoerfer, 2011). Figure 1 illustrates a table providing an overview of all operations concerning the *Primitive->Structural* group.

Each coupled operation can be classified according to language preservation into refactoring (r), constructor (c) and destructor (d), as well as according to model preservation into model-preserving (p), safely (s) and unsafely (u) model-migrating (Herrmannsdoerfer, 2011).

The creation of non-mandatory metamodel elements, such as packages, classes, optional features, enumerations, literals and data types is model-preserving. Creation of mandatory features is safely model migrating. On the other hand, the deletion of metamodel elements requires deleting instantiating model elements, such as objects and links, by the migration. This poses the risk of migration to inconsistent models. Therefore, deletion operations are bound to metamodel level restrictions (e.g., packages may only be deleted, when they are empty. Classes may only be deleted, when they are outside inheritance hierarchies and are targeted neither by non-composite references nor by mandatory composite references. References may only be deleted, when they are neither composite, nor have an opposite. Enumerations and data types

may only be deleted, when they are not used in the metamodel and thus obsolete) (Herrmannsdoerfer, 2011).

In this paper, we only address the structural primitives operations. These coupled operations focus on structural changes of a metamodel, i.e., they focus on creating new things as well as deleting existing ones. The ontological act expressed by the Enterprise Ontology theory (described in the next section) refers to the creation of new original things, and DEMO provides the ontological knowledge regarding the structure, composition and environment of organizations which is precisely what we are trying to model with respect to metamodels.

2.2 Enterprise Ontology

The Enterprise Ontology (EO) theory proposed by Dietz (Dietz 2006) is based on four axioms – operation, transaction, composition and distinction – and the organization theorem. Each of these axioms and the organization theorem are supported by a set of foundations covered within seven EE theories ranging from four domains - philosophical, ontological, ideological and technological.

The φ -theory and τ -theory are put in the class of philosophical theories for two reasons: their concern is about conception and perception and the majority of papers and books in these fields are published in philosophical journals or book series. The ψ-theory and the δ-theory are put in the class of ontological theories because they are about the nature of things, in particular the nature of systems. Then, the β-theory is a technological theory. The μ-theory (model theory) is also put in this class because its main practical use is to bridge ontology to technology. Lastly, the σ-theory is undoubtedly an ideological theory (Dietz et al., 2013). All these theories provide the foundations for each of the four EO theory axioms described below.

The **operation axiom** states that the operation of an enterprise is constituted by the activities of actor roles being elementary chunks of authority and responsibility, fulfilled by subjects.

These subjects perform two kinds of acts: production acts (P-acts) and coordination acts (C-acts) and each these acts have definite results: production facts (P-facts) and coordination facts (C-facts), respectively. By performing production acts the subjects contribute to bringing about the goods and/or services that are delivered to the environment of the enterprise. By performing coordination acts subjects enter into, and comply with, commitments towards each other regarding the performance of production acts.

Competence is the collective knowledge, know-how and experience that is necessary and sufficient for a subject to perform production acts of a particular kind. Competence is related to a subject's profession (e.g. teacher). Authority is defined as the being authorized of a subject by an institution, e.g., by a company (employee) or by a society (client), to perform particular production acts and/or coordination acts (e.g. engineer of Company X). Responsibility is the socially felt need by a subject to perform the coordination acts for which it is authorized, in an accountable and self-aware manner.

The **transaction axiom** states that coordination acts are performed as steps in universal patterns. These patterns, also called transactions, always involve two actor roles (initiator and executor) and are aimed at achieving a particular result.

The **composition axiom** establishes the relationships between transactions. This axiom states that every transaction is enclosed in another transaction, or is a customer transaction of another transaction, or even a self-activation transaction.

Finally, the **distinction axiom** states that there are three distinct human abilities playing a role in the operation of actors, called *performa*, *informa*, and *forma*.

The EO theory highest point is the **organization theorem**. This theorem states that the organization of an enterprise is perceived as a heterogeneous system constituted by the layered integration of three homogeneous systems: the **B-organization** (from Business), the **I-organization** (from Intellect), and the **D-organization** (from Document).

Their relationships are that the D-organization supports the I-organization, and the I-organization supports the B-organization. The integration is established through the cohesive unity of the human being. Next we describe the methodology behind the EO theory (DEMO) with respect to the B-organization layer.

2.2.1 Demo

DEMO (Design & Engineering Methodology for Organizations) (Dietz 2006) is a methodology for modeling, (re)designing and (re)engineering organizations and networks of organizations. DEMO consists of four aspect models (Construction Model – CM, Process Model – PM, Action Model – AM, and the State Model - SM) in which the ontological knowledge of (the organization of) an enterprise is

expressed in an easily accessible and manageable way. Each of these models is represented by a set of diagrams, tables and lists (Dietz, 2006).

The 4 aspect models constitute the complete ontological model of the B-organization and subsequently represent the ontological model of the corresponding enterprise. The Actor Transaction Diagram (ATD) and the Transaction Result Table (TRT) express the Construction Model (CM). The Process Structure Diagram (PSD) and the Information Use Table (IUT) express the Process Model (PM). The Action Model (AM) is expressed by action rules specifications. The Object Fact Diagram (OFD), TRT, and IUT express the State Model (SM).

Concerning the scope of our research, the models that best describe the structure and the ontological rules and constraints of metamodel evolution are the CM and the SM respectively. The CM, provides useful practical applications such as showing the boundary of the organization, as well as the interface transactions with actor roles in the environment. The CM also shows the ontological units of competence, authorization and responsibility. Applying the CM to the scope of metamodel evolution can provide valuable insights on how metamodel changes can be seen from an 'organizational' perspective where actors (whether being humans or informational entities) can be identified in a set of transactions representing a specific metamodel change.

The SM is the source of ontological knowledge about the production world. This can be very suitable in practice since it can not only provide the concepts that are essential for the enterprise, but also help in conceiving the best concepts. It also simplifies the identification of business components (software components), based on the chunks of fact types around categories. This model can help us obtaining the knowledge regarding the core concepts of a metamodel and how these concepts interact with each other through their respective coexistence rules.

In the next section we will present, through means of an ATD and OFD diagrams, both the CM and the SM respectively within the scope of metamodel evolution, i.e., considering the coupled operations (structural primitives) described by Herrmannsdörfer.

3 METAMODEL EVOLUTION AND DEMO

In order to perceive, from an ontological perspective, how metamodels adapt to new language requirements we used DEMO as a way of expressing the ontological acts, i.e., the creation of new original facts. DEMO provides a set of models that produce insight on the ontological knowledge with respect to the structure, composition and the environment of an organization. Therefore, we had to adjust the system of interest from organizations to metamodels. So, the models we present below focus not on the essential aspects describing an organization, but rather on those describing metamodels.

With this approach, one can understand who are the main actors and transactions that support a metamodel change, how these actors relate with each other as well as the elements composing the metamodel. Furthermore, one can identify the cardinality and existence constraints that influence the relationship between elements of the metamodel. We used as input for each diagram Herrmannsdörfer coupled operations that create and delete metamodel elements (structural primitives) since those are the ones that directly influence the structure of a metamodel. The reasons for choosing the CM and the SM in particular are described at the end of section 2.3.

3.1 The Metamodel Evolution Construction Model

The Construction Model (CM) specifies the composition, environment, and structure of an organization, in this case, of a metamodel. The *composition* and *environment* are both a set of actor roles. The interaction *structure* consists of the transaction types in which the identified actor roles participate as initiator or executor. The Interaction Model (IAM) is therefore a sub-model of the CM and represents the execution of transactions between actor roles. All of this is expressed in an Actor Transaction Diagram (ATD) and a Transaction Result Table (TRT).

For expressing the main transactions concerning a metamodel structural change, we consider each coupled operation in Figure 1 to be a transaction since each operation, from a transaction perspective, creates a different fact.

Each transaction has the purpose of either creating or deleting a metamodel element. For example, if one considers the transaction type **T03 (create class)**, the transaction result of initiating and executing this transaction type is **R03 (class C has been created)**. The same line of thought is applicable to the remaining transaction types. The transaction type **T16 (create feature)** is not a

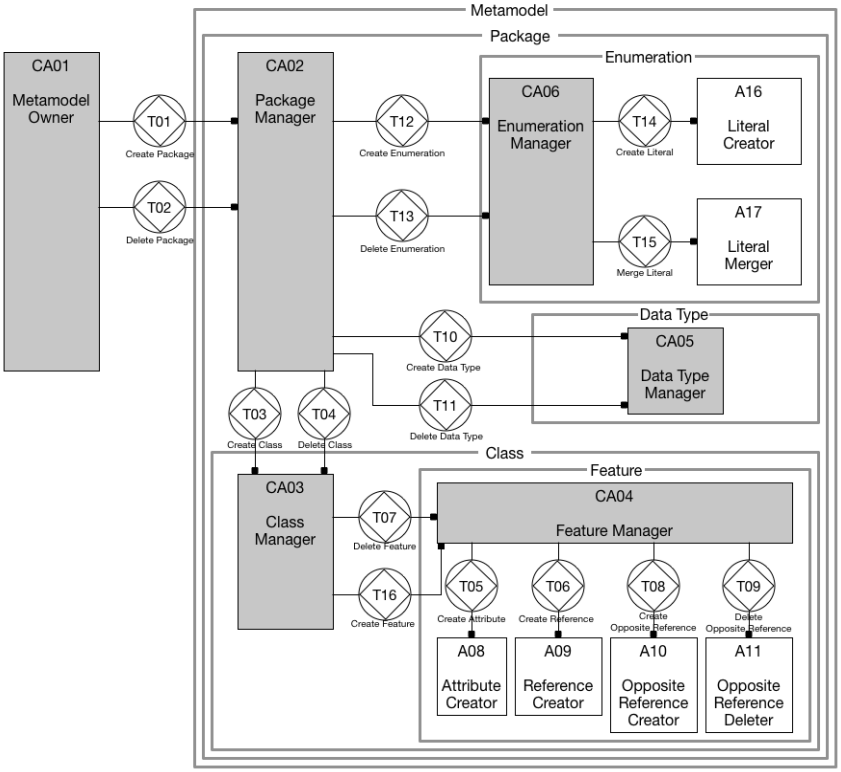

Figure 2: Actor Transaction Diagram (ATD) for metamodel evolution.

coupled operation from Figure 1; however its existence did not happened by chance. After explaining the ATD in more detail it will become clearer why we had the need to create this transaction type.

Figure 2 illustrates the global ATD with respect to a metamodel. All the actor roles within the metamodel boundary are the internal metamodel entities (e.g., informational objects managing the creation and/or deletion of metamodel elements such as classes, data types, etc.) responsible for executing a specific transaction type.

The metamodel organization is divided into several sub-organizations each one corresponding to one of the metamodels elements, such as Package, Class, Data Type, etc.

Within each sub-organization, there is a composite actor role managing all the execution part of creating and/or deleting the metamodel element for which the actor role is responsible (e.g., the Package Manager has the responsibility of executing the transaction type T01 - create package and T02 - delete package). These composite actor roles are divided into elementary actor roles. However, we abstracted those in order to simplify the diagram. The CA01 is out the metamodel boundary, meaning that this composite actor role is responsible for making a metamodel adaptation due to new language change requirements, whether is adding an element to the metamodel or removing one.

The need for the Metamodel Owner (CA01) to create or delete a Data Type or any other element from the metamodel is implicit in the ATD. When the Metamodel Owner wishes to create or delete a package, it triggers a sequence of transactions among the actor roles within the metamodel organization where new elements will be created or deleted from a package. This can be seen as a hierarchized delegation of responsibility. The only issue here had to do with the Class Manager not being able to delegate the Feature Manager the responsibility of creating a new feature, i.e., making a transaction request for the Feature Manager to create a new feature. Since a feature is a generalizations for both attribute and reference, the coupled operations specialize, at creation time, which feature will be created (if an attribute or a reference) therefore being unnecessary to have couple operation for creating a feature. However, not having the create feature transaction type breaks the ‘chain of delegation’ thus rendering the Feature Manager unable of executing the transaction type T05 (create attribute) and T06 (create reference).

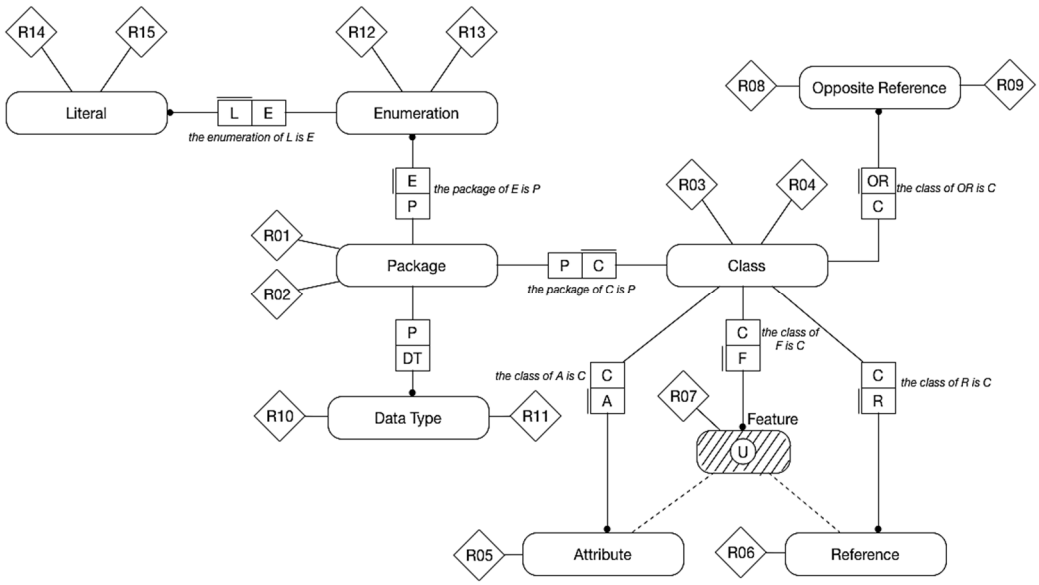

Figure 3: Object Fact Diagram (OFD) for metamodel evolution.

This is the reason why we created this new transaction type T16 (create feature).

Hence, the ATD proves to be useful in understanding the interaction structure, environment and composition of a metamodel. It expresses all the relevant transaction types that are involved in a metamodel adaptation from a structure perspective (i.e., either the creation or the removal of metamodel elements). With this diagram one can observe how the actor roles responsible for changing the metamodel structure interact, which ones are responsible for creating new production facts (i.e., new things), and which ones request the creation of those facts. In the next section we will describe the metamodel state model composed of the Object Fact Diagram (OFD).

3.2 The Metamodel Evolution State Model

The state model (SM) of an organization (in this case, of a metamodel) is the specification of the state space of the P-world consisting of specifying the object classes, fact types, and the result types, as well as the existential laws that hold (Dietz 2006). The SM is expressed through an Object Fact Diagram (OFD) and an Object Property List (OPL), although the latest will not be considered for the scope of our work. The OFD is fully based on the WOSL language (Dietz 2006). Figure 3 illustrates the OFD for the metamodel evolution.

Each category (rounded rectangle) presented in the OFD corresponds to each metamodel element that was created and/or deleted as a result of performing a specific transaction type. Associated to each category are the transaction results related to the metamodel element represented by the category. This diagram shows the relations among categories through the use of binary facts (the two box rectangle) representing the instances of each category. For example, a package instance P relates to a class instance C through the binary fact "the class of P is C". Some of these binary facts have unicity laws (lines above the facts) that act as cardinality constraints, i.e., in the case of the relationship between Package and Class, a package P can have many classes C, however a class C can only be related to one package, and so on for the remaining facts. The dependency laws (dots) illustrated in some of the categories demand that, for a specific category instance to exist, a binary fact relationship between that category and another must also exist. For example, in order for a Data Type DT to exist, a relationship between DT and a package P must also exist. Another aspect of this diagram worth mentioning is the generalization of the Feature category. A generalization type in WOSL language is the union of two or more categories. An example of a generalization is VEHICLE, defined as the union of CAR, AIRCRAFT, and SHIP. This means that, in order to identify a vehicle, one must not use a vehicle registration number but a car, an aircraft or a ship registration number. In this specific case, in order to identify a feature, one must use either the attribute or the reference identification.

Using this diagram, it is possible to identify the main constraints with respect to the existence and relationships of metamodel elements. By combining both the ATD and the OFD, it is possible to understand the restrictions and the consequences of creating or deleting metamodel elements by requesting and executing transactions. Also, a contribution of this diagram is the explicit representation of the relationships between metamodel entities and respective cardinality and dependency constraints that are not approached by Herrmannsdörfer in his research, thus giving a more comprehensive overview of the interaction amongst the metamodel elements that define and compose the metamodel structure.

4 CONCLUSIONS

In this paper we presented the modeling of the ontological aspects surrounding the evolution of metamodels according to new language specific requirements. The presented Actor Transaction Diagram (ATD), and Object Fact Diagram (OFD) provide the essential aspects and knowledge with respect to the main actors and transactions involved in a metamodel adaptation. For that we used Herrmannsdörfer coupled operations, more specifically the structural primitive operations, on account of being the ones that directly influence a metamodel structure when a structural adaptation is required. Also, the diagrams (in particular the OFD) present the main constraints associated to the existence of certain metamodel elements and their relationships with other elements.

These insights concerning the cardinality and existence constraints of these elements and their relationships are not explicitly covered in Herrmannsdörfer work, thus being a contribution for understanding, from an ontological perspective, how the structural elements of a metamodel interact amongst them and what are the respective dependencies. However, the main contribution of this work was objectifying Herrmannsdörfer metamodel evolution approach by using DEMO white-box models. The models used provide a set of advantages such as showing the composing boundaries of a metamodel, the interface transactions with actor roles in the environment, presenting the interface units of collaboration, showing the ontological units of competence, authorization, and responsibility, providing a holistic metamodel map, and identifying the essential concepts of a metamodel.

Nevertheless, this work lacks a practical validation, therefore being a limitation of our research. A field study with real practical examples could reinforce this research and would add a sound validation. For future work, and to provide a more thorough analysis of the ontology behind the evolution of metamodels, other DEMO models can be applied to this context, such as the Process Model (PM) or even the Action Model (AM). Also, applying a practical validation in the future would consolidate this research.

REFERENCES

Bézivin, J., 2005. On the unification power of models. *Software and Systems Modeling*, 4(2), pp.171–188.

Bézivin, J. & Heckel, R., 2006. Guest editorial to the special issue on language engineering for model-driven software development. *Software and Systems Modeling*, 5(3), pp.231–232.

Dietz, J. L. G., 2006. *Enterprise ontology: Theory and methodology*,

Dietz, J. L. G. et al., 2013. The discipline of enterprise engineering. *International Journal of Organisational Design and Engineering*, 3(1), pp.86–114.

Favre, J.-M., 2005. Languages evolve too! Changing the software time scale. In *Principles of Software Evolution, Eighth International Workshop on*. IEEE, pp. 33–42.

France, R. & Rumpe, B., 2007. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering (FOSE '07)*. pp. 37–54. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4221611\n http://dl.acm.org/citation.cfm?id=1254709.

Guizzardi, G., 2005. *Ontological Foundations for Structural Conceptual Model*, Available at: http://doc.utwente.nl/50826.

Herrmannsdoerfer, M., 2011. *Evolutionary Metamodeling*. Technische Universität München.

Jouault, F. & Bézivin, J., 2006. KM3: A DSL for metamodel specification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. pp. 171–185.

Kleppe, A., 2008. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*, Addison-Wesley Professional.

Object Management Group, 2013. OMG Meta Object Facility (MOF) Core Specification, Version2.1.4, 2(April)

Pretschner, A. et al., 2007. Software engineering for automotive systems: A roadmap. In *FoSE 2007: Future of Software Engineering*. pp. 55–71.

Sprinkle, J.M., 2003. *Metamodel driven model migration*. Vanderbilt University, Nashville, TN, USA.