On Instance-model Querying and Meta-model Transformation

Viet Cuong Nguyen, Xhevi Qafmolla

***Abstract*—In today's market environment, change is an integral part of all projects. As such, its proper management is a crucial task when it comes to reducing both time and cost of development. The classical modeling approach can improve the situation up to a proper extent but it is not enough, because the process is usually variable and complex. Therefore it is necessary to introduce different level of abstractions for modeling. Each of these levels should serve at a certain phase for a certain purpose in the process. In the early stages several elements are grouped together and aggregated at the higher level, in an abstract model. Throughout time, granularity becomes smaller while understanding of the concepts becomes clearer and clearer and we need to see already working prototypes. In this situation, the approaches on instance- and meta-modeling techniques promise to bring productivity and efficiency to the process. This paper outlines practices from both approaches. We introduce the approach of using Object Constraint Language (OCL) with a Domain Specific Language (DSL) for instance-level model querying, illustrating this method with some examples. We analyze OCL from the broad perspective discussing its advantages and pointing out some its limitations. Moving to a higher level of abstraction, we also present the usage of Kermeta - an extension to the meta-data language with an action language for specifying semantics and behavior of meta-models. We show how Kermeta provides the possibility of automated meta-model transformations.**

***Index Terms*—Domain Specific Languages, Instance-model, Kermeta, Model Driven Development, Meta-model, OCL.**

I. Introduction

Models, in general, are nowadays a well-established approach for representing processes in terms of a set of tasks that need to be performed in order to achieve a certain output. But in order to adapt to fast changing business requirements, companies are in need of flexible systems that allow rapid prototyping, testing and/or reconfiguration. From this perspective, models are viewed as artifacts that posses capability of querying, configuration, change and transformation.

Over the past years, different process modeling languages have been consequently proposed and established. They track different levels of detail, ranging from high-level models for business analysts, to specific and executable process models. To name the most common few of them, we could refer to Event-driven Process Chain (EPC) [5], Business Process Modeling Notation (BPMN) [11] or Unified Modeling Language (UML) [3].

In UML, generic modeling approaches are used. Since the late 90's UML has become de-facto a standard for Object-oriented analysis and design of information systems. UML maintains several aspects of software engineering; however it lacks the capability of working and supporting queries, which are often needed since the early stages of the Objects-first approach. Object Constraint Language (OCL) was originally designed specifically for expressing constraints of an UML model. Moreover, its ability to navigate the model and form collections of objects has lead to attempts to use it as query language [7] [8]. From this aspect OCL should have the expressive power of relational algebra. OCL and UML together posses enough semantics for presenting an indirect method for forming queries. In the first part of this paper we discuss the approach of using OCL for querying models.

In the second part we move on to a higher level of abstraction, meta-models. For many complex systems, a lot of aspects need to be put together from architectural to dynamic behaviors, functionalities and user interfaces. The design process can be described as the weaving of all these aspects into a detailed design model [4]. Model-driven methods aim at automating this weaving process. Our opinion is that the role of meta-modeling techniques is becoming just next to model querying, more and more significant. Meta-modeling structure, providing flexible capabilities in different environments, has become a great help in solving and adapting to different problems. Therefore in the second part of this paper we give a presentation of the meta-modeling concepts and methodologies by using Kermeta - an open source meta-modeling environment.

II. Querying Models with OCL

UML is well known for its extensive graphical notation and diagramming techniques. However, not all complicated designs can be simplified and fully expressed by pictures. There are cases where additional information needs to be captured in a different way. For this reason the UML includes the OCL, a textual language that allows a UML modeler to specify additional constrains and other requirements that sometimes graphical models are just not enough good for.

V. C. Nguyen is with Software Engineering research group at Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague - Karlovo náměstí 13, 121 35 Prague 2, Czech Republic (e-mail: nguyev1@fel.cvut.cz).

X. Qafmolla is with Software Engineering research group at Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague - Karlovo náměstí 13, 121 35 Prague 2, Czech Republic (e-mail: qafmox1@fel.cvut.cz).

A. *Preliminaries*

In the Object Management Group's (OMG) Model-driven Architecture (MDA), precise modeling and behavior of action, execution, query and transformation on models are essential. OCL is one of the approaches that can be used for this purpose. We often see OCL appear in an UML diagram or in the supporting documentation describing a diagram such as business rule definitions. However, this does not mean that OCL is strictly entitled with UML. We can also use OCL on non-UML diagram for the same purpose.

In terms of language category, OCL is a declarative language for describing rules that apply to a UML model. OCL was developed at IBM and now is a part of UML standard. Initially, OCL was only a formal specification language extension to UML. Now, as we mentioned, OCL can be used with any Meta Object Facility (MOF) [13] OMG's meta-model, including UML.

UML combined with OCL enables the resolution for many of the tasks that are required for MDA. OCL was originally viewed as a way to introduce constrains or to restrict certain values in a model. But moreover, OCL can also be used to support query expressions, derived values, conditions, business rules etc. OCL can express concepts that are not supported by UML diagrams, hence making models at all levels more precise. OCL can also support transformation tools and code generation as a key component to MDA.

B. *Case-study with OCL*

To illustrate this approach, we discuss an example of using OCL for supporting instance-level queries with UML. Some of the queries can be formed quite conveniently, while some more complicated queries may require some other techniques and ideas in order to be built successfully within OCL. Our example UML diagram for the case is as follow:

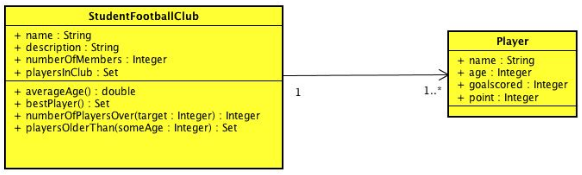

Fig. 1: Student Football Club models

Figure 1 depicts a class model diagram of a Student Football Club in a very basic way. Each club can have several players. These players are recorded with how many goals they have scored and certainly some personal information about them. Let's suppose we have the data of players as follow (instance-level): Four players are considered as members of the Club. Obviously each player has his own attributes. In our case-study, we are interested mainly in the number of goals that he scored. This leads to the number of points that he obtained accordingly. Notice that the player's data are presented in the diagram not in an UML way. The arrows only represent the relationship of a player with a club, basically showing he belongs to a club.

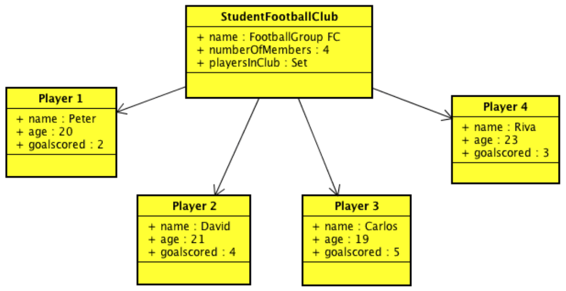

Fig. 2: Student Football Club sample data

We can use OCL to query data at instance-level. At first, we can derive some values for several functions in the models. For instance, let's assume that each player will get 5 points for each goal he scored. The attribute *point* can be derived in OCL as follow:

```
context Player::point: Integer
derive: 5*goalscored
```

To get the number of players in the club, we can derive as follow:

```
context
StudentFootballClub::numberOfMembers
                            :Integer
derive: players->size()
```

One convenient way to query data with OCL is to use the OCL *select()* query. Assuming we have to query for the number of players that have scored an amount of goals greater than a certain value:

```
context StudentFootballClub::
        numberOfPlayersOverTarget(
        target:Integer):Integer
body:   players->select(goalscored>target)
        ->size()
```

The OCL query above should return an integer value as the number of players having numbers of scored goals greater a value as *target*. If we need to query a collection, OCL can return the result also as a collection:

```
context StudentFootballClub::
        playersOlderThan(
        someAge:Integer):Set(Player)
body:   players->select(age>someAge)
```

This function returns a Set as result of the query. In general, OCL supports several ways to define collections [7]. OCL collections can be the one of the following:

- *Set:* no duplicates, not ordered.
- *OrderedSet:* just ordered set, not sorted.
- *Bag:* may contain duplicates, not ordered.
- *Sequence:* an ordered Bag.

This provides the facility for defining the result of querying. We know that some aggregate functions are often used in a query language such as Structured Query Language (SQL). In OCL this is not quite straight-ahead, but these functions still can be implemented. A query to find the average value of a collection can be implemented with OCL as follow:

```
context StudentFootballClub::averageAge()
                            : Real
body:   players.age->sum()/players->size()
```

This function returns the average age of all players in the Student Football Club as a real number. When translated into SQL this could simply be the query:

```
SELECT AVG(player_age) FROM Player;
```

We see that most Database Management Systems (DBMS) provide many built-in aggregate functions and they can be used quite conveniently for cases like this. With OCL, however we have to manually define our functions for such purpose. In the case of finding the average value, we had to get the sum of all players' age and divided by the number of players we have there:

```
players.age->sum()/players->size().
```

In the case of finding the maximal value, our sample is to find the best players which are the players that scored the highest number of goals. Some calculations need to be performed:

```
context StudentFootballClub::bestPlayer()
                             :Set(Player)
body: players->select(goalscored =
      players->sortedBy(goalscored)
                ->last().goalscored)
```

This OCL query required a sorting of all players according to their number of scored goals. After that, we did a projection to the list of all players with the condition where the number of scored goals equals to the maximal number of goals from that sorted list. Again, we see that this would have been a lot easier with a built-in aggregate function from a DBMS. This OCL query could be translated into SQL as simple as:

```
SELECT MAX(goals_cored) FROM Player;
```

With this example we demonstrate that OCL is quite handy in supporting several simple queries despite the fact that sometimes we have to manually implement some of the aggregate functions and manually manipulate data. However, we still do not see the support of OCL over more complex queries requiring, for example, join or product.

C. *OCL as a Query Language for Models*

We have seen examples on querying with OCL, nevertheless, the question of whether OCL can be really considered as a query language still remains. In order for OCL to be considered a fully expressive query language, we must be able to translate any SQL statement into an equivalent OCL expression. There are some certain obstacles to overcome in this direction for using OCL as a query language. OCL does not provide a facility for structured aggregation (such as a "struct" or a "tuple") and OCL is not expressive enough to define all of the operations required by a relational algebra and hence it does not form an adequate query language.

In terms of operations and functionality, OCL supports three (out of five) required relational algebra operations. Union, Difference, and Select are all supported by operations defined on the OCL collection types. However, the operation Product (or Cartesian Product) and Project are not supported, and cannot be supported as they directly require a facility for structured aggregation or a notion of tuples [7].

To overcome these obstacles, several research papers and study have emerged. Some of the authors introduce the method of combining UML and OCL together to form expressions with the functionality of relational algebra operators. One way to do that is to introduce the concept of a tuple, as we mentioned above. This idea is based on the UML concept of AssociationClass and n-ary Association. An instance of an Association is called a Link and an instance of an AssociationClass then can be used to represent a tuple of other objects. The relationship between the elements of the tuple is then expressed by the Link. With this notation we describe an indirect method for providing the functionality of the Product operator over a number of sets. We can start with three sets. After that, in a similar way, we can support the operation over n sets for any value of n.

By definition, the product of three sets S, T and U is the set of all tuples (s,t,u) such that $s \in S$, $t \in T$ and $u \in U$. Since OCL is a side-effect-free language, an OCL expression cannot create new objects. Thus, the result of the Product operation must be formed by selecting appropriate tuples from a set in which they already exist. The only way to ensure that such a set exists is to constrain the model as a whole [7]. First thing we have to deal with, is the size of the result of the Product operator. This can be expressed as a derivation from the multiplication of the associated sizes of the Sets. For instance with a model containing classes X, Y, Z, we introduce XYZ as an AssociationClass for representing the result of the Product, and the following constraint can define the size:

```
context XYZ
inv: XYZ.allInstances->size =
               X.allInstances->size*
               Y.allInstances->size*   (1)
               Z.allInstances->size
```

The second point is to prove the correctness of the constraint with the semantics of UML. As we use XYZ as a placeholder, this results in a model definition. In this model, the instances of class XYZ correspond to the Cartesian Product of the instances of X, Y and Z. These constraints can be expressed as:

```
XYZ.allInstances = X.allInstances x
                   Y.allInstances x     (2)
                   Z.allinstances
```

We need to verify that this is true with respect to our approach. The first thing we see is that the Left Hand Side (LHS) of (2) is a subset of the Right Hand Side (RHS). This can be seen from our notion of XYZ, each XYZ object is a LinkObject connecting X, Y and Z objects and a Link Object is equivalent to a tuple. It follows that all elements of the LHS are also elements of the RHS. In other words, LHS is a subset of RHS.

On the other hand, the UML standard states that "There are no two Links of the same Association which connect the same set of Instances in the same way". From the constraint (1) introduced above about the sizes, we know that the sets returned from LHS and the RHS have the same number of elements. Combining this with the fact that LHS is a subset of RHS, we can say that LHS equals to RHS in equation (2). A Cartesian Product between two arbitrary Sets of objects can be formed by selecting appropriate instances from the Set of allInstances of an AssociationClass defined in this way [7].

OCL and its tools [12] are proven to be a good tool for querying models, however, there are still several disadvantages including the fact that not many people know how to read OCL in models. This can somehow restrict the audience of the models. Another difficulty is that OCL statements are depicted on UML diagrams and this makes the models complex. Vaziri et al. from MIT Labs of Computer Science have concluded in their research that some shortcomings of OCL make it too implementation oriented to be a well suited language for the conceptual modeling paradigm [9]. The problematic aspect here is that OCL uses operations in constraints, which makes it stay closer to an

Object-oriented programming language rather than to a conceptual language. Also OCL expressions are claimed to be too verbose and sometimes hard to read. Finally, it is a matter of discussion that OCL is not a standalone language, but it needs UML class diagram for modeling. But as we have already shown, even though we often see OCL appear in an UML diagram or in the supporting documentation describing a diagram, this does not mean that OCL is strictly entitled with UML, but we can also use OCL on non-UML diagram for the same purpose. This makes OCL enough generic and independent.

To conclude, OCL supports three out of the five required relational algebra operations. Union, Difference, and Select are all supported directly by operations defined on the OCL collection types. The operations Product (or Cartesian Product) and Project are not directly supported. However, indirect methods that may be introduced based on the notion of tuple by using the instance of an AssociationClass, making a solution to this problem. There were several proposals on extending and aligning OCL to be a fully expressive query language. With these aligning on OCL itself we can build neat and accurate queries without modifying or adding more into the underlying UML models. OCL can become more effective as a query language in the future. For now, we can use OCL as long as all our needs for model querying can be fulfilled.

III. Meta-model Transformation with Kermeta

A. Introduction

In the recent years, business requirements are changing more rapidly and the complexity of enterprise applications is growing continuously. This makes it hard that any approach will perform well enough and deliver a software system of type one-size-fits-all. In this situation, the role of meta-modeling techniques is becoming just next to (instance) model querying, more and more significant in prototyping, business and workflow process design etc. Meta-modeling with its meta-model structure and flexible capabilities in different environments has provided a great help in solving and adapting to different problems by managing their complexity. In this section we give an introduction to the meta-modeling concepts and methodologies by using Kermeta - an open source meta-modeling environment. Kermeta has been designed as an extension to the meta-data language Essential MOF (EMOF), adding an action language for specifying semantics and behavior of meta-models.

A model can give us the view of attributes such as functionality, time constrains, security etc. In real life, models are developed through extensive communication among product managers, designers, and members of the development team. As the models approach completion, they enable the development of software and systems. A meta-model is yet another abstraction, highlighting properties of the model itself. A model conforms to its meta-model in the way that a computer program conforms to the grammar of the programming language in which it is written. Model-driven Engineering (MDE) with meta-models and automatic model transformation promises to bring productivity to software development.

Kermeta, as a language dedicated to meta-model engineering, was initiated in 2005 by the Triskell team of the Research Institute in Computer Science and Random Systems in France. The name Kermeta is an abbreviation for Kernel Meta-modeling, therefore implying that it is the basis of meta-modeling. It borrows concepts from languages such as MOF, OCL and Query/View/Transform (QVT) [14]. The model transformation part of Kermeta is also inspired by BasicMTL and previous experience with Model Transformation Language (MTL). Kermeta is an object-oriented language that provides support for classes and relations, multiple inheritance, late binding, static typing, class generality, exception, typed function objects. We can see that the object-oriented nature of Kermeta comes from both UML and the structural part based on also the syntax from object-oriented meta-data language EMOF [15].

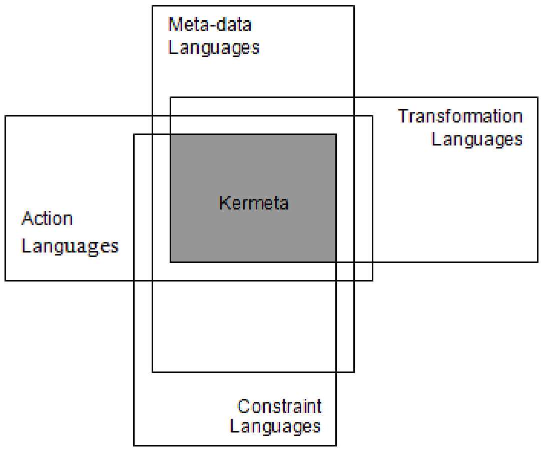

Fig. 3: Representation of Kermeta domain

Kermeta can be considered as a tool for general application of main model-oriented technologies. We found this fact to be useful in our work, especially in generative programming. We aim to do modeling and implementation of systems, not only by automatically generating it from specifications written in a Domain Specific Language (DSL), but also for its transformation in the context of requirements changes [8]. We see that definition of data, actions, constraints and transformations are being shared and they are now converging to the shape of a common denominator in MDE.

The Kermeta platform has an open-source framework of execution that is implemented as a plug-in to Eclipse, under the Eclipse Public License. Features of the Kermeta workbench are an interpreter and a debugger, a text editor with syntax highlighting and code auto-completion, a graphical editor and various import/export transformations. A model prototyper that will allow running the defined model using the behavior defined in Kermeta is under development [6].

B. *Case-study with Kermeta*

We will now briefly describe how we found Kermeta useful to implement meta-model transformation, by demonstrating it through an example. Starting with the modeling idea, we had a model of a conference in conceptual scheme, consisting of papers and committee members (persons), person being the class describing the authors of the papers too. During development phase, we went through the process of transformation of conceptual scheme to Relational Database Management System (RDMS) tables. We found that this process may be generic and common for various projects dealing with some implementation of a management system [10]. So we approach the usage of meta-models for this purpose. Our meta-models are classes (conceptual scheme) and tables (RDBMS). Kermeta provides tools that make this transformation process automatic.

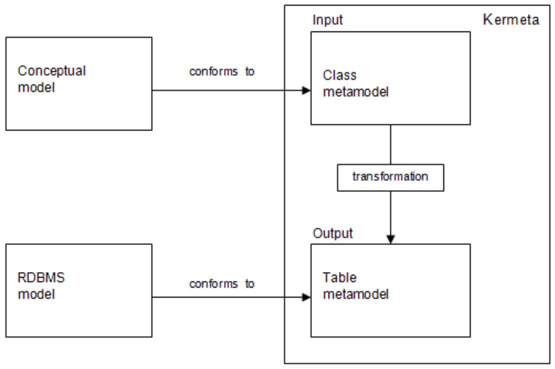

Fig. 4: Input and output meta-models

The first step is to provide input and output meta-models. Here is how the abstraction of a relational DBMS could be described in Kermeta language:

```
require kermeta
using kermeta::standard
class ForeignKey
{
reference references : Table
reference cols : Column[1..*]
}
class Column
{
attribute columnname : String
attribute type : String
}
class Table
{
attribute tablename : String
attribute columns : Column[1..*]
reference primarykey : Column[1..*]
attribute foreignkeys : ForeignKey[0..*]
}
class RDBMSModel
{
attribute table : Table[1..*]
}
```

To keep track and represent a 1-to-1 mapping between two types of objects we define a Trace class that is bound with the type of the source and target objects. Once the models are provided the next step would be to propose the transformation. The proposed transformation is to generate tables from the constant classes. Then we generate columns in the tables for the attributes and update foreign keys for the relationships. There are three steps:

1. Create tables from constant classes
2. Create columns for each attribute of constant classes and construct foreign keys, but we cannot initiate them until primary keys of the references tables are processed
3. Update the foreign keys

Using the keyword '*required* ' in Kermeta, input and output meta-models are imported and after that a class is created that performs the actual transformation. Kermeta gives us the possibility to execute the transformation by running it directly in Eclipse. This transformation demonstrates the ability of having execution paradigm of models in Kermeta.

As shown above, from practical viewpoints in our work we have found that static model definitions are sometimes not well enough fitted for their purpose. Therefore, presenting some layer of abstraction will increase effectiveness by making code (and spent time for preparing it) more useful and easily adaptable in the future. On the other hand, changes in requirements, platforms and sometimes even personnel require fast altering of existing artifacts at minimal costs. So the need of an automatization process is in place and indispensable. Languages like Kermeta that bring semantics and actions in the meta-model world, while keeping support for the smaller subdomains as its predecessors, seem to be proper for this rationale.

IV. Conclusions

In this paper we have discussed some techniques from the modeling paradigm, namely instance-model querying and meta-modeling transformation. We observed and explained the benefits of such approaches from the business viewpoint. In today's market environment, change is an integral part of all projects. Therefore we find the necessity introducing different level of abstractions for modeling. We have introduced the usage of OCL in the sense of instance-model processing and the approach for instance-model using OCL, which is becoming more and more popular in the recent years. We have illustrated such method with some practical examples. Even though from our perspective we find OCL very practical and useful, some limitations of this approach were also discussed and some possible solutions to them were pointed out. From a higher abstraction level we have also discussed meta-modeling techniques, namely the usage of Kermeta - an extension to the meta-data language EMOF with an action language for specifying semantics and behavior of meta-models and the possibility of transformation. From practical viewpoints in our work we have found that static model definitions are sometimes not well enough fitted for their purpose. Therefore, presenting some layer of abstraction will increase effectiveness by making code (and spent time for preparing it) more useful and easily adaptable in the future. On the other hand, changes in requirements, platforms and sometimes even personnel require fast altering of existing artifacts at minimal costs. So the need of an automatization process is in place and indispensable. Languages like Kermeta that bring semantics and actions in the meta-model world, while keeping support for the smaller sub-domains as its predecessors, seem to be proper for this rationale. Both discussed techniques are quite young and far from becoming an actual standard in the variety of existing tools available in the domains they belong to, therefore our research in this area will continue and our goal is to find objective improvements and further practical appliance of such techniques in every day working processes.

REFERENCES

[1] D. Roberts and R. Johnson, Evolve frameworks into domain-specific languages. In *3rd International Conference on Pattern Languages*, Allerton Park, Ill., September 1996.

[2] D.H.Akehurst and B.Bordbar, "On Querying UML data models with OCL," In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, 2001, pp. 91 – 103.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide," Addison Wesley Longman, 1993.

[4] J. Klein,"Weaving behavior into Metamodels with Kermeta, " Seminars notes, University of Luxembourg, 2008.

[5] J. Mendling, M. Nüttgens, "Exchanging EPC Business Process Models with EPML," In: Proc. of the *1st GI Workshop XML4BPM - XML Interchange Formats for Business Process Management* at Modellierung 2004, Marburg Germany, 2004, pp. 61-79.

[6] Kermeta Project documentation, https://www.kermeta.org/documents/, 2008.

[7] L. Mandel and M. V. Cengarle, On the Expressive Power of OCL; FM'99 - Formal Methods, *World Congress on Formal Methods in the Development of Computing Systems*, Toulouse, France, Springer LNCS 1708 (September 1999), pp 854 - 874.

[8] M. Gogolla and M. Richters, On Constraints and Queries in UML; *Proc. UML'97 Workshop* `The Unified Modeling Language - Technical Aspects and Applications' (1997).

[9] M. Vaziri and D. Jackson, Some shortcomings of OCL, the Object Constraint Language of UML, *MIT*, 1999.

[10] N. V. Cuong and X. Qafmolla, Meta-model Transformation with Kermeta, *in 13th International Conference OBJEKTY 2008 proceedings*, p. 109-116.

[11] Object Management Group, "Business process modeling notation," February 2006, http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf.

[12] Octopus: OCL Tool for Precise UML Specifications, official web-site. http://www.klasse.nl/ocl/octopus-intro.html.

[13] OMG's Meta Object Facility Specification, http://www.omg.org/spec/MOF/2.0/

[14] OMG's Revised submission for MOF 2.0 Query/View/Transformation, Object Management Group (QVTMerge Group), http://www.omg.org/cgi-bin/apps/doc?ad/2005-03-02, 2005.

[15] P. A. Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, and J. M. Jézéquel. "On executable meta-languages applied to model transformations," In *Model Transformations In Practice Workshop*, Montego Bay, Jamaica, October 2005.