A Proposed Engine Implementation Mechanism to Execute the Code of Relations Query/View/Transformation Language

Mohammed Abdalla Osman Mukhtar, Mohd Fadzil B. Hassan, Mohamed Nordin B. Zakaria
Department of Computer and Information Science
Universiti Teknologi PETRONAS (UTP)
Tronoh, MALAYSIA
Abofatima92@gmail.com, mfadzil_hassan@utp.com.my, nordinzakaria@utp.com.my

ABSTRACT

Depending on our best knowledge and despite of the well documentation of the transformation Language Query/View/Transformation QVT-Relation which is used as a tool in Model Driven Architecture (MDA) technique that provided by Object Management Group (OMG), there is still no specific description about the execution engine for the QVT-Relations (QVT-R) Transformation Language because the aim of MDA technique is to separate the system implementation from the system description. In this paper we will divide the QVT-R code to parts in order to provide a proposed description about the implementation (execution) model of QVT-Relation Language, first part is about the method of instantiation to generate instances of the target model using Warren’s Abstract Machine (WAM), and the second part is about the sequence of the code execution using the directed graph.

Keywords: MDA; QVT; WAM; XMI; Model Transformation; PIM; PSM.

1. INTRODUCTION

For the most of the computing industry, and a major standardize by the OMG, Model Driven Architecture becomes the preferred method to re-use program code. "MDA separates the fundamental logic behind a specification from the specifics of the particular middleware that implements it". Since 2001 MDA has been the driving rationale behind a large number of the OMG standards efforts.

To support the development of large software and complex systems, the OMG has proposed the Model Driven Architecture (MDA) [1] to provide architecture with which:

- Systems have capability to meet new requirements.
- The ability of harmonization for current, old, and new technologies.
- Business logic can be controlled and protected when using new technologies.
- Legacy systems are integrated and harmonized with new systems.

In MDA technique, models are used to describe all steps of development until generating a target platform, through providing systematic model transformation definition according to mapping specification and using of well-established model transformation language QVT.

Motivation

In model transformation it is necessary to define a set of transformations that allows converting source model (Platform Independent Model [PIM]) that describes one view of the system into the model (Platform Specific Model [PSM]) that corresponds to another view of the same system [2]. The language which is the most used for this purpose is Relations QVT. Depending on our best knowledge there is no specific documentation about how the engine for this language can execute the code. For that we will try to provide a preliminary proposal overview about the engine of Relation QVT.

2. MODEL-DRIVEN ARCHITECTURE & METAMODELS

A. MDA

At the end of the year 2000, the OMG proposed a rapid change from OMA and interpretative approaches to MDA and transformational approaches (generating model from other models). Transformation can be either one direction from PIM instances to PSM instances, or bi-directional.

The models are executed whether by interpretation or code generation, and this execution can be possible only if there is a good behaviour representation, but there are widely differing views on how behaviour should be represented in a model [3].

MDA provides architecture to solve the complexity of software development and maintenance which has no precedents. It claims to create and maintain software artifacts with little effort from system modeller and software developers. However, there are some issues such as mapping specification and transformation definition in MDA approach need solutions [4]. Figure 1 defines the MDA steps towards the final code with effort as little as possible.

Not all ideas provided by the MDA approach are new. For example, the generation of code from a model, the database domain has somehow model transformation (e.g. transformation from entity-relationship to relational-
tables and SQL schema). However, the standardization of an approach based on models to enable the development and maintenance of software systems is a big advance in software engineering.

However, this does not mean the end of the former, but the introduction of models as a supplementary layer to address the development of complex and large software systems. In fact, models are the top layer and the other paradigms are the bottom layer in the MDA approach.

**B. Metamodels**

A metamodel is a conceptual model for the syntax of a modelling system. The syntax specified can be either concrete (if the metamodel tells the designer how to create a model instance) or abstract (if the metamodel does not tell the designer how to create a model instance). The metamodel also specifies a schema for a repository. A repository stores model instances. Constraints can be expressed as queries on the repository. A repository supports a CASE (Computer-Aided Software Engineering) tool which performs model creation, editing, rendering, browsing etc. A metamodel often has both a class’s model and an instances model.

A conceptual modelling language used to represent metamodels is called metametamodel. There are many metametamodels. A metametamodel commonly used in programming languages is Bachus-Naur Form or BNF to specify the language syntax [6]. For the modelling languages MOF (Meta-Object Facility) is the most used to specify other modelling language (like UML), the Object Management Group (OMG) specifies its standards.

![Figure 1. Creating Model-Driven Architecture (MDA) Systems](image)

Despite of several case studies have demonstrated that MDA is a potential solution and the future for developing software systems, we cannot yet assess that the MDA approach will give us the solution for the all problems in software system development because some issues are still under research such as mapping, transformation, semantic distance, traceability, and so on [5].

![Figure 2. 4-Layers Architecture of MDA](image)

Referring to figure 2 we can divide MDA technique into four layers, each layer on the top depends on the other layer in the bottom and these four layers are metametamodel, metamodel, model, and information (on the real world).

In level M0, an operational example of a model is the final representation of a software system. In level M1, a model is a well-formed specification for creating software artefacts. In level M2, a metamodel is a well-formed specification for creating models. In level M3, a metametamodel is a well-formed specification for creating metamodels.

According to this architecture, there is a few metametamodels such as MOF [7] and Ecore [8], several metamodels such as UML, UEML [2] and EDOC [9], more real life description models such as a booking system, and finally infinite information such as the implementation of this booking system model using Java. Here, it is important to pay attention to the existence of several metamodel languages, providing a Domain-Specific Language [10] or a general-purpose language (e.g. UML). In fact, the four layers are models. However, it is important to understand that each model level achieves a different goal in software development.

### 3. QVT-R

#### A. Introduction

QVT (Query/Views/Transformation) is the OMG standard language for specifying model transformations in the context of MDA. It is regarded as one of the most important standards since model transformations are proposed as major operations for manipulating models [11].

The three concepts that are used in the name of the QVT language as defined by OMG documents are: [12]

- **Query**: A query is an expression that is evaluated over a model. The result of a query is one or more instances of types defined in the source model, or defined by the query language.
• **View:** A view is a model which is completely derived from another model (the base model). There is a ‘live’ connection between the view and the base model.

• **Transformation:** A model transformation is a process of automatic generation of a target model from a source model, according to a transformation definition.

QVT languages are arranged in a layered architecture shown in Figure 3. The languages Relations and Core are declarative languages at two different levels of abstraction.

The specification document defines their concrete textual syntax and abstract syntax. In addition, Relations language has a graphical syntax. Operational Mappings is an imperative language that extends Relations and Core languages. Relations language provides capabilities for specifying transformations as a set of relations among models. Core language is a declarative language that is simpler than the Relations language. One purpose of the Core language is to provide the basis for specifying the semantics of the Relations language. The semantics of the Relations language is given as a transformation RelationsToCore. This transformation may be written in the Relations language.

Sometimes it is difficult to provide a complete declarative solution to a given transformation problem. To address this issue the QVT proposes two mechanisms for extending the declarative languages Relations and Core: a third language called Operational Mappings and a mechanism for invoking transformation functionality implemented in an arbitrary language (Black Box implementation).

**B. QVT-R Metamodel**

Metamodel is used to describe the grammar of the language. In figure 5 we can see the main and essential part of QVT-R metamodel.

![Figure 5: The Basic Metamodel of QVT-R][13]

It has a root tag named `RelationTransformation` with composition-relationship to Relation and `TypedModel` Instances. The `TypedModel` specifies the `ModelParameters`, which define the used models for model transformation. Instances of Relation have a Boolean attribute `TopLevel` that refers to the top level property. When and `Where` clauses are used to specify which `Pattern` can be used because `Relation` may have more than one pattern, and this pattern can hold several `Predicates` to call referred `Relations`, and with each call there is an instance predicate added to that pattern. The call is executed using the type `RelationCallExp` which is a subtype of `OclExpression`. In this paper we are focusing on these predicates because used `Relation` to create target model instances must be called by specific predicate which included in `When` or `Where` clauses.

**4. XML METADATA INTERCHANGE (XMI)**

To achieve a complete conversion there must be a complete definition of the source model to the goal of transformation between the model rules, the transformation rules are common, there is no need actual conversion to be as a model. But it is better to create a mapping model, which is the model required to define the relationship between the source and target model members, and it is providing transformation rules and standards [13]. MDA provides a method of mapping two models: the type of mapping and example of mapping. Depending on the two mapping methods, MDA gives two basic method of model conversion: the type of the Mapping model conversion methods and the model example converting mapping methods.

XML is a meta language, the users can be used to create their own need and other tag language, which makes...
the XML application to quickly introduce to the various domain. XML allows users defined instances, defined examples and marked their property and the use of tags and attributes to make an example. XML Schema is that source model, which is based on the XMI generation, examples of the model is based on XML and XMI document conversion.

XMI is based on the XML metadata exchange, XMI specification provides a standard method that mapping of the object model and the example model become XML. Through the application of XMI standards, which can complete UML model elements to the XML mapping. XMI is a standard that the UML model definition automatically generated XML DTD and Schema.

Based on the previous tag language, the object and other objects associate with the existence of many difficulties, which can be resolved in the XMI. In addition, XMI based on the characteristics of XML, that means metadata and examples of their elements content can coexist in the same document. It allows applications that can be easily passed its meta-data to understand the examples. At the same time, XMI self-description nature and characteristics of synchronization, which is why XMI based on the exchange the models, for that it is so important in the distributed and heterogeneous systems.

XMI comply with standard rules to generate, the MOF model example generates DTD or Schema, the definition of the rules of generation, UML model example converting to XML documents [14].

5. How QVT-R Code can be executed?

To write your own QVT code you have to create your own Platform Independent Model (PIM) as source model, beside all instances for this PIM model (this is the main drawback for QVT that you have to provide the instance model along with your PIM model), also you must have Platform Specific Model (PSM) as target model. Figure 7 shows briefed description for the prerequisite steps to create QVT-R code and the output (resulting model) after executing that code, whereas figure 8. defines small case study to transform PIM (Package) to PSM (Schema) [4]. Firstly we need to specify the mapping transformation between each element from PIM with another element from PSM.

![Figure 6: Model Based on XMI Mapping Ruler to Generate XML Schema](image)

![Figure 7: Prerequisite steps to create QVT-R code and the resulting Model](image)

![Figure 8: Model Transformation from PIM (Source Model) to PSM (Target Model)](image)

After that there must a repository to store all these models along with the generated target model, and it will be XML file in this case. Now figure 9 gives a conception about the engine of Relation QVT and how it is working to generate the instance model of PSM corresponding to the instance model of PIM, which is already exist.

A. Code Execution Sequence

We can represent the execution sequence for QVT-R code as a directed graph and try to cast each relation to one node of that graph. To create like this graph an entry point has to be found, which called root node.

As we can see in figure 10(a), QVT-R introduces two different types of relations (top relations and non-top relations). These types of relation control the direction of QVT-R code execution. There are two subtypes from non-top relations, first one is when-clause relation (without precondition) and the second one is where-clause relation (with precondition).
A transformation can be invoked either to check two models for consistency or to modify one model to enforce consistency.

Two domains are declared, each domain specifies a simple pattern (a package with a name, and a schema with a name). The same property for the two patterns is bounded to the same variable 'Pn', that means they should have the same value.

The when clause specifies the conditions under which the relationship needs to hold. The where clause specifies the condition that must be specified by all model elements participating in the relation.

Figure 9: Brief Conception about Relation QVT Code

Figure 10. (a) Directed Relation Graph for the QVT-R Code (b) Example

Now we can use Depth-First Search (DFS) algorithm to find appropriate relation will be executed during QVT-R code running time (Figure 11).

DFS(s):
for each vertex u ∈ V
do color[u] := White ; not visited
Time := 1 ; time stamp
for each vertex u ∈ V
do if ... DFS-Visit(v,time)
color[u] := Black
f[u] := time := time+1 ; f=finish time

DFS-Visit(u,time):
color[u] := Gray ; in progress nodes
d[u] := time ; d=discover time
time := time+1 ; time timestamps
for each v ∈ Adj[u]
do if color[v]=White
then DFS-Visit(v,time)
color[u] := Black
f[u] := time := time+1 ; f=finish time

Figure 11: Directed Graph [Depth-First Search (DFS) Algorithm]

A. Instantiation

The Warren Abstract Machine (WAM) is a language and machine architecture intermediate between Prolog and underlying computer. A Prolog program is transformed into WAM instructions by a compiler, and the resulting WAM code is either executed by bytecode emulator or further translated to machine instructions.

WAM has several areas of memory and a number of registers. The memory areas are code area, control stack, copy stack, trail, and unification work area [15]. There are two types of registers, control registers and unification registers. The latter are described in the section on unification below. The control registers are Program pointer, Continuation pointer, Variable-binding frame for clause, Latest backtrack point, Top of control stack, Top of trail, Top of copy stack, and Copy stack backtrack pointer.

We find that QVT-R transformation language looks like Prolog language in many aspects. For that the appropriate machine that can hold the responsibility to execute the code of QVT-R is Warren’s Abstract Machine. The WAM is a quite efficient QVT-R execution engine.

6. CONCLUSION

The significance of this research is to provide a solid understandability for the Relation QVT implementation (execution) model, which can be simulated using different platform.

In this paper we try to proposed a mechanism to illustrate what happen behind the Relation QVT model transformation language interface (The QVT-R engine) because this language becomes widely used and it has a good documentation and it could – in few years – to attract and gaining high interesting from the researchers.
REFERENCES