International Journal of Computer Applications (0975 – 8887)
Volume 15– No.7, February 2011

# A Proposed Compiler to Integrate Model Driven Architecture with Web Services – Road Map

Mohammed Abdalla Osman Mukhtar
Azween Abdullah, Alan G. Downe
Department of Information and Computer Science
Universiti Teknologi PETRONAS
Tronoh, MALAYSIA

## ABSTRACT

Model Driven Architecture (MDA) technique is mainly depend on two processes; mapping specification and transformation definition, this last one (sometimes called model transformation definition) has a tool to implement, which the most used one is Query/View/Transformation (QVT) Relational Language. This language sponsored by Object Management Group (OMG), and we found that it needs a repository for the both target model and source model, that mean these two models must be exist before execute the transformation definition (QVT Code). To eliminate this repository problem we propose in this paper a solution for this problem that we can compile the QVT code into Business Process Execution Language for Web Services (BPEL4WS). This solution will provide valuable contribution especially for MDA infrastructure, firstly to eliminate repository requirements for target model, secondly to make model transformation as web services.

## General Terms

Model Driven Architecture; Model Transformation; Integration of Web Services with other Techniques.

## Keywords

MDA; PIM; PSM; QVT; BPEL4WS.

## 1. INTRODUCTION

If we need to make model transformation in MDA to be as web service, we need first to understand how the selected model transformation language (QVT in this case) has worked. As we found it is depend on Warren's Abstract Machine, which is consider as foundation base for prolog language. In contrast we need to choose appropriate web service language to be the target language while using QVT as source language. Depending on some criterion we selected Business Process Execution Language for Web Service (BPEL4WS) or BPEL as target language.

## 2. QVT RELATIONAL LANGUAGE

QVT 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 [1]. The three concepts that are used in the name of the QVT language as defined by OMG documents are: [2]

***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 1 [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).

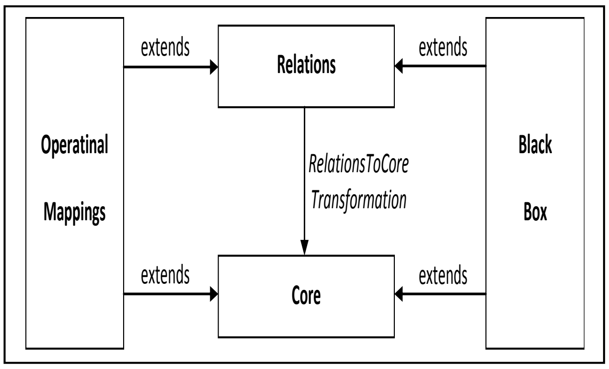

**Fig 1: Layered Architecture of QVT Languages**

Depending on the four layer approach that provided by MDA (figure 2), we can imagine that we are working on level1 (M1) or level2 (M2), and the mapping specification will be done from PSM (QVT code) to PSM (BPEL4WS code) with the same level as a refinement in this case.

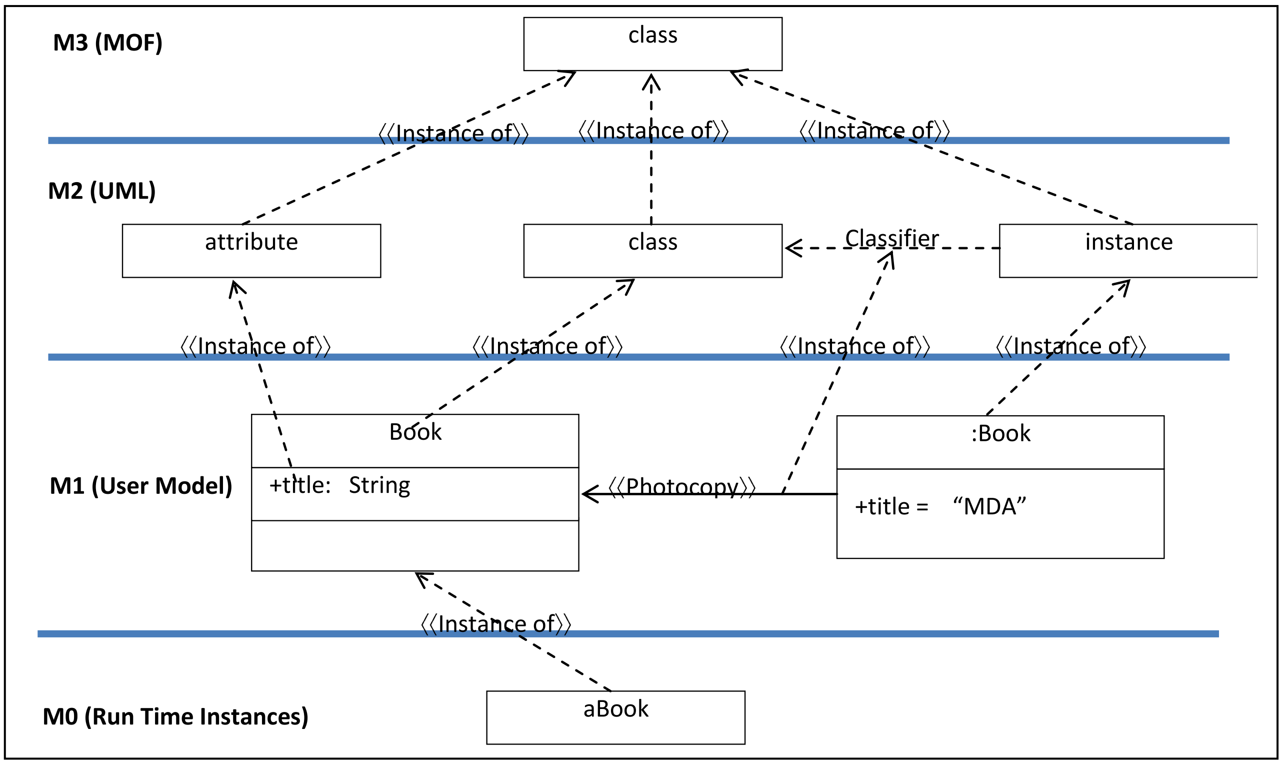

**Fig 2: 4-Layers Architecture of MDA**

If we need to move in deep to understand how QVT Relational Language is working, we need firstly to know the state-of-art for Warren's Abstract Machine, which the Prolog Language has been built on, and we know that QVT is prolog-like. Secondly we will discuss Deductive Database because QVT can be seen as it, finally we will display the repository to exchange models between MDA tools XML Metadata Interchange (XMI) because QVT absolutely need for XMI to store and retrieve models.

## 2.1 Warren's Abstract Machine

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 [4]. We find that QVT Relational Language is based on prolog liked code, that mean it is based indirectly on WAM.

## 2.2 Deductive Database

QVT can be seen as deductive database. The theory of deductive database is generally expressed in the relational data model. Permitted operations are select, project and join. Negation and recursion are allowed. A deductive database starts with a collection of instances of predicates (extensional database, or EDB) and a set of rules for creating instances of other predicates (the rules are the intentional database [IDB]). The instances created are the model of the IDB. One of the consequences of this is that the rules can be executed in any order. Those rules whose precondition is not satisfied will not create anything (will not fire). The IDB is executed systematically, possibly several times. Execution can stop when no rule can create any new bindings. This execution property depends on having no negative predicates in the IDB [5]. QVT does not in fact allow negative predicates. Both the *checkonly* and enforce clauses are positive. Negation is not allowed in the when clauses. Recursion is allowed. It is possible for a *checkonly* clause to test predicates that are created in enforce clauses. QVT can compute transitive closures. This means that care must be taken when translating a QVT transformation into a procedural workflow (like BPEL4WS).

## 2.3 Extensible Markup Language (XML) and 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. Mapping model is the model required for the conversion of model members in the definition of the relationship between the mapping, which provides transformation rules and standards. 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 user 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 [6].

QVT relational language needs a repository to render instances of both source model and target model, which must be with XMI format.

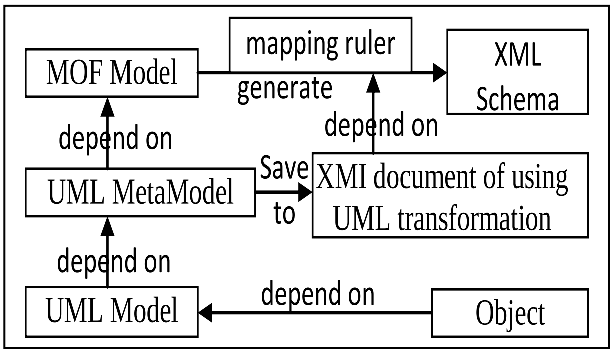

**Fig 3: Model Based on XMI Mapping Ruler to Generate XML Schema**

## 3. MODEL TRANSFORMATION

Development of two models instead of one (as in standard approaches) implies supplementary effort and thus reduces MDA efficiency. This is a serious weakness, which can overbalance all advantages of this approach. However, MDA developers stress that, after a PIM has been constructed, the process of platform-dependent system development can essentially be automated [7]. These two models present the same system and, consequently, have much in common. To obtain a PSM, one should describe in what way different constructions of the system are mapped from one model to another. Hence, it is necessary to define a set of transformations that allows converting a source model that corresponds to one view of the system into the model that corresponds to another view of the same system. Each of these views consists of its own set of notions and elements that are typical of specific technological platform. If model transformation is described in some formal language and there exists an algorithm for its automated execution, then it is called model transformation definition [8].

To understand model transformation very well, let's consider we have model a (*Ma*) and another model b (*Mb*). For each model there must be an existent metamodel compatible with MOF [9], that mean we must have metamodel a (*MMa*) and metamodel b (*MMb*). Now we must consider that (if we need to transform *Ma* → *Mb*) the transformation from model a to model b is model itself (we can call *Mt*), and we can formulate this like *Mt*: *Ma→Mb*, which Mt is a model written in the same language of Ma and Mb metamodel. Obviously since Mt is a model, we postulate the existent of a generic transformation metamodel *MMt*, which would similar to any other MOF based MDA metamodel [10]. For that we find that QVT as a model transformation language thus it has a metamodel compatible with MOF. Figure 4 try to explain the definition of model transformation from the perspective of metamodel, and the place of transformation language (QVT in this case) compared with the concept of 4-layers Architecture of MDA.

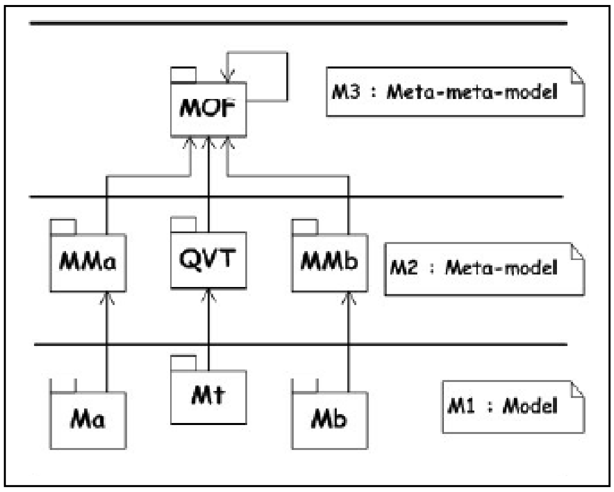

**Fig 4: The Place of QVT Transformation Language [10]**

## 4. BPEL4WS

Web services provide the basic technical platform required for application interoperability. They do not, however, provide higher level control, such as which web services need to be invoked, which operations should be called and in what sequence. Nor do they provide ways to describe the semantics of interfaces, the workflows, or e-business processes. Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) [11] is the missing link to assemble and integrate web services into a real business process BPEL4WS standardizes process automation between web services. This applies both within the enterprise, where BPEL4WS is used to integrate previously isolated systems, and between enterprises, where BPEL4WS enables easier and more effective integration with business partners (see Figure 5). In providing a standard descriptive structure BPEL4WS enables enterprises to define their business processes during the design phase. Wider business benefits can flow from this through business process optimization, reengineering, and the selection of most appropriate processes. Supported by major vendors - including BEA, Hewlett-Packard, IBM, Microsoft, Novell, Oracle, SAP, Sun, and others - BPEL4WS is becoming the accepted standard for business process management [12].

BPEL allows specifying business processes and how they relate to Web services. This includes specifying how a business process makes use of Web services to achieve its goal, and it includes specifying Web services that are provided by a business process. Business processes specified in BPEL are fully executable and they are portable between BPEL conformant environments. A BPEL business process

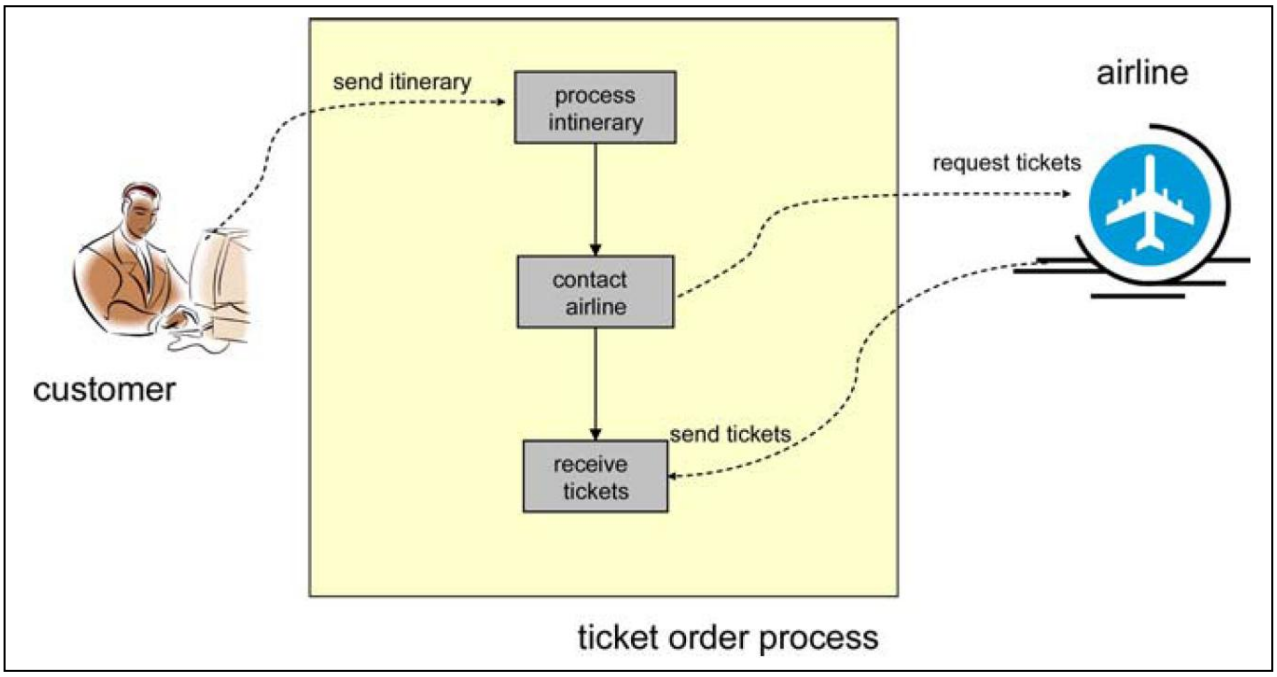

**Fig 5: The Basic Elements of BPEL**

interoperates with the Web services of its partners, whether these Web services are realized based on BPEL or not. Finally, BPEL supports the specification of business protocols between partners and views on complex internal business processes [13]. As any executable code language, BPEL has executable code model, figure 6 shows general abstract syntax of executable code models.

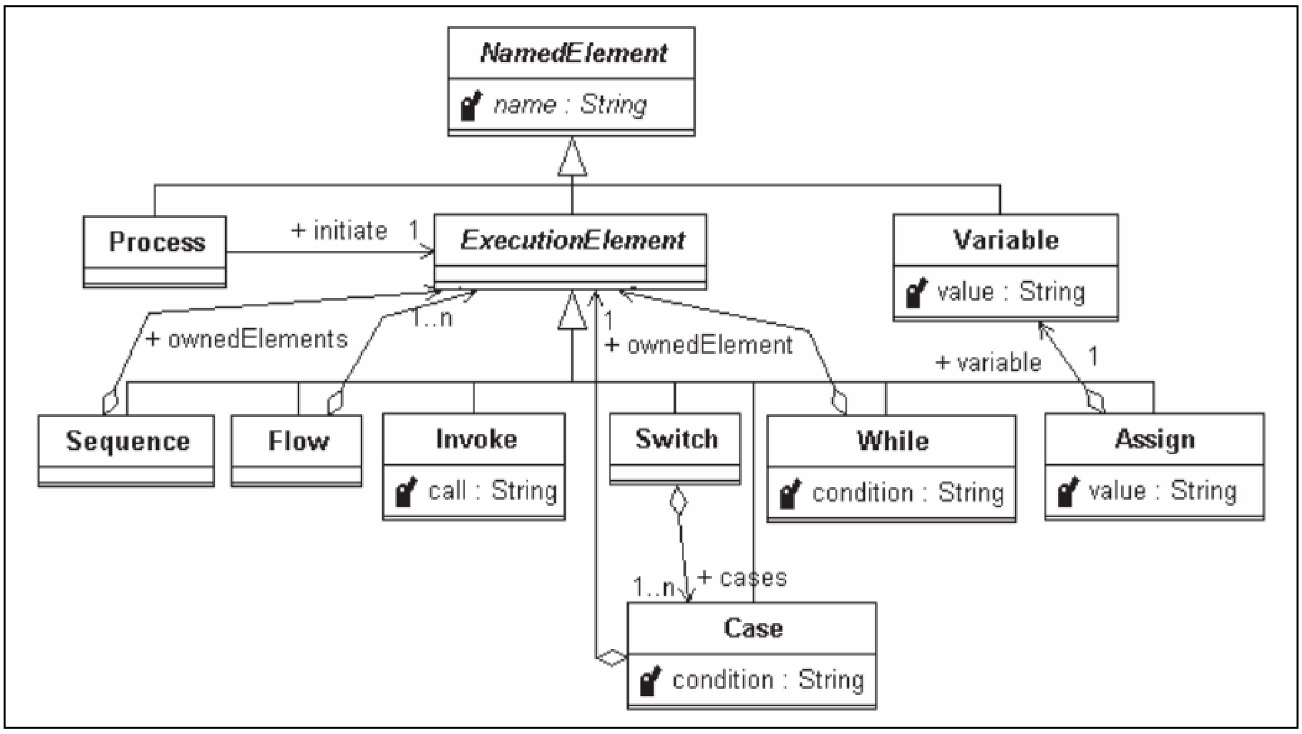

**Fig 6: Abstract Syntax of Executable Code Models**

## 5. THE SCENARIO

We will consider how we can couple PIM to PSM and using QVT Relational Language to implement model transformation between two instance models, before that we must have mapping specification to specify what is correspondence for members of PIM and PSM. Finally (as a main contribution) we need to compile the QVT code to BPEL code (like in figure 7).

There is another direction (Direction 1)to act this scenario make model transformation from one of UML diagrams (class, activity, or sequence diagram) as PIM to Business Process Model Notation (BPMN) as PSM and try to find standard tool to generate BPEL code directly. But in this direction there will be additional component that becomes an overhead for the process as whole. Figure 8 show these two directions.

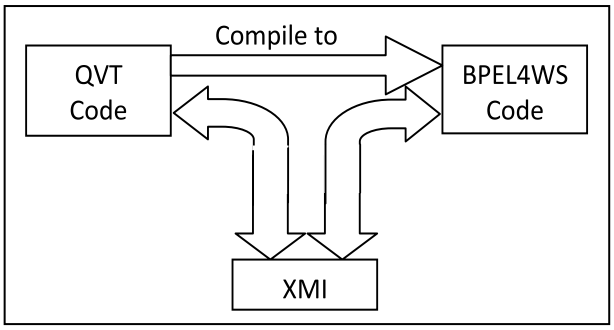

Fig 7: Compile from QVT to BPEL Using XMI Repository

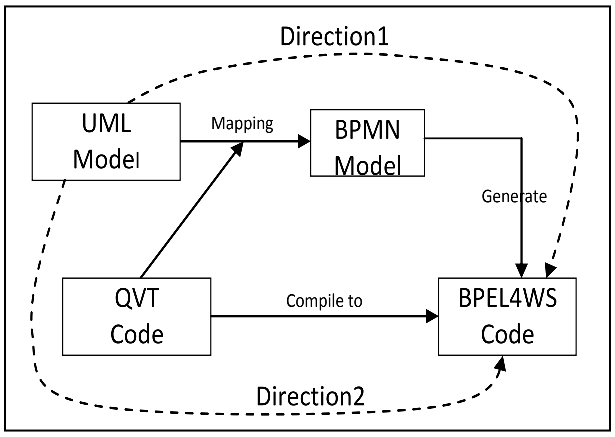

Fig 8: Two Directions to Generate BPEL Code

## 6. CASE STUDY

We want to build Graph Editor (GE), which will allow us to specify a Simple State Machine (STM) [15]. GE will be sufficient to develop the design shown in Figure 9. The design is of an application in which Patients buy treatment card at the time they obtain a location on reservation list. When they are going to the pharmacy they will buy medication as it has been described in the prescription. Due to do more examinations, patients may be rescheduled on reservation waiting list.

### 6.1 Step 1

Create the metamodel of STM as PIM (figure 10).

### 6.2 Step 2

Create the metamodel of GE as PSM (figure 11).

### 6.3 Step 3

Specify the mapping between PIM metamodel and PSM metamodel.

**Table 1. Mapping Specification Between PIM & PSM**

| PIM | PSM |
|---|---|
| State | Node |
| Transition | ArcEnd, which will been specified by EndNotation attribute ('Simple' if it's source node, or 'None' if it's target) |
| Event | Arc |

### 6.4 Step 4

Write QVT code to implement transformation from STM metamodel to GE metamodel (one direction).

```
transformation STMGE(STM:PIM_STM, GE:GE_PSM){
        key State {name};
        key Transition {source, target,
                        triggeredBy};
        key Event {name};
        key Node {nodeText};
        key ArcEnd{arc};
        key Arc {source, target};
top relation STMToGraph {
        n : String;
        checkonly domain STM s :PIM_STM::STM
                            {name = n};
        enforce domain GE g :GE_PSM::Graph
                         {name = n};
}
top relation StateToNode{
        n, ns : String;
        checkonly domain STM s:PIM_STM::State{
        stm = st : PIM_STM::STM {name = n},
```

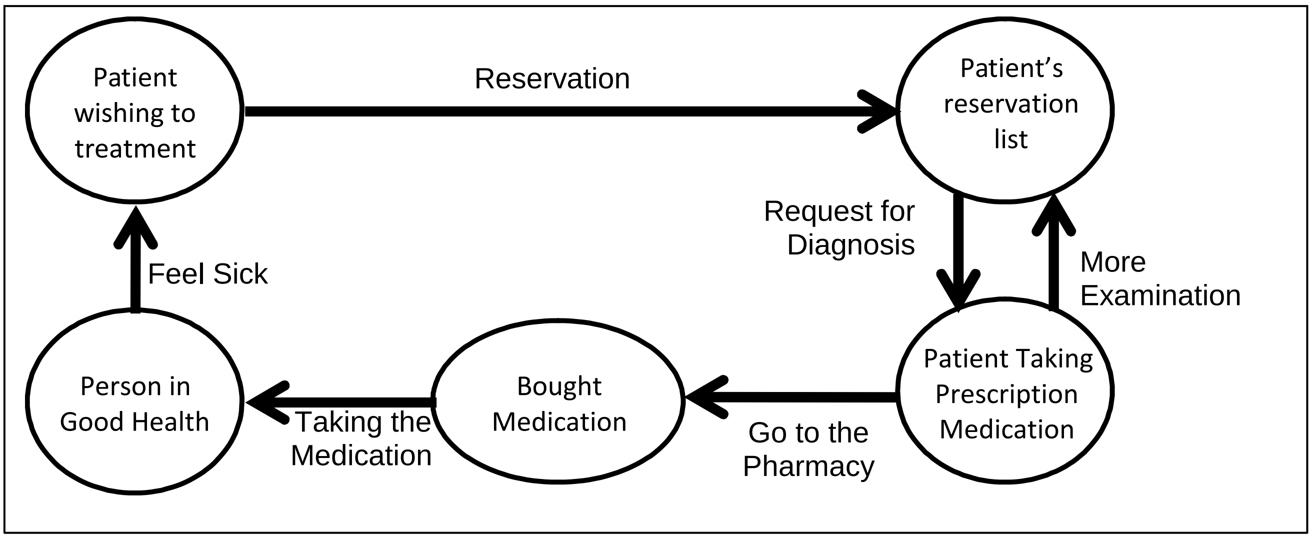

Fig 9: Simple State Machine to Describe Patient Treatment Design

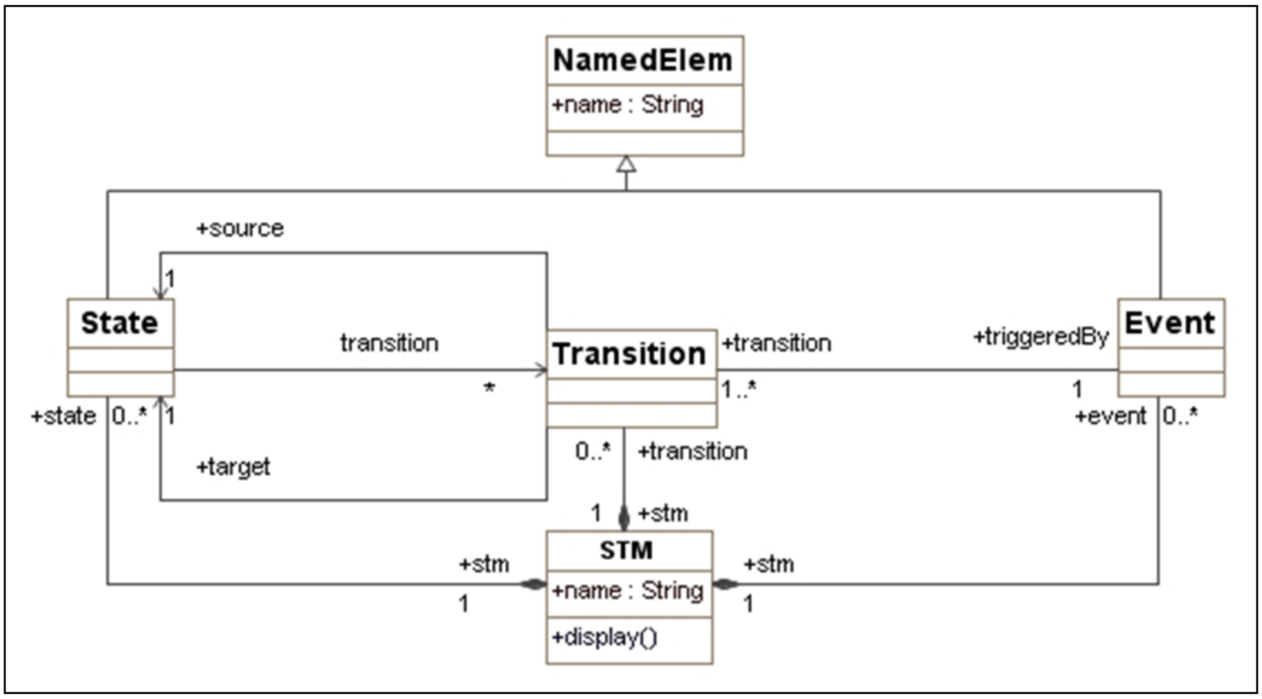

**Fig 10: STM Metamodel (PIM)**

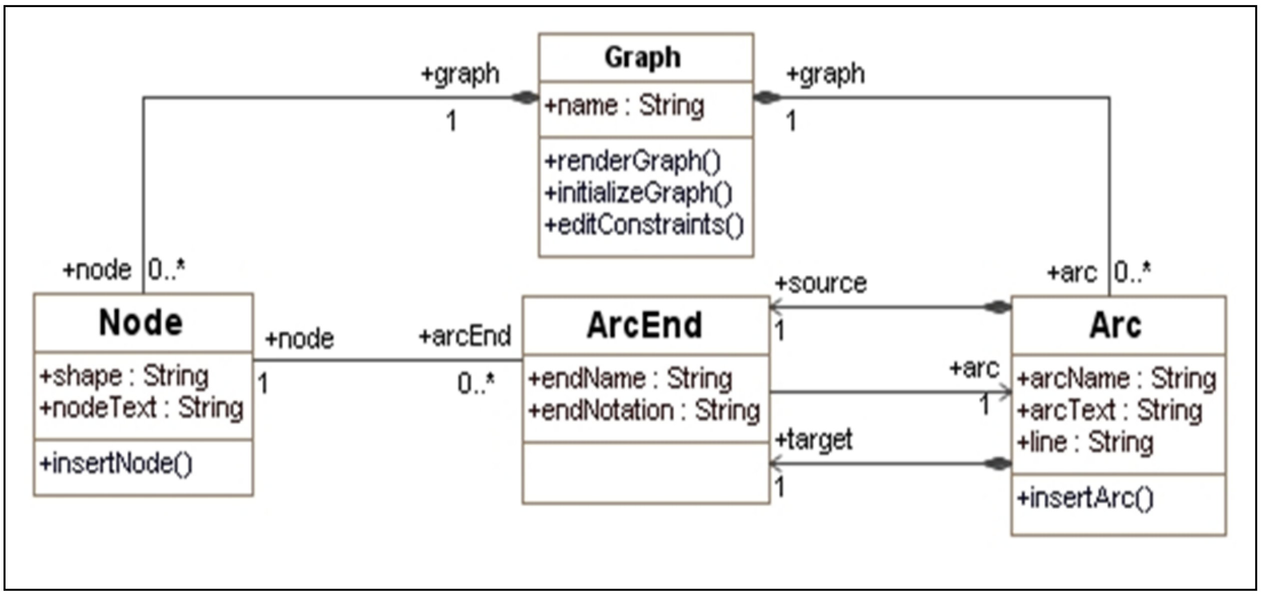

**Fig 11: GE Metamodel (PSM)**

```
        name = ns};
        enforce domain GE nd : GE_PSM::Node{
        graph = gr : GE_PSM::Graph {name = n},
        shape = 'circle', nodeText = ns
        };
        when {
        STMToGraph (st, gr);
        }
}
top relation TransitionToArc{
        ng : String;
        checkonly domain STM s:
                        PIM_STM::Transition{
        stm = st : PIM_STM::STM {},
        source = so : PIM_STM::State {},
        target = ta : PIM_STM::State{}
        };
        enforce domain GE a : GE_PSM::Arc{             }
```

```
graph = gr :GE_PSM::Graph {},
line = 'thin',
source = sr : GE_PSM::ArcEnd {
arrowHead = 'none',
node = no : GE_PSM::Node {}
--arc = a : GE_PSM::Arc {}
},
target = tr : GE_PSM::ArcEnd {
        arrowHead = 'simple',
        node = nta : GE_PSM::Node{}
        arc = a : GE_PSM::Arc {}
}
};
when {
STMToGraph (st, gr);
StateToNode(so, no);
StateToNode(ta, nta);
}
```

```
top relation EventToArc{
        n : String;
        checkonly domain STM tr :
                           PIM_STM::Transition{
        triggeredBy = e:PIM_STM::Event{name=n}
        enforce domain GE a:
                       GE_PSM::Arc{arcText=n};
        when {
        TransitionToArc(tr, a);
        }
        }
}
```

### 6.5 Last Step

Apply the compiler program to compile QVT code to BPEL code.

## 7. CONCLUSION

In this paper we propose an optimized method to integrate model transformation of MDA with web services when compiling QVT Relational Language code to BPEL4WS code, and if this work is done perfectly, we sure that it will be a good contribution to make a valuable changing in MDA infrastructure. Also it can open a new direction to apply other programming concepts beside compiler design concept.

## 8. REFERENCES

[1] G. Wachsmuth, “Modelling the Operational Semantics of Domain-Specific Modelling Languages,” Structure, 2008, pp. 506-520.

[2] I. Kurtev, “State of the Art of QVT : A Model Transformation Language Standard,” Data Engineering, 2008, pp. 377-393.

[3] M.A.O. Mukhtar and A. Abdullah, “Mapping of Behavior Model using Model-Driven Architecture,” International Journal, vol. 13, 2011, pp. 35-39.

[4] C. Beierle, “Logic programming with typed unification and its realization on an abstract machine,” IBM Journal of Research and Development, vol. 36, 2010, pp. 375-390.

[5] S. Mallet and M. Ducass, “Myrtle : A Set-Oriented Meta-Interpreter Driven by a ‘ Relational ’ Trace for Deductive Databases Debugging,” 1999, pp. 328-330.

[6] Y. Xiao-mei, “Mapping Approach for Model Transformation of MDA based on XMI / XML Platform,” Methods, 2009.

[7] M.B. Kuznetsov, “UML model transformation and its application to MDA technology,” Programming and Computer Software, vol. 33, Feb. 2007, pp. 44-53.

[8] X. Qafmolla, “Automation of Web Services Development Using Model Driven Techniques,” Architecture, vol. 3, 2010, pp. 190-194.

[9] O.M.G. (OMG), “Meta Object Facility ( MOF ) Core Specification,” Management, 2006.

[10] A. Evans and J.S. Willans, “Metamodelling for MDA (First International Workshop) Proceeding,” Language, 2003.

[11] T. Andrews, F. Curbera, Y. Goland, and D. Roller, Business Process Execution Language for Web Services, 2003.

[12] M.B. Juric, Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition "Abstract", Packt Publishing, 2006.

[13] F. Leymann and D. Roller, “Modeling business processes with BPEL4WS,” Information Systems and e-Business Management, vol. 4, Nov. 2005, pp. 265-284.

[14] R. Hauser and J. Koehler, “Compiling Process Graphs into Executable Code,” 2004, pp. 317-336.

[15] R.M. Colomb, “Developing Methods for Using Model-Driven Architecture to Develop Quality Software Products at Low Cost Entirely by Re-Use of Existing Components,” vol. Funded Pro, 2007, p. 15.