An Extended Model Driven Framework for End-to-End Consistent Model Transformation

Mr. G. Ramesh
Research Scholar in CSE, JNTUA College of Engineering
Ananthapuramu, Andharapradesh, India-515002
ramesh680@gmail.com

Dr. T. V. Rajini Kanth
Professor in CSE, SNIST, Ghatkesar, Hyderabad, Telangana State, India.
rajinitv@gmail.com

Dr. A. Ananda Rao
Professor in CSE & DAP, JNT University, Anantapur
Ananthapuramu, Andharapradesh, India-515002
akepogu@gmail.com

Abstract - Model Driven Development (MDD) results in quick transformation from models to corresponding systems. Forward engineering features of modelling tools can help in generating source code from models. To build a robust system it is important to have consistency checking in the design models and the same between design model and the transformed implementation. Our framework named as Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) proposed in our previous papers supports consistency checking in design models. This paper focuses on automatic model transformation. An algorithm and defined transformation rules for model transformation from UML class diagram to ERD and SQL are being proposed. The model transformation bestows many advantages such as reducing cost of development, improving quality, enhancing productivity and leveraging customer satisfaction. Proposed framework has been enhanced to ensure that the transformed implementations conform to their model counterparts besides checking end-to-end consistency.

Key words: Model Driven Development, model transformation, XRTSDIC framework, end-to-end consistency checking.

I. Introduction

Model refers to a representation that is part of Software Development Life Cycle (SDLC) which is based on Model Driven Development (MDD). In the context of MDD, there are different models as presented in Figure 1. Each model represents a system differently for different purposes. However, in the wake of Computer Aided Software Engineering (CASE) it is desirable to have tools to build models with ease. Then it is desirable to have consistency checking mechanisms to ensure that the models are consistent. It is also desirable to have model transformations from one model to another model. In fact model driven transformation is at the heart of MDD. The transformation again should be compatible and based on the transformation rules besides consistency rules. The automated transformation from a design model to another design model can have many benefits. The benefits include improvement of quality as it checks compatibility and use criteria for transformation rules, enhances productivity as it takes less time, reduces cost, improves accuracy and satisfies the stakeholders of the system as they can view the deliverables faster and give feedback on them instead of waiting for long time to have a glimpse of the proposed system.

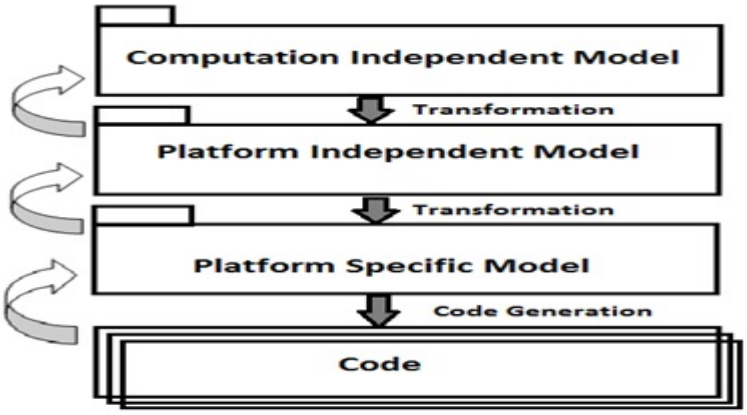

Figure 1 – Shows life cycle of MDA software

As shown in Figure 1 three types of models are offered by MDA. They are Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). The first model is independent of computations. It does not reflect details of system architecture. Rather it shows a product that is not dependent on any computer system. It throws light into requirements of system and its implementation details. It is also known as domain model which can serve as vocabulary for the practitioners at domain level. The PIM is a view of a system in abstract without having any details of platform. This model represents a system that can be implemented in any platform. Thus it is the platform independent representation of a system. It is the link or intermediate view between the system models represented in CIM and PSM. As a matter of fact, the MDA recommends UML as a modelling language to represent PIM. On the other hand, PSM is dependent on the platform. It is the basis for generating code in the process of building a product.

Table 1 – Models mapped to roles in SDLC

Model	Role
CIM	Business Analyst
PIM	Designer
PSM	Developer

As shown in Table 1, each model can be mapped to a role played by software engineers in SDLC. The business analyst is associated with CIM as it focuses on the equipments engineering. The designer role is associated with PIM as it is meant for modelling in platform independent fashion. The developer is associated with PSM as it is to build applications in platform specific approach. Many tools came into existence to support model transformation. In this paper an improvement has been done to the tool named XRTSDC in order to let it support model transformations with consistency based on the transformation rules. Our contributions in this paper are as follows.

Table 2 - Acronyms

Acronym	Description
MOLA	MOdel Transformation LAnguage
MDA	Model Driven Architecture
ERD	Entity Relationship Diagram
UML	Unified mark up language
CIM	Computation Independent Model
PIM	Platform Independent Model
MDD	Model driven Development
QVT	Query/View/Transformation
UDG	use case diagram graph
ADG	activity diagram graph
SDG	sequence diagram graph

SYTG	system graph
TTF	Transformations based Tool Framework
XML	Extendable mark up language
ATL	Atlas Transformation Language
BORM	Business Oriented Relational Modelling
BPMN	Business Process Modelling Notation
PSM	Platform Specific Model
MDWE	Model Driven Web Engineering
CDM	Conceptual Data Model
DFA	Data Flow Approach
ER	Entity Relationship

- A framework for model transformations has been proposed. The main focus is on UML Class Diagram → ERD → and SQL. It does mean to PIM → PIM → PSM. This framework also supports model transformations of UML Class Diagram → Graph.
- We built the functionality in our prototype application to demonstrate the proof of concept. We also evaluated the proposed transformations in terms of consistency and performance.

The remainder of the paper is structured as follows. Section reviews related works. Section III presents the proposed model transformation approach. Section IV presents case study and the model transformations. Section V presents experimental results while section VI concludes the paper besides providing directions for future work.

II. Related works

Audris Kalnis *et al*. (2010) proposed a model transformation language known as MOLA which provide graphical notations used to representation transformation of a system from one model to another model in the context of Model Driven Architecture (MDA). In Audris Kalnis *et al.* (2011) explored the patterns matching concept on MOLA tool as part of model transformation. Abdelouahed Kriouile *et al*. (2015) proposed a Model Driver Architecture based method that can be used to transform models from CIM to PIM. Amani Abdel-Salam Al-Btoush (2015) explored the generation of Entity Relationship Diagram (ERD) from English sentences based on the heuristic rules that help in achieving this. They employed natural language processing to build heuristics. Ariel Gonzalez *et al*. (2015) explored transformation from UML state machines to DEVs models in the context of MDD. They used a language known as declarative QVT relations. Namita Khurana *et al.* (2015) proposed a novel technique that extracts test cases from PIM. They employed graph approach to achieve this. They build different graphs for PIM model such as use case diagram graph (UDG), activity diagram graph (ADG), sequence diagram graph (SDG) and an integrated graph named system graph (SYTG). Once graphs are generated, genetic algorithm is used to generate test cases and optimize them.

Audris Kalnins, Janis Barzdins (2010) presented a model driven software development framework that is based on reusable components. Their framework is based on Transformations based Tool Framework (TTF). Lola Krogh. (2009) made a provision for model transformation from ERD to class diagram.

Photchana Sawprakhon and Yachai Limpiyakorn (2014) explored source models (XML), ATL (Atlas Transformation Language) transformation, XSLT transformation and visualization of sequence diagram.

Kalyanasundaram and Ugale (2015) made a review of model transformation approaches. De Souja and Giorno (2015) proposed a model transformation methodology from use cases to sequence diagrams and thus leveraging and updating of domain model. Tuma and Hanzlik (2015) explored model-to-model transformation from Business Oriented Relational Modelling (BORM) to Business Process Modelling Notation (BPMN). The case they considered is Petri nets and final state machine.

Chavez *et al* (2015) present a model based approach to check inconsistency PIM and its corresponding PSM. Precisely they did it for UML class model and Java implementation. Barbier (2015) explored the MDD with characterization of enterprise level model transformations. The transformations are presented from PIMs to PIMs and PIMs to PSMs. Moreno *et al*. (2015) Model Driven Web Engineering (MDWE) based on MDA for effective transformations. Ferhat Erat (2015) explored many approaches and technologies for model-to-model transformations. Kahani and Cordy (2015) explored the evaluation of different tools used in the literature for model transformation. They found many approaches for M2M transformations. They include declarative approaches, constructive approaches, graph-based approaches, hybrid approaches, visitor based approaches and template based approaches.

Mens *et al.* (2005) explored taxonomy of model transformations in order to apply it to the process of graph transformation. Baudry (2015) focused on the testing of model transformation in order to improve the quality in transformations. They generated test cases in order to achieve the testing of the source and destination models with two black-box strategies. Angadi and Mohan (2015) presented a methodology for designing a parser that helps in transforming from UML to Conceptual Data Model (CDM). Rafe *et al*. (2011) presented an approach for transforming UML class diagrams to relational tables. They employed graph – transformation approach to achieve this. UML activity diagrams are explored in (Muhammad Touseef Ikram and Naveed Anwer Butt, 2015) for generating tests cases automatically. Kamarudin *et al*. (2015) explored the process of converting user requirements to use case and activity diagrams. Prasanna *et al*. (2010) presented criteria for generating test cases from UML class diagrams. They followed an approach known as Data Flow Approach (DFA). Sarkar *et al*. (2015) presented a methodology to build software application from Entity Relationship (ER) model.

III. Proposed Approach

In the previous works, A framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) has been proposed. This framework was meant for checking software model inconsistencies. Moreover it was made flexible to software engineers as it provides personalized configuration which allows preferences in terms of choosing modelling tool, consistency rule language and visualization mechanism. The architecture has provision for consistency checking. The architecture has both execution model and personalized configuration. The personalized configuration is to provide choices to users as preferences and execution model is for actual model checking and finding inconsistencies. Complete description of the tool is not in the scope of this paper. However, the details of the framework can be found in our previous paper (G. Ramesh, Dr. T. V. Rajini Kanth and Dr. A. Ananda Rao, 2016). Here, the model transformation part of the enhanced framework has been described.

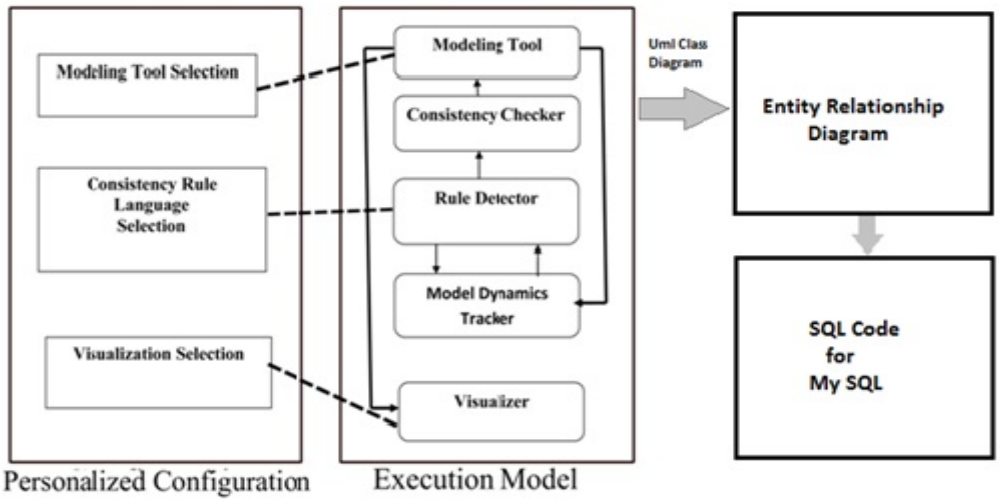

Figure 2 – Extended XRTSDIC framework with model transformation

As shown in Figure 2, the extreme right portion of the framework showing entity relationship diagram and SQL code for My SQL. The aim of this extension is to have model transformations from UML class diagram to ERD and from ERD to SQL code. The model transformation is done with both consistency checking and checking model transformation rules. This is to say that the framework helps in modelling UML diagrams. The class diagram is taken from model and transformed to ERD. Then from ERD corresponding SQL code for My SQL is generated. This is broadly the purpose of the extension to our framework DRTSDIC.

Model Transformation Methodology

The broad methodology is described here. The methodology starts with perosnalized configuration. Then a model is drawn as per the system requirements. While drawing model, design inconsistencies are performed. If there are consistencies then the model is redrawn to rectify issues. Else the model is used for transformation. In this case, the model has been transformed from UML class diagram (PIM) to ERD (PIM). Then the ERD is transformed to SQL targeting the RDBMS MY SQL.

```
1   Start
2   Choose modelling tool
3   Choose consistency rule language
4   Choose visualization technique
5   Draw a model
6   C = Consistency checking
7   IF C == 0 THEN
8      Perform model transformation
9   ELSE
10     Resolve inconsistencies
11  END IF
12  End
```

Algorithm 1 – Overview of methodology

The model transformation of the proposed system is described here.

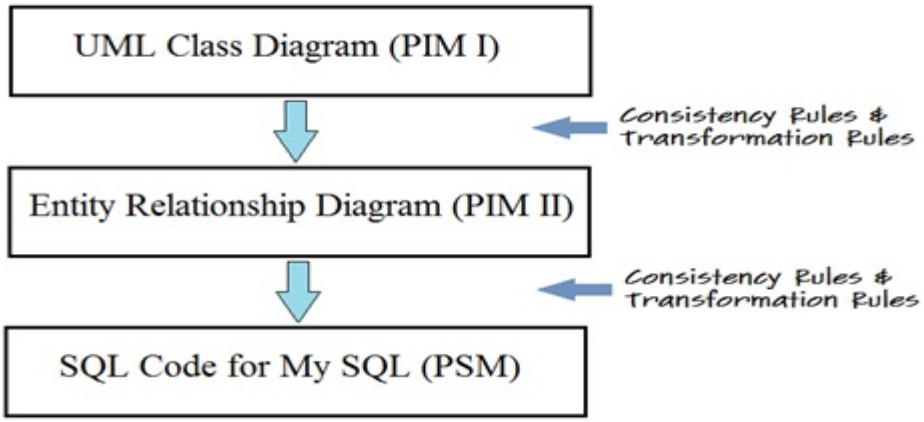

Figure 3 – Model transformation dynamics

As can be seen in Figure 3, it is evident that there is model transformation from one platform independent model to another platform independent model. To state differently, there is transformation between UML class diagram to ERD. In the transformation process consistency rules and transformation rules are applied in order to have accurate transformation. Then the ERD is transformed into a platform specific model that is SQL code for My SQL. Since My SQL uses a dialect of SQL, its code is obviously not same as other RDBMS. Though SQL commands are same as per global standards, there are certain differences in terms of commands and data types. These slight differences are taken care of when model is transformed from PIM to PSM.

Class Diagram General Transformation Rules

Class diagram contains class name, attributes and methods. The class diagram is transformed to ERD according to the rules specified below.

Class Name → Entity Name

Consistency and transformation rule:

if(a new entity is created then)

the entity name should be unique and should be available in class diagram

Class Attribute → Entity Attribute

Consistency and transformation rule:

if(a new attribute is created then)

the attribute name should be unique and should be available in the class attributes

Class Attribute Type → Entity Attribute Type

Consistency and transformation rule:

if(attribute type is determined then)

the attribute type should match or compatible with that of class attribute

Class Method → Entity Method Type

This transformation is not applicable as the entity cannot have operations.

Class Method Arguments → Entity Method Arguments
This transformation is not applicable as the entity cannot have operations.
Class Method Return Type → Entity Method Return Type
This transformation is not applicable as the entity cannot have operations.

Listing 1 – Consistency and transformation rules (Class Diagram → ERD)

These rules are applied and the generated ERD is written to an XML file. The XML file is platform independent, device independent and protocol independent standard for representation of data. Thus XML is being to represent ERD. Then the XML is used to transform from ERD to SQL commands compatible with the RDBMS My SQL.

ER Diagram General Transformation Rules

Entity Name → Table Name
Consistency and transformation rule:
if(a new table is created then)
the table name should be unique and should be available in ER diagram
Entity Attribute → Table Column
Consistency and transformation rule:
if(a new column is created then)
the column name should be unique and should be available in the entity's attributes
Entity Attribute Type → Table Column Type
Consistency and transformation rule:
if(attribute type is determined then)
the table column type should match or compatible with that of entity attribute
Entity Attribute Size → Table Column Size
Consistency and transformation rule:
if(attribute size is determined then)
the table column size should match or compatible or else defaulted

Listing 2 – Consistency and transformation rules (ERD → SQL)

As shown in listing 2, the XML content is used to transform from ERD to SQL that is compatible with My SQL. While performing transformation, the specified consistency rules and transformation rules are applied.

Handling Issues with Class Relationships

Classes in object oriented programming language can have the following relationships.

1. **IS A** Relationship (inheritance relationship)
   a. Generalization
   b. Realization
2. **HAS A** Relationship (containment relationship)
   a. Aggregation
   b. Composition
3. **USES** Relationship (dependency)

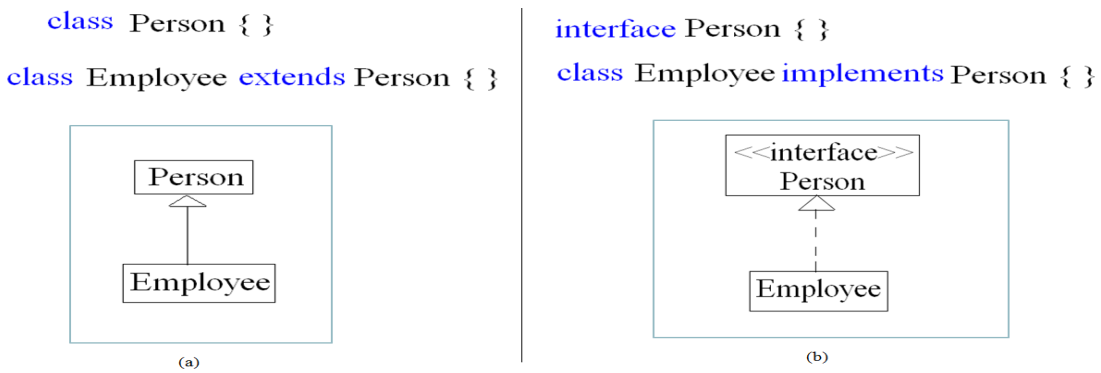

Figure 4 – Shows difference between Generalization (a) and Realization (b)

Generalization occurs when sub class is derived from super class. Realization occurs when the super class is an interface. The inheritance relationship is thus of two types namely generalization and realization.

In the same fashion, the HAS A relationship is of two types namely aggregation and composition. The aggregation source code, memory and UML representation are shown in Figure 5.

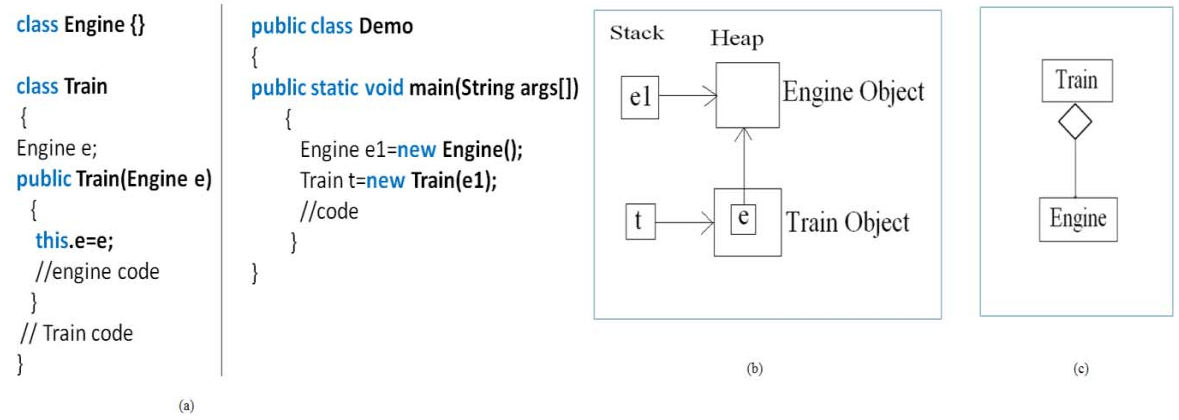

Figure 5 – Aggregation source code (a), memory representation (b) and UML notation (c)

As shown in Figure 5, the source code sample for aggregation, its corresponding memory structure and UML notation are presented.

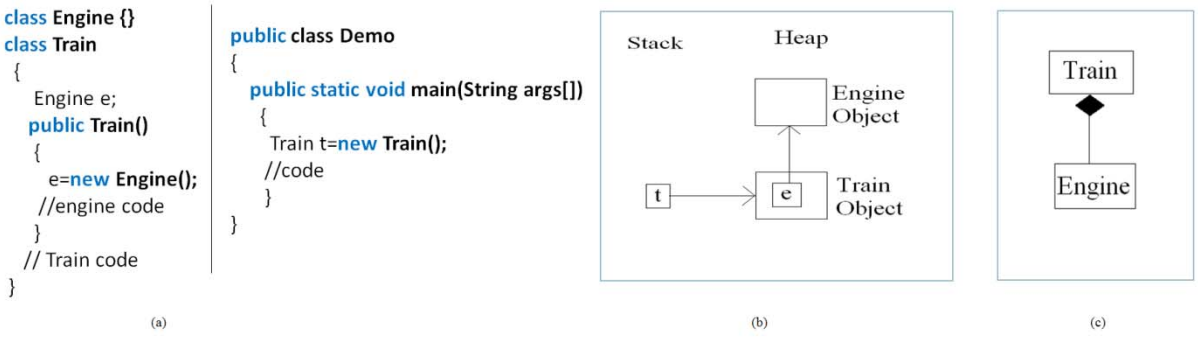

Figure 6 – Composition source code (a), memory representation (b) and UML notation (c)

As shown in Figure 6, the source code sample for composition, its corresponding memory structure and UML notation are presented.

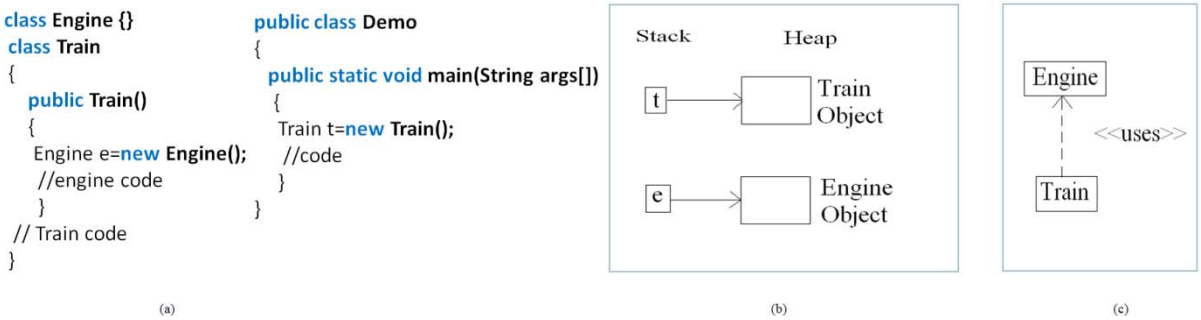

Figure 7 – Dependency source code (a), memory representation (b) and UML notation (c)

As shown in Figure 7, the source code sample for dependency relationship, its corresponding memory structure and UML notation are presented.

Different Types of Mapping from Class Diagram to ERD

There are three different types of mappings known as horizontal mapping, vertical mapping and filtered mapping. The way of mapping determines its name. The mapping dynamics of these categories are presented here.

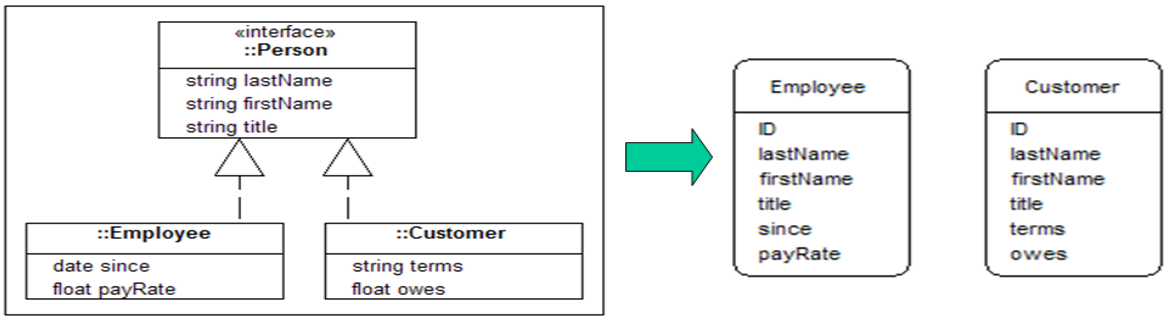

Figure 8 – Shows horizontal mapping

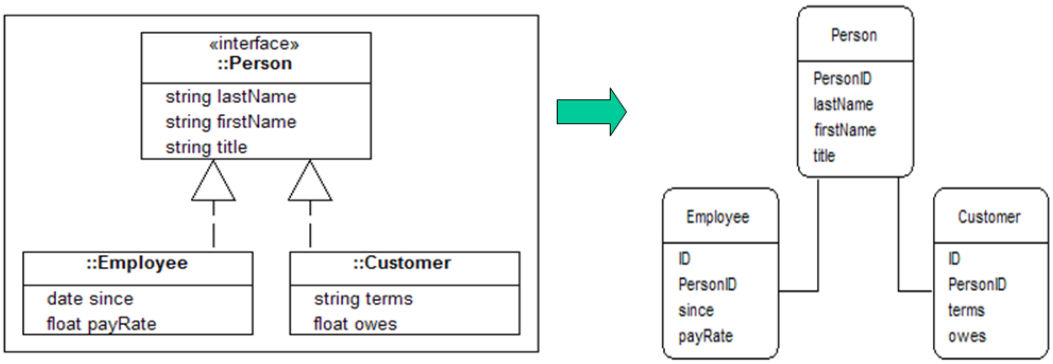

Figure 9 – Shows vertical mapping

As shown in Figure 8, the horizontal mapping eliminates inheritance and polymorphism in the relationships of ERD. A separate entity is used for every non-abstract class. All attributes of classes including inherited ones are mapped to corresponding entities.

As shown in Figure 9, the vertical mapping ensures that every sub class including abstract classes and interfaces has its own table and the inheritance is preserved in the ERD.

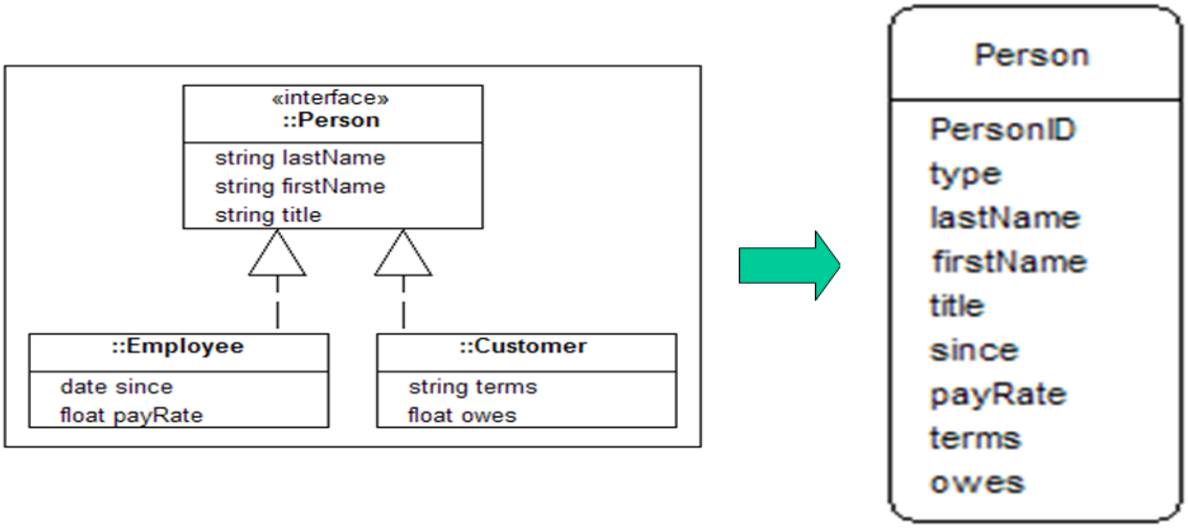

Figure 10 – Shows filtered mapping

As shown in Figure 10, the filtered mapping preserved polymorphism by de-normalizing the relational model. To differentiate the entities it makes use of discriminatory attribute that can differential between entities.

IV. Prototype and Case Study

Our prototype application with the framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) has been improved to equip with model transformation capabilities. It was earlier limited to consistency checking in UML models. The application employs the proposed algorithm for model transformations. Thus tool focuses on the focus is on two different transformations. First, UML Class Diagram

→ ERD → and SQL. It does mean to PIM → PIM → PSM. Second, UML Class Diagram → Graph. The case study considered is a Hospital Management System (HMS) with minimum functionalities to be simple for model transformation demonstrations. The HMS has different objects such as Doctor, Patient, Staff, Ward and so on. The UML class diagram shows different classes such as Patient, Registration, Edit, Expenditure, Appointment, Doctor, Staff, Income, and TestOperator. The attributes and methods of each class can be found in the class diagram shown in Figure 11.

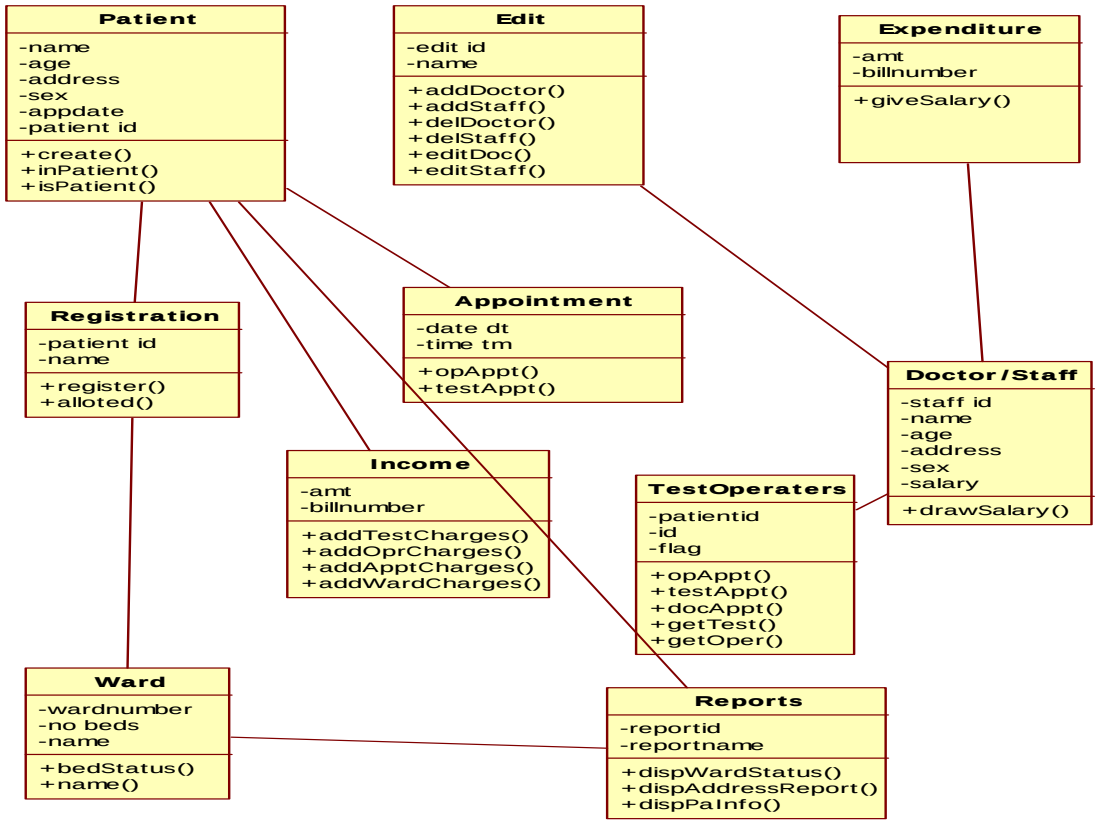

Figure 11 – UML class diagram for HMS (PIM I)

As can be seen in Figure 11, it is evident that the classes are with attributes and actions. The attributes are considered for model transformation from UML class diagram to ERD. ERD is related to backend of the application while the UML class diagram represents classes involved in the implementation in technology platform independent fashion.

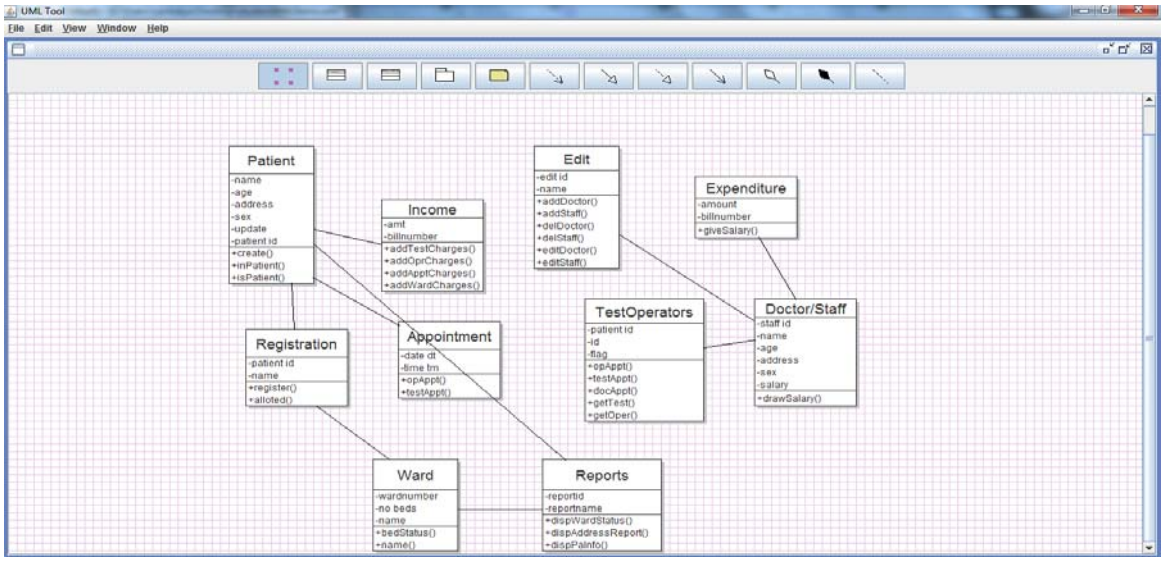

Figure 12 – HMS class diagram drawn using our modelling tool

By employing our algorithm for transformation from UML class diagram to ERD, the following is generated automatically. The transformation is done in XML format and visualized the same here.

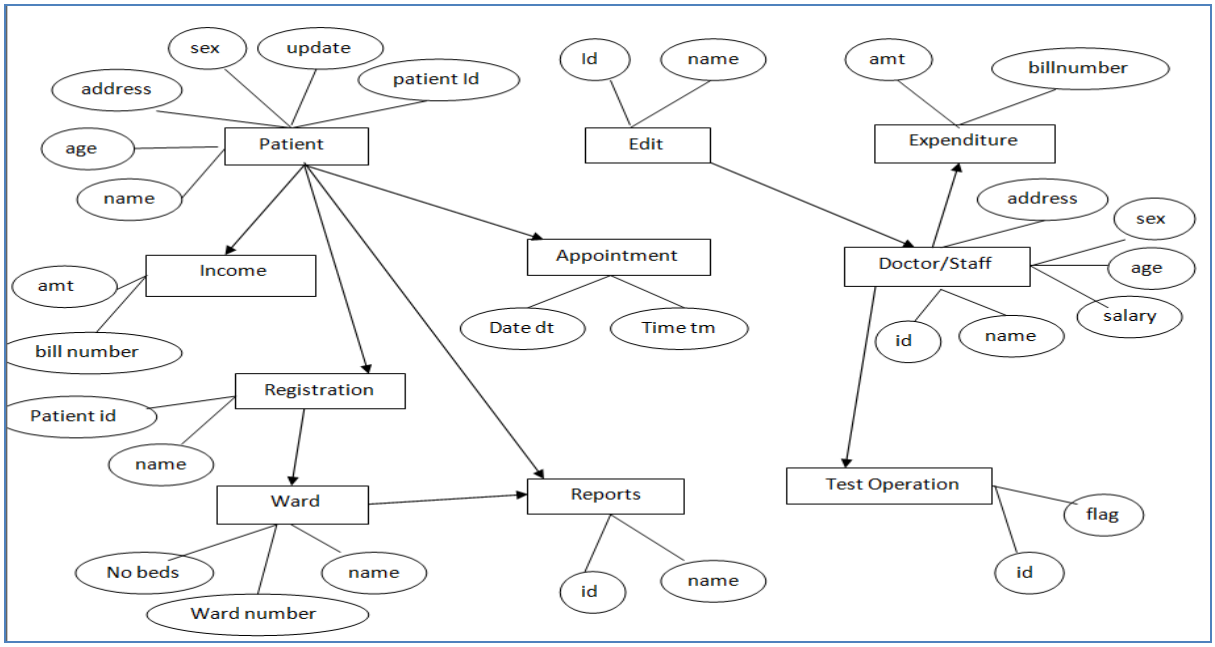

Figure 13 – Transformed ERD (PIM II)

As can be seen in Figure 12 and Figure 13, it is evident that the model transformation is from PIM to PIM. Both source and destination models are PIMs. The transformation is based on certain criteria as shown in Table 3.

Table 3 – Transformation criteria

UML Class Diagram (PIM I)	ERD (PIM II)
Class Name	Entity Name
Attribute of Class	Attribute of Entity
Actions	N/A

As shown in Table 3, the class name is mapped to entity name and attribute of class is mapped to attribute of entity and no transformation is done with respect to actions of that are members of classes. The rationale behind this is that ERD does not reflect actions. However, ERD shows the relationships. The actions can be exploited to identify correct relationships.

```
<Classes>
<class name="patient">
<property name="name" type="String"/>
<property name="age" type="String"/>
<property name="address" type="String"/>
<property name="sex" type="String"/>
<property name="appdate" type="String"/>
<property name="patientid" type="int"/>
</class>

<class name="edit">
<property name="editid" type="int"/>
<property name="name" type="String"/>
</class>

<class name="expenditure">
```

```
<property name="amt" type="String"/>
<property name="billnumber" type="int"/>
</class>

<class name="registration">
<property name="patientid" type="int"/>
<property name="name" type="String"/>
</class>

<class name="appointment">
<property name="datedt" type="date"/>
<property name="timetm" type="time"/>
</class>

<class name="income">
<property name="amt" type="int"/>
<property name="billnumber" type="int"/>
</class>

<class name="ward">
<property name="wardnumber" type="int"/>
<property name="nobeds" type="int"/>
<property name="name" type="String"/>
</class>

<class name="testoperaters">
<property name="patientid" type="int"/>
<property name="id" type="int"/>
<property name="flag" type="String"/>
</class>
<class name="Doctor/Staff">
<property name="staffid" type="int"/>
<property name="age" type="String"/>
<property name="address" type="String"/>
<property name="name" type="String"/>
<property name="sex" type="String"/>
<property name="salary" type="int"/>
</class>

<class name="Reports">
<property name="reportid" type="id"/>
<property name="reportname" type="String"/>
</class>

</Classes>
```

Listing 3 – Generated XML file representing ERD

As shown in Listing 3, the from class diagram (PIM I) ERD (PIM II) is generated in the form of an XML file. The XML content contains the entities, their attributes with data and metadata required to get transformed into PSM. After generating PIM II, the framework focused on ERD → SQL. It is from PIM to PSM. The SQL generated is for MY SQL which is an open source Relational Database Management System (RDBMS). The generated SQL queries are as shown in Figure 3.

```
CREATE DATABASE `diagrams`
USE `diagrams`;
/*Table structure for table `appointment` */
DROP TABLE IF EXISTS `appointment`;
CREATE TABLE `appointment` (
`date` varchar(10) default NULL,
`time` varchar(20) default NULL
)
/*Table structure for table `edit` */
DROP TABLE IF EXISTS `edit`;
CREATE TABLE `edit` (
`edit id` varchar(10) NOT NULL,
`name` varchar(20) default NULL,
PRIMARY KEY  (`edit id`)
)
/*Table structure for table `expendature` */
DROP TABLE IF EXISTS `expendature`;
CREATE TABLE `expendature` (
`amt` varchar(10) default NULL,
`billnumber` varchar(20) default NULL
)
/*Table structure for table `income` */
DROP TABLE IF EXISTS `income`;
CREATE TABLE `income` (
`amt` varchar(10) NOT NULL,
`billnumber` varchar(20) NOT NULL,
PRIMARY KEY  (`amt`,`billnumber`)
)
/*Table structure for table `patient` */
DROP TABLE IF EXISTS `patient`;
CREATE TABLE `patient` (
`patientid` varchar(10) default NULL,
`name` varchar(20) default NULL,
`age` varchar(30) default NULL,
`address` varchar(40) default NULL,
`sex` varchar(50) default NULL,
`appdate` varchar(60) default NULL
)
/*Table structure for table `registration` */
DROP TABLE IF EXISTS `registration`;
CREATE TABLE `registration` (
`patient id` varchar(10) NOT NULL,
`name` varchar(20) NOT NULL,
PRIMARY KEY  (`patient id`,`name`)
)
/*Table structure for table `reports` */
```

```
DROP TABLE IF EXISTS `reports`;
CREATE TABLE `reports` (
`report id` varchar(10) NOT NULL,
`report name` varchar(20) NOT NULL,
PRIMARY KEY  (`report id`,`report name`)
)
/*Table structure for table `staff` */
DROP TABLE IF EXISTS `staff`;
CREATE TABLE `staff` (
`staff id` varchar(10) default NULL,
`name` varchar(20) NOT NULL,
`age` varchar(30) default NULL,
`address` varchar(40) NOT NULL,
`sex` varchar(50) NOT NULL,
`salary` varchar(60) default NULL,
PRIMARY KEY  (`name`,`address`,`sex`)
)
/*Table structure for table `testoperators` */
DROP TABLE IF EXISTS `testoperators`;
CREATE TABLE `testoperators` (
`patient id` varchar(10) NOT NULL,
`flag` varchar(20) default NULL,
PRIMARY KEY  (`patient id`)
)
/*Table structure for table `ward` */
DROP TABLE IF EXISTS `ward`;
CREATE TABLE `ward` (
`ward number` varchar(10) default NULL,
`number of beds` varchar(20) default NULL,
`name` varchar(30) NOT NULL,
PRIMARY KEY  (`name`)
)
```

Listing 4 – SQL code generated for My SQL

As seen in Listing 4, the code is generated to target My SQL database. Though SQL is command for all RDBMS, there are different dialects that are proved to be slightly different.

V. Experimental Results

The proposed tool has been used to have experiments on model transformations. Many models have been used to study the functionality of the tool. This paper throws light on one model known as Hospital Management System (HMS). First of all a model is being designed using UML notations after user selecting preferences on modelling tool, consistency rules and visualization technique. Once the model is proved to be consistent, the model has been transformed into another PIM known as ERD. The ERD is generated in the form of XML. Then the ERD is transformed into PSM known as SQL which is targeted for an RDBMS named My SQL.

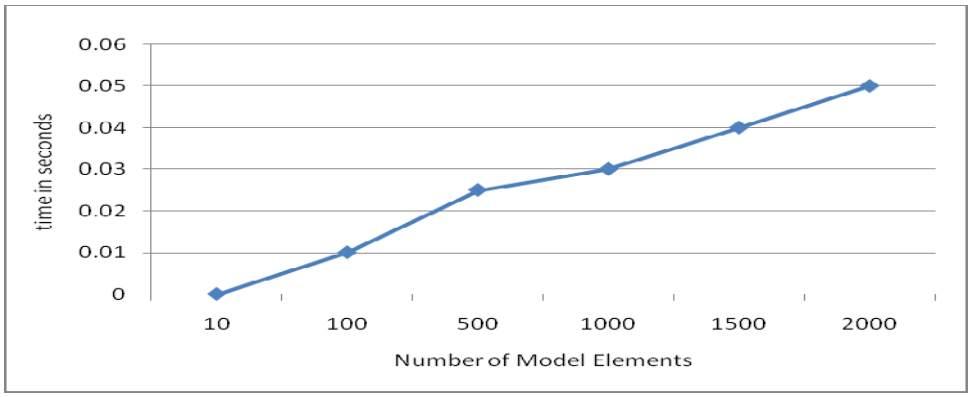

Figure 14 – Model transformation performance (PIM to PIM)

As shown in Figure 14, the horizontal axis represents number of elements while the vertical axis represents time in seconds. There is clear trend in results which says that the time taken is directly proportional to the number of model elements in the source PIM.

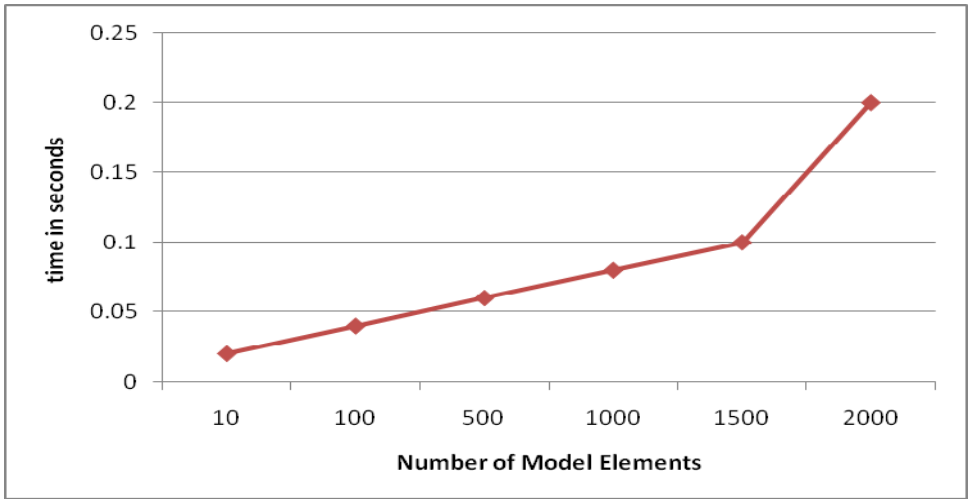

Figure 15 – Model transformation performance (PIM to PSM)

As shown in Figure 15, the horizontal axis represents number of elements while the vertical axis represents time in seconds. There is clear trend in results which says that the time taken is directly proportional to the number of model elements in the source PIM. Interestingly PIM to SIM took more time when compared with that of PIM to PIM transformation.

VI. Conclusions and Future Work

Model Driven Development (MDD) can exploit transformation models to help expedite software development. Many modelling tools like Rational Rose came into existence that supports Computer Aided Software Engineering (CASE) with provision for forward engineering to generate skeletal programs automatically. Such tools increase developer productivity and consistency. Design models should be consistent and the ability to transform models is desirable for rapid application development. In this paper our research is an extension to our previous work. In our previous work, A tool XRTSDC for automatic detection of design inconsistencies had been built. A modular extension has been made to this tool to support model transformation from PIM→PIM and PIM→PSM. An algorithm is being proposed and defined model transformation rules and design consistency rules so as to help the tool to transform models quickly and consistently. This prototype application is used to demonstrate the proof of concept. A model has been designed using UML then transformed its class diagram (PIM) to ERD (PIM). Then from ERD (represented in XML), the tool transforms into PSM known as SQL compatible to MySQL. Our empirical results with a case study model revealed that the proposed approach achieve model transformation with significant accuracy and performance. In future our framework will be extended further to have model transformation from PIM (class diagram) to its implementation (PSM) in Java and C++.

References

[1] Abdelouahed Kriouile, Najiba Addamssiri, Taoufiq Gadi, 2015. An MDA Method for Automatic Transformation of Models from CIM to PIM.American Journal of Software Engineering and Applications, p271-280.

[2] Amani Abdel-Salam Al-Btoush, 2015. Extracting Entity Relationship Diagram (ERD) from English Sentences. International Journal of Database Theory and Application. 8 (2), p1-9.

[3] Ariel Gonzalez,Carlos Luna,Roque Cuello, Marcela Perez, Marcela Daniele, 2015. Towards an automatic model transformation mechanism from UML state machines to DEVS models. CLEI ELECTRONIC JOURNAL VOLUME. 18 (2), p10-19.

[4] Audris Kalnins, Janis Barzdins,. 2010. MDA Support by Transformation Based Tool. SCM, p.1370-1381.

[5] Audris Kalnins, Janis Barzdins, Edgars Celms. 2010. Model Transformation Language MOLA. CSO, p.20-30.

[6] Audris Kalnins, Edgars Celms, Agris Sostaks. 2011. Simple and Efficient Implementation of Pattern Matching in MOLA Tool. CSO, p.20-30.

[7] Benoit Baudry. 2015. Testing Model Transformations: A case for Test Generation from Input Domain Models. Model, p.20-30.

[8] Fabio Cardoso de Souza, Fernando Antonio de Castro. 2015. Automatic Generation of Sequence Diagrams and Updating Domain Model from Use Cases. SOFTENG 2015 : The First International Conference on Advances and Trends in Software Engineering, p271-280.

[9] Ferhat Erat. 2015. D3.1.1 Review of Model-to-Model Transformation Approaches and Technologies. Text & Model-Synchronized Document Engineering Platform, p.70-85.

[10] G. Ramesh, Dr. T. V. Rajini Kanth and Dr. A. Ananda Rao, January 2016. XRTSDIC: Towards a Flexible and Scalable Framework for Detecting and Tracking Software Design Inconsistencies", the first A. P. Science Congress (APSC'2016).

[11] Hector M. Chavez, Wuwei Shen, Robert France, Benjamin A. Mechling, Guangyuan Li. 2015. An Approach to Checking Consistency Between UML Class Model and Its Java Implementation. IEEE, p1-10.

[12] Ismalic. 2011. From Class diragram to relations . Electronic Commerce Research and Applications, p1-9.

[13] Jakub Tuma and Petr Hanzl. 2015. Automated Model Transformation Method from BORM to BPMN. Jakub Tuma and Petr Hanzlk. 9 (116), p.429-442.

[14] Lola Krogh. (2009). Generate Class Diagram from Entity Relationship Diagram (ERD). Visual Paradigm Generate Class Diagram from Entity Relationship Diagram (ERD), p.811-820.

[15] M. Franck BARBIER. 2015. Enterprise Model-Driven Development with BLU AGE™ A Netfective Technology White Paper. J2EE, p1370-1381.

[16] M. Prasanna,K.R. Chandran,Devi Bhakta Suberi. 2010. Automatic Test Case Generation for UMLClass Diagram using Data Flow Approach.CSO, p1-10.

[17] Muhammad Touseef Ikram,Naveed Anwer Butt. 2015. Testing from UML Design using Activity Diagram: A Comparison of Techniques.International Journal of Computer Applications. 131 (5), p1370-1381.

[18] Namita Khurana, Rajender Singh Chhillar, Usha Chhillar. (2016). A Novel Technique for Generation and Optimization of Test Cases Using Use Case, Sequence, Activity Diagram and Genetic Algorithm. Journal of Software. 11 (3), p271-280.

[19] Nathalie Moreno, José Raúl Romero, and Antonio Vallecillo1. 2015. An Overview of Model-Driven Web Engineering and the MDA. Software, p271-280.

[20] Naseh Kahani and James R. Cordy. 2015. Comparison and Evaluation of Model Transformation Tools. Technical Report, p.20-30.

[21] Nuri Jazuli Kamarudin, 2Nor Fazlida Mohd Sani, 3Rodziah Atan. 2015. Automated Transformation Approach from User Requirement to Behavior Design. Journal of Theoretical and Applied Information Technology. 81 (1), p.20-30.

[22] Pallavi Kalyanasundaram,Sunita P. Ugale. 2015. Model Transformation: Concept, Current Trends and Challenges. International Journal of Computer Applications (0975 – 8887). 119 (14), p.90-101.

[23] Photchana Sawprakhon, Yachai Limpiyakorn. 2014. Sequence Diagram Generation with Model Transformation Technology.Proceedings of the International MultiConference of Engineers and Computer Scientists. 1 (14), p.20-30.

[24] Souradeep Sarkar1, Mr. Debasish Hati2, Mr. Prasun Kumar Mitra3. 2015. Software Application Generator: An ER Model-based Software Product Building Too. International Conference on Information Engineering, Management and Security, p1370-1381.

[25] Suneeta H. Angadi, S. Mohan. 2015. Design Parser For CDM Graph Geneartion in Graph Transformation Based User Interface Modeling.Research Journal of Applied Sciences, Engineering and Technology. 9 (5), p1-10.

[26] Tom Mens,Pieter Van Gorp, 2005. Applying a Model Transformation Taxonomy to Graph Transformation Technology. CSO, p1370-1381.