EER TO UML MODEL TRANSFORMATION USING ATLAS TRANSFORMATION LANGUAGE (ATL)

Shivaji Arjun Kakad[1], Prof. Y. N. Patil[2]
[1,2]Department of Computer Engineering
Dr. BAT University, Lonere, Raigad (MH, INDIA)

Abstract: The purpose of this report is to transform Enhanced (Extended) Entity Relationship (EER) to Unified Modeling Language (UML) that is EER to UML. EER contains all concepts of ER. EER adds more concepts like Specialization/Generalization, Subclass/Super Class, Categories and Inheritance to get more accuracy than ER. UML includes a set of graphic notation techniques to create visual models of Object-oriented software intensive systems. Unified Modeling Language is a standard for software development. EER Diagram on the other hand is used for database design. The class diagram included in the UML alone is contender for replacing entity relationship models. UML was not explicitly created to support database design. This provides an easy way to enter the specifications of a database and to make UML class diagram from it.
Keywords: ATL, IDE, EER, EMF, MDE, Model-to-Model Transformation, UML.

I. INTRODUCTION

This chapter gives overview about existing method and techniques for model transformation and transformations present now a days.

A. Literature Survey

The following figure shows the database modeling and implementation. Here the ideas in our mind can be modeled using the techniques like Entity Relationship (ER), Enhanced Entity Relationship (EER), Unified Modeling Language (UML), etc. Then it is converted to the relational schema, and then we can do the Relational Database Management System (RDBMS) implementation shown below

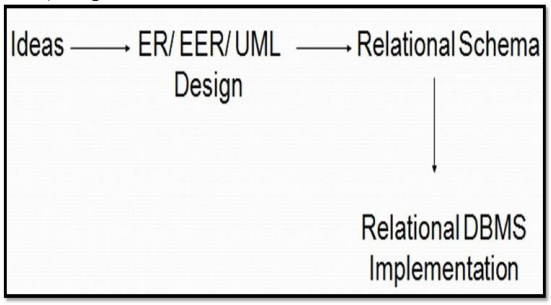


Figure 1: Database Modeling and Implementation Process

B. Existing Transformations

There are various transformations are present at the Eclipse Foundation, ATL examples at the website given here is http://www.eclipse.org/atl/documentation/basicExamples-Patterns/. The list of some of them is given below

1. UML to JAVA
2. Class to Relational
3. Tree to List
4. Book to Publication
5. Families to Persons
6. KM3 to Metrics
7. KM3 to Measure
8. Excel to S/W Quality Control
9. Ant to Maven
10. BibTeXML to DocBook
11. Make to Ant
12. OpenBlueLab to UML
13. Table to Microsoft Office Excel
14. Java source to Table
15. MOF to UML
16. Measure to XHTML
17. Measure to table
18. KM3 to EMF
19. KM3 to ATL copier
20. KM3 to XML
21. MySQL to KM3
22. ATL to BindingDebugger
23. Maven to Ant
24. Monitor to Semaphore
25. Simple Class to Simple RDBMS
26. Public to Private, etc.

C. Available Transformation Languages

There are some transformation languages are listed below
ATL- a transformation language developed by the INRIA.

Tom- a language based on rewriting calculus, with pattern-matching and strategies.

JTL- a bidirectional model transformation language specifically designed to support
non-bijective transformations and change propagation.

M2M- is the Eclipse implementation of the OMG QVT standard.

QVT- the OMG has defined a standard for expressing M2M transformations, called MOF/QVT or in short QVT.

VIATRA- a framework for transformation based verification and validation environment.

II. Model Driven Engineering

Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that is, abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts.

A. ATL

ATL is Atlas Transformation Language. ATL is a model to model transformation language developed at INRIA. It is specified both as a metamodel and as a textual concrete syntax. It is a hybrid style combination of declarative style and imperative style. In declarative style, simple mappings can be expressed simply. Imperative style constructs are provided to express complex mappings.

B. STRUCTURE OF ATL

In MDE everything is called a model. A model refers to an abstract view of a real world system of interest, i.e., a simplification of a system. A model conforms to a metamodel while a metamodel conforms to a metametamodel (MMM). Model Driven Development (MDD) is copyrighted term by Object Management Group (OMG). One of the most important operations in MDE/MDD is model to model transformation. There are different kinds of model transformations including model to model and model to code, model to text. We are using model to model transformations in this paper.

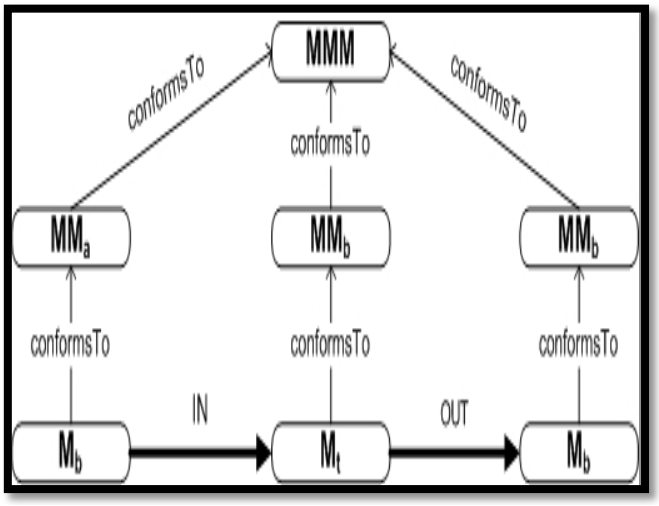


Figure 2 Structure of process for ATL

Above figure shows the structure of process of model transformation. Every artifact in MDE is a model; the model to model transformation is also a model that conforms to a metamodel. The transformation model defines how to generate models that conform to a particular metamodel from models that conform to another metamodel or the same metamodel. In above figure, the transformation model Mt transforms Ma to Mb. Mt conforms to MMt while Ma and Mb conform to MMa and MMb. The three metamodels conform to a common metametamodel MMM. This is shown in above figure.

III. ATL DEVELOPMENT

The ATL module corresponds to a model to model transformation. Here we are taking example of Circle to Square transformation. The file extension for this is .atl. It consists of Header, Import, Helpers, and Rules thee are explained below

A. HEADER

The header section is the first section, which defines the names of the transformation module and the variables of the source and target metamodels. The following ATL example code represents the header of the Circle2Square.atl file, thus the ATL header for the transformation from Circle2Square:

```
module Circle2Square;
create OUT :Square from IN :Circle;
```

The keyword module defines the module name. The keyword create introduces the target models declaration, while from introduces the source models declaration.
This is compulsory section.

B. IMPORT

The import section is the second section, which declares what libraries have to be imported. For example, to import the strings library, one would write:

```
uses strings;
```

This is optional section. The keyword uses declares the libraries that have to be imported. We can import several libraries.

C. HELPERS

This is third section; Helpers can be used to define (global) functions and variables. Helper functions can call each other (recursion is possible) or they will be called from within rules. In general, they serve to define repetitive pieces of code in one place. The following text shows the example of helper

```
helper context MM!Color
def: toString() : String = self.value;
```

Also, this is optional section.

D. RULES

Rules describe the transformation from a source model to a target model by relating metamodels. Each rule contains a unique name. It is introduced by the keyword rule that is followed by the rule's name. Its implementation is surrounded by curly brackets. The example of relations transformation shown in below rule

```
Rule Relations
{
from RC:MM!Relation
to RS:MM1!Relation
(
Source <- RC.Source,
Target <- RC.Target
)
}
```

Also, this is optional section.

IV. EXECUTION OF ATL

The process of the ATL execution environment is shown in figure 4.1 given below. That contains the following parts across several layers,

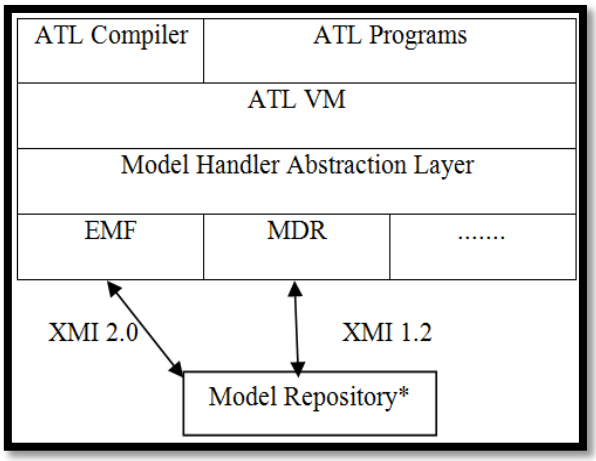
• ATL Compiler. This transforms ATL programs into programs in byte-code,

• ATL Virtual Machine. It executes the byte-code generated by the compiler. It will help for provides set of instructions for model manipulation and handling models.

• Model Handler Abstraction Layer. The VM may run on top of different model management systems. To separate the machine from their specifics an intermediate level is introduced called Model Handler Abstraction Layer. This layer transforms the instructions of the Virtual Machine (VM) for model manipulation to the instructions of a specific model handler.

• Model Handlers. These components that provide programming interface for model manipulation. Some of the examples are Eclipse Modeling Framework (EMF) and Metadata Repository (MDR).

• Model Repository. This provides storage facilities for models. As given in figure below, shows the simplest form of a model repository is the file system that stores models as XML files serialized according to the XML Metadata Interchange (XMI) standard. Because of this type of layered architecture we can achieve the requirements for flexibility of the execution engine. Additions of new language features affect mainly the ATL compiler. A more efficient execution requires changes in the compiler (some static optimizations may be performed by the compiler) and the implementation of the VM.



*The simplest form of a model repository is a file system

Figure 4. Process of Execution

Now a day's present program will run on top of a new Virtual Machine provided that it conforms to the same set of instructions. A specification of Atlas Transformation Language Virtual Machine is provided on the GMT website.

The following figure shows the requirements of the transaction while execution. Here, the requirements for the transformations are input model (M), input metamodel (MM), output metamodel (MM). and last one and important is ATL file that contains ATL rules for transformation of model.

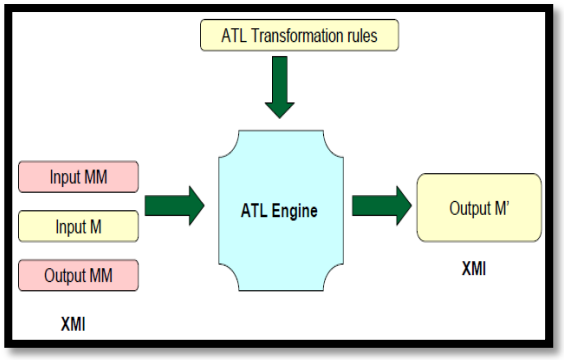


Figure 5. Requirements of transformation

V. Mapping EER to UML

We are giving the EER model as input, then doing preprocessing on EER model, after preprocessing we can use ATL transformation rules, for the transformation of EER to UML. ATL transformation rules are described in next chapter. The transformation architecture is given below. shows the scenario of above. At the time of transformation we are using metamodels of both EER and UML. We get the output model i.e. UML model in .xmi file format.

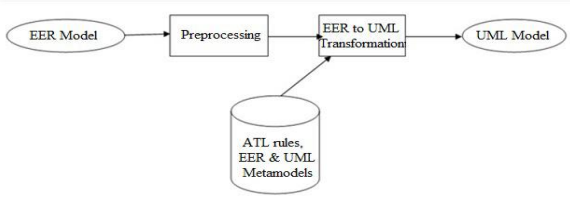


Figure 6. EER to UML transformation architecture

In below figure we can see various similar terminologies between EER and UML. We are going to use these terminologies for various transformations between EER to UML. Here the EER model is transformed in UML model by using following conversion of respective elements. Like entity type from EER is transformed into class in UML, entity from EER is transformed into object in UML, attribute from EER is transformed into attribute in UML, relation type from EER is transformed into association in UML, composite attribute from EER is transformed into structured domain in UML, derived attribute from EER is transformed into operation in UML, relationship instance from EER is transformed into class in UML, cardinality from EER is transformed into multiplicity in UML, etc. The following figure shows the EER entity to its equivalent UML class.

EER vs UML

EER Diagram	•	UML Class Diagram
Entity Type	•	Class
Entity	•	Object
Attribute	•	Attribute
Domain	•	Domain
Composite Attribute	•	Structured Domain
~ [Derived Attribute]	•	Operation
Relationship Type	•	Association
Relationship Instance	•	Link
Cardinality & Participation	•	Multiplicities

Figure 7. EER vs. UML Terminology

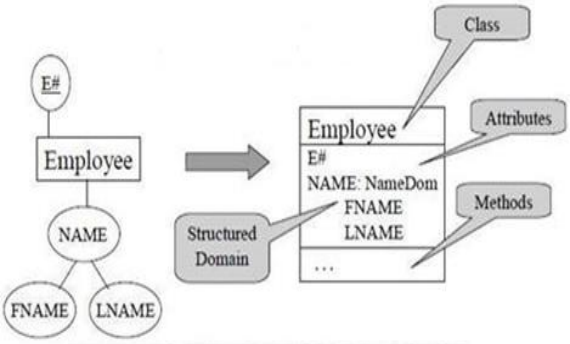


Figure 8. Entity to its equivalent Class

Implementation:

For the above modeling transformation we need to install the modeling version of Eclipse. Once you are ready with the eclipse then you need to install some plug-in i.e. ATL SDK, GMF(Graphical Modeling Framework) Tooling SDK, etc. Then you need to create a new ATL project named as EER2UML in which you will need to develop source Metamodel and target metamodel. Along with these two metamodels we also need to write transformation rules for converting EER elements into corresponding UML elements. The eclipse screen shots are given in figure. As shown in figure, in ATL configuration we need to select EER as well as UML metamodels from existing workspace in respective sections. Also the XMI file of the source model or as input is needed to select in field Source model. And at target model we have to write the name of the output XMI file. Thus after this we have to click on Apply button and then RUN button. Thus after successful execution we get the UML XMI file as output in package explorer section of Eclipse.

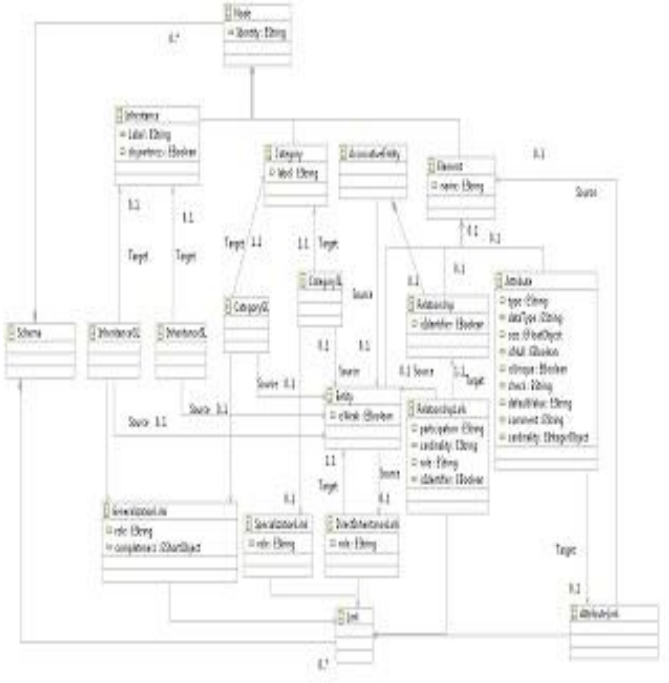


Figure 9. EER Metamodel

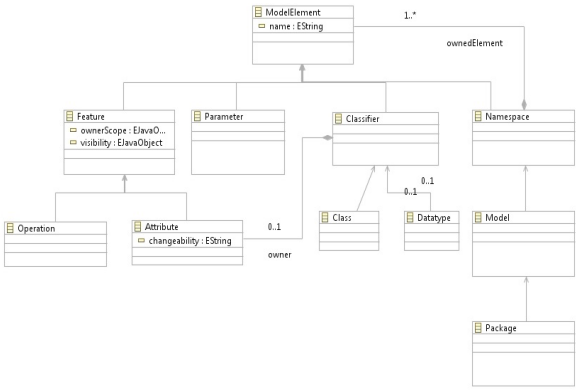


Figure 10. UML Metamodel

UML is not simply a replacement for Entity Relationship Diagramming. It is a complete, integrated object-modeling environment with n parts. The class diagram included in the UML alone is contender for replacing entity relationship models. We no longer operate in a development environment where there will be many competing modeling standards.

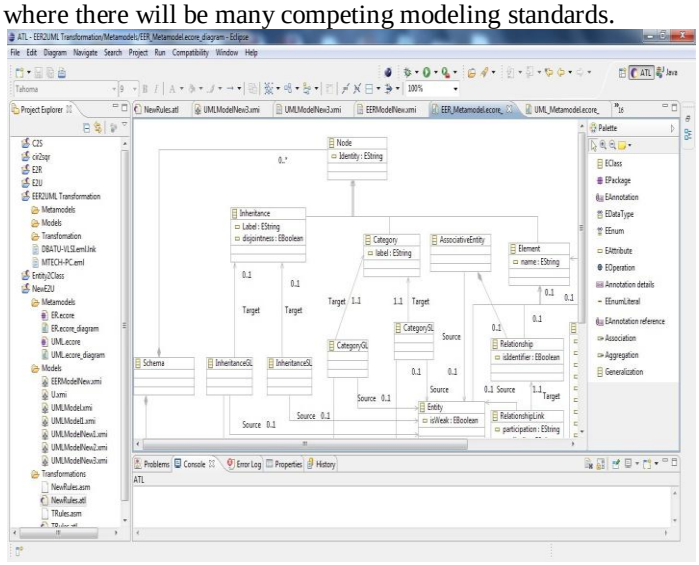


Figure 11. Eclipse metamodel editor (EER)

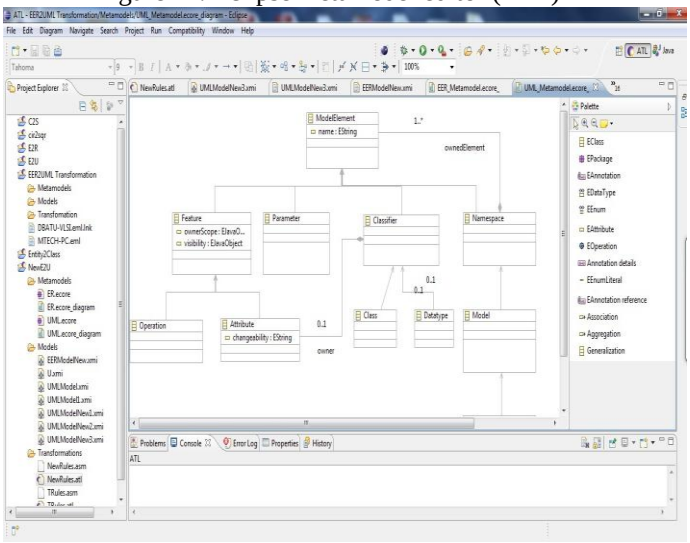


Figure 12. Eclipse metamodel editor (UML)

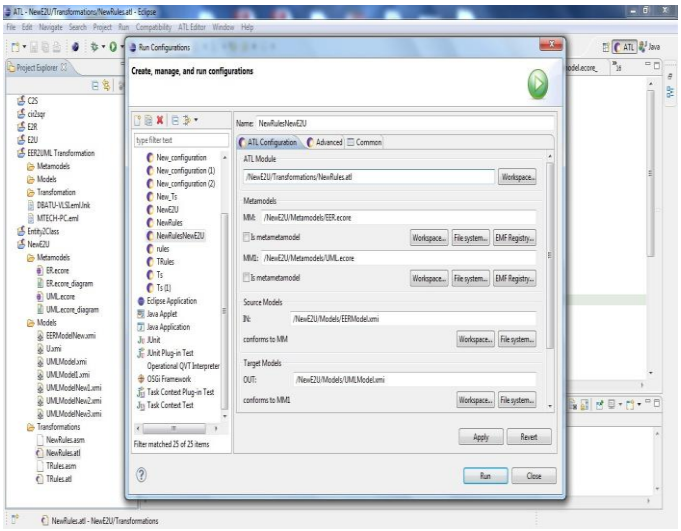


Figure 13. ATL run configuration

Despite its advantages over ER diagramming, UML is not without its weaknesses. In some ways, UML diagrams have more symbols making them more cluttered and therefore less easily understood by users. This may be due to the fact that UML was not explicitly created to support database design. However, class diagrams in UML are a superset of entity relationship modeling. There is nothing that can be expressed with ERDs that cannot be expressed in UML notation; and there are many more structures and relationships possible to express in UML that are not possible with ERD notation. A major strength of UML is that it is explicitly extendable. This openness of UML architecture raises the specter of deviation from the standard. However, UML is rich enough that such deviation should be minimal and easily controlled. UML can and should be used now for both logical and physical relational database modeling. The only reason not to make the shift now is that we do not currently have the products to support this shift.

VI. CONCLUSIONS

In this paper first we presented ATL (Atlas Transformation Language), which is a model transformation language created as a part of the ATLAS Model Management Architecture. ATL is built on top of the Eclipse environment supported by a set of development tools. In this project we have done EER to UML model transformation in eclipse. In which we have defined the metamodels and rules for the EER to UML transformation. Unified Modeling Language is a standard for software development. Extended Entity Relationship Diagram (EERD) on the other hand is used for database design. The class diagram included in the UML alone is contender for replacing entity relationship models. UML was not explicitly created to support database design. This provides an easy way to enter the specifications of a database and to make UML class diagram from it. This accepts input as EER Diagram. The diagram can save the given information. This information will save this information in XMI file. By analyzing the saved information, the mapping from EERD to UML will be done and then the output viz. the corresponding UML class diagram will be generated.

VII. ACKNOWLEDGEMENT

I would like to thank Prof. Y. N. Patil for his guidance for database system, by using which I have done this. We would like to thanks for our parents for everything. We also would like to thank Dr. A. W. Kiwelekar head of Department of Computer Engineering and other staff.

REFERENCES

[1] ATLAS Transformation Language, http://en.wikipedia.org/wiki/ATLAS-Transformation-Language
[2] Eclipse Foundation, Generative Model Transformer Project, website address http://www.eclipse.org/gmt/.
[3] ATLAS group, INRIA & LINA, KM3: Kernel MetaModel Manual. 2004.
[4] ATLAS group LINA & INRIA Nantes, "ATL User Manual version 0.7," Online resource available at: http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf.
[5] Bézivin, J., Jouault, F., and Touzet, D. An Introduction to the ATLAS Model Management Architecture. Research Report LINA, (05-01).
[6] Netbeans Meta Data Repository (MDR). http://mdr.netbeans.org.
[7] ATLAS group, Installation of ADT from source 2004 http://www.sciences.univnantes.fr/lina/atl/www/papers/ATL/ATL_Documentation/ADTInstallation.pdf.
[8] Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45 (2006) 621–645.
[9] Eclipse Foundation, ATL examples http://www.eclipse.org/atl/documentation /basicExamples -Patterns/.
[10] Agrawal A., Karsai G., Kalmar Z., Neema S., Shi F., Vizhanyo A.The Design of a Simple Language for Graph Transformations, Journal in Software and System Modeling, in review, 2005.
[11] Joseph Fong, Mapping of ER to object modeling techniques.
[12] Frdric Jouault, Ivan Kurtev, Transforming Models with ATL.

