Asian Journal of Scientific Research

ISSN 1992-1454

science alert
http://www.scialert.net

ANSInet
an open access publisher
http://ansinet.com

Asian Journal of Scientific Research 8 (1): 54-66, 2015
ISSN 1992-1454 / DOI: 10.3923/ajsr.2015.54.66

Framework to Compare the Model Generation Methods

[1,2]Rim Bouhaouel, [1,4]Naoufel Kraïem, [3]Camile Salinesi, [4]Zuhoor Al-khanjari and [1]Sami Ouali

[1]RIADI Laboratory, ENSI, Campus of Manouba, Tunisia
[2]Esprit University, Tunisia
[3]CRI, Panthéon Sorbonne University, Paris, France
[4]Department of Computer Science, University Sultan Qaboos, Muscat, Oman

Corresponding Author: Rim Bouhaouel, Department of Software Engineering, Esprit University, Z.I. Chotrana II, 72083, Pôle, Technologique-El Ghazala, Ariana, Tunisia

ABSTRACT

The Model Driven Engineering (MDE) needs to abstract with different levels the specification or the solution needed to construct a system and represent it in a model. Our study aims to generate fragments of UML diagrams by using the software product line. To outline our contributions, we need to acquire a thorough knowledge of similar methods using models generation. Therefore, in this study we propose a framework to compare model generation methods using the four worlds framework namely: Nature, system, usage and development. We use it to compare four methods of model generation: FUML, UMLAUT, RSM, FUJABA and to represent the contributions of our study.

Key words: Model, MDE, UML, comparison framework

INTRODUCTION

One of the problems in software engineering is the complexity of the process of software development. Hence, the apparition of the MDE could clarify and simplify concepts and harness them to build software systems. The MDE principles are based on two concepts: system and model and two basic relations: Conformance and representation (Bezivin, 2006). It aims to shift the focus of software development from coding to modeling. The model construction is further reduced by using well-formed rules and constraints of heterogeneous sources. To ensure more comprehension to the model, it's recommended to use a standard representation. The most widespread models are presented by the Unified Modeling Language (UML) which is a language for specifying, visualizing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. "UML allows a model to be constructed, viewed, developed and manipulated in a standard way at analysis and design time" (Soley *et al*., 2000). In our study, we aim to provide the reuse of UML diagrams by generating UML fragments using the software product line strategy. "It aims to reduce development time, effort, cost and complexity by taking advantage of the commonality within a portfolio of similar products" (Voelter and Groher, 2007). Indeed, the reuse of the software construction is a priority target for the software community and modeling is considered as a part of this process, so we need to ensure the reuse of models. In this context, many methods to generate models appear in the MDE. Regardless of their purpose, each one aims to reuse several types of models by using different approaches to satisfy variant goals.

The objective of our study is to define a comparison framework for generation diagram methods based on the four world comparison defined by Rolland (1997) to detect variabilities and commonalities and define our contributions.

MODEL DRIVEN ENGINEERING

The main purpose is to accelerate and facilitate the construction of complex software using objects named models (as defined in the next section). The MDE uses models to represent the elements of software system and expresses solutions and phases in the process of software construction (such as requirements, designs, data structures, scenarios and code). According to Schmidt (2006), MDE is composed from two principal aspects: The Domain Specific Language (DSL) and the transformation engines and generators.

DOMAIN SPECIFIC LANGUAGE

Van Deursen *et al.* (2000) stated that it is a programming language or executable specification, it offers through appropriate notations and abstractions, an expressive power to a particular problem domain.

TRANSFORMATION ENGINES AND GENERATORS

It allow making some changes in the models by analyzing or extracting some aspects and then transform it to code or even another model. This transformation respects relationships and constraints specified in the DSLs. In this study, we mean by the model generation methods the transformation engines and generators.

As shown in Fig. 1, the core element of MDE is the model that represents a software system. The model should be conform to its specific metamodel. We will explain and define those notations in the next sections.

METHODOLOGY

Model: The use of model is not related to the recent science or to software engineering world. Regardless of the domain or the age of use the goal stay the same. It facilitates the representation of thoughts or concepts. Usually when we explain an idea we find words insufficient and the exhaustive description with all details is onerous.

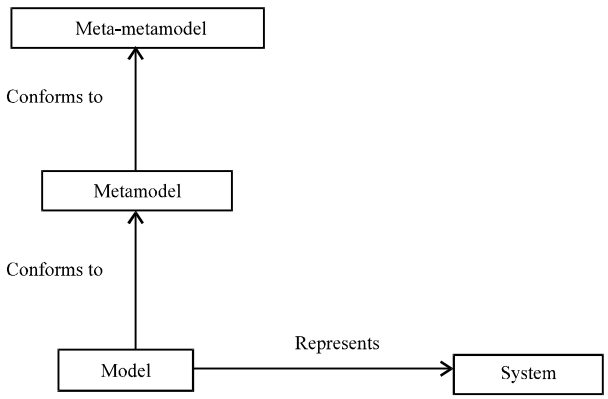

Fig. 1: Basic notations in MDE (Diaw *et al.*, 2008)

In literature, there are many definitions of the model, it can be defined as a "Simplification of a system built with an intended goal in mind. "The model should be able to answer questions in place of the actual system" (Bezivin and Gerbe, 2001). The model driven architecture MDA defines a model of a system as (OMG., 2003) "A description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language."

So a model is built in an area of representation that captures a set of common elements or concepts and establishes relationships between them.

Although, there's no consensus that define model but all the definitions agree that a model represents a semantic concept represented with a set of symbols or elements, each one explains a part of the concept and respect specific rules and notations. This is the role of the meta-model, the purpose of the next subsection.

Meta-model: A model is a representation of phenomena in the real world and a metamodel represents highlighting properties of the model itself (Di Ruscio, 2007).

The metamodel presents the entire possible notations to be instanced in the model. It defines the syntax of the modeling language. It can be considered as the representation of a class of all models expressed in that language. This model should conform to its metamodel like a program conforms to the grammar of the programming language (Bezivin, 2005). Metamodel in the context of MDA has a meta-meta-model expressed by the MOF from which we can instantiate many metamodel like the SPEM (Software Process Engineering Metamodel Specification).

METHODOLOGY OF COMPARISON

To better understand the transformation engines and generators in MDE, to classify and evaluate methods of model generation, we will extract the different criteria and establish a comparison framework. To ensure a methodical approach in this comparison, we adopt the "Four world framework."

Four world framework: The idea is to fix a domain and to discuss it with different points of view regrouped in four worlds: Nature, system, usage, development. Each view is composed in facets which represents criteria of comparison in a specific world. This framework was used by Rolland (1997) to compare the IS development process engineering and by Jarke *et al.* (1992) to compare namely information system engineering and requirement engineering (Jarke *et al.*, 1993). It was inspired from Prieto-Diaz (1991) to classify object-oriented components.

As shown in Fig. 1 we have determinate the different worlds in the framework comparison for the model generation methods (Fig. 2).

Nature world: Presents the different types of knowledge provided by the models generation methods.

System world: Defines the objectives of the model generation methods and the goals behind it.

Usage world: Specifies the different forms of model generation methods and the different representations used to define it.

Development world: Defines the process to elaborate the model generation methods and different phases.

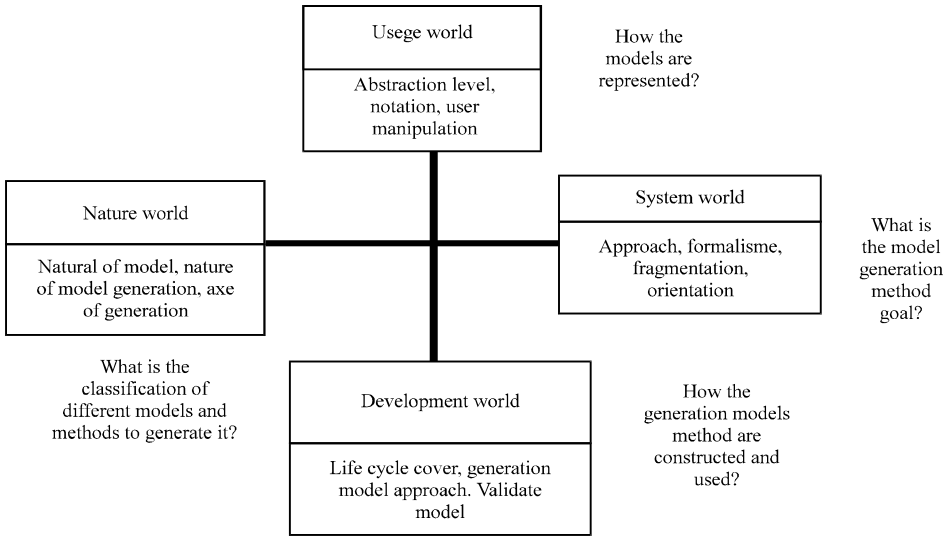

Fig. 2: Framework comparison for the model generation methods

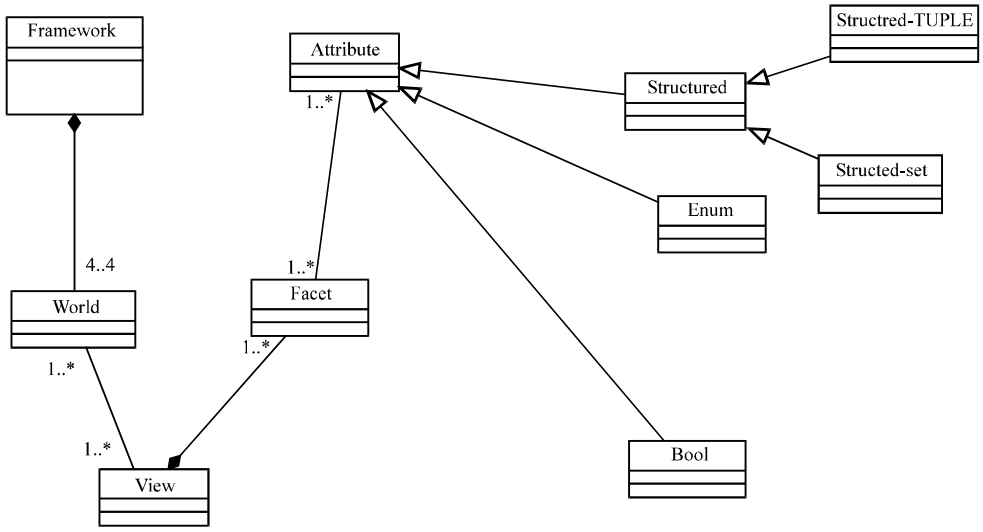

Fig. 3: Four world framework

By using the four world framework we aim to cover the different criteria detected in the state of art of the models generation methods and to be able to specify and reveal our contributions.

In Fig. 3, we represent the metamodel of comparison framework. We define four worlds. Each one is represented by a view which expresses a comparison criterion. A facet is a subject of classification and each one is measured by attributes defined in a field. It can be a Boolean attribute, or enumerated (ENUM {x, y, z}), or a structured type (SET or TUPLE).

In the next sections of study we will detail the four worlds and for each the appropriate facets then we will compare the UML models generation methods using this framework.

Nature world: In this section, we are interested to the nature of the model generation method and their classification. Accordingly, there are three facet in nature view namely the nature model, nature of model generation and nature of the axe modeling.

Nature model: The objective of those methods is to generate models. It's obvious that the first preoccupation in the nature world is to define the different types of models. As we defined in the previous section, a model is an instance of a metamodel which we express as generic domain knowledge using a specific notation. Basing on this definition, we can consider code as a type of a model. In fact code can be an instance of a metamodel. It represents the different possible combinations of a specific notation to achieve an executable semantic. In literature, we define two types of model generation; we can generate an executable model or an inexecutable model.

Executable model: It is a set of modeling elements that we can execute to visualize a result or to achieve a specific goal that can be a model basing on schema or an executable code. We consider the code as a model based on instructions and conform to a metamodel which the coding language is specified.

Inexecutable model: Graphic is a model based on the abstraction of elements with a special graphical notation. In general, it's constituted on understandable elements connected to express a concept or an idea but we can't execute it.

- Executable model: Boolean

Nature of method generation: In this facet, we deal with the type of the input method and its relation as the resulting model. In some methods, the input is described in the same modeling language of the output model. Where's in other methods it is not the case. Thus we need to ensure the mapping between the two modeling language metamodels.

Local generation: Means the transformation from a model to another that respects the same specific modeling language (for example from an UML diagram to another UML diagram). This type of transformation is widely used when we refine a model.

Extern generation: The input and output of a model generation method and the are instancieded from different metamodels, like from a description of system with XML to a line product model or from class diagram with UML to an entity association model. The type of transformation is more difficult than the local. We need to define the rules of transformation and the rules to match between these two metamodels. In this type of transformation we deal with two metametamodels, thus the transformation is more complicated.

- Generation model: (Enum local generation, extern generation)

Nature of the axe modeling: We mean by axe navigation on the abstraction level defined by MDA and presented in the usage world. When models are generated in the same level of abstraction in this case it is a local generation. If the translation from a level to another is allowed we call this generation a vertical modeling, for example from PIM to PSM or from PSM to code.

- Nature of the axe modeling: Set (Vertical, horizontal)

System world: In the system world, we answer the question "why the model generation methods were appeared?". We try to define the objectives and goals that these types of methods achieve. The

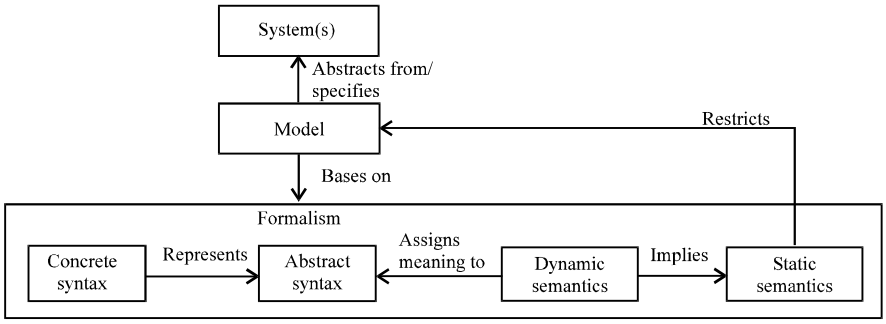

Fig. 4: System, model and formalism (Metzger, 2005)

model generation methods have different goals and are not related to a specific domain. The real challenge here is to define facets that are generic enough to cover all goals.

Approach: Kleppe (2009) does not differentiate program and model. He considers that the program is a model and uses the term "mogram" to express it. In fact program or code has a specific metamodel to respresent it and based on a set of notation elements named instructions. So we define in this facet two approachs: The model to model approach, that we will transform a model as the initial source and is the specification to model approach and aims to generate model from a specification source expressed in the natural language, or plain text. This can be represented as follow:

- Approach: Enum (model to model, specification to model)

Formalism: The model represents forms and notations to express in details an idea or some concepts with a defined meaning, so all the models contain two principal aspect the semantic and syntax. The semantic aims to clarify more the meaning or the comprehension of the subject. However the syntax aims to more illustrate some ideas or make it easier in extending it with more expressive syntax or notations. Thus, the need to generate models is in reality the need to represent more the semantic or the syntax or both of them.

But as shown in Fig. 4, (Metzger, 2005) classify the semantic or the syntax goal on static semantic, dynamic semantic, static syntax and dynamic syntax.

Semantic: There are two types of semantics: Static semantic and dynamic semantic:

Static semantic: Defines properties of models that can be determined without considering either inputs or execution.

Dynamic semantic: Named also semantic execution. It defines the model of computation for the behavior execution of models.

Syntax: There are two types of syntax: Static syntax and dynamic syntax.

Abstract syntax: The abstract syntax, signature or abstract data type of a language describes the basic structure (skeleton) of a language (Andova, 2012).

Concrete syntax: Specifies the readable representation of the abstract notational elements.

- **Formalism:** The TUPLE (Semantic (static semantic, dynamic semantic); Syntax (concrete syntax, abstract syntax))

Fragmentation: A fragment is a portion of model which presents a product or artifact (e.g., UML diagram). Some constraints specifying how and when a process included in a fragment can be started (Rougemaille *et al*., 2009). It's a dissembling of a model in a set of models to better express a semantic or an idea.

- Fragmentation goal: Bool

Orientation: There are principally two approaches.

Aspect oriented: Lee (2002) define aspect as a modular unit of crosscutting implementation. The aspect oriented programming means that each aspect can be expressed in a natural form and represents an autonomic module that executes a set of procedures or functions. The aspects can finally be merged together to achieve a goal and this way we increase reusability of codes.

Intention oriented: This approach represents a decomposition of the code or models using the goals. Each element is represented to achieve a mission or intention. We call this strategy the intention oriented fragmentation.

- Orientation: SET (aspect oriented, Intention oriented)

Usage world: As we indicate previously, this view defines the different representation and forms of UML fragments.

Abstraction level: Modeling in general is an abstraction of concepts. We can represent all the details of concept, ever the technical part or just introduce the idea without any indication of the framework or tools used to support it. So the levels of abstraction of a model change with the intention or goal of the users. In this context MDE defines three levels of abstraction modeling:

CIM: Computation Independent Model (CIM) presents the specification of system functionalities from a computation independent viewpoint. CIM expresses the first view of the system without any indication of how the functionalities can be realized (Czarnecki and Helsen, 2006).

PIM: Represents the system from the platform independent viewpoint. A PIM exhibits specified degree of platform independence. It could be suitable for use with a number of different platforms of similar types (Bezivin and Gerbe, 2001).

PSM: Represents the platform specific viewpoint of the system. It integrates the details of technology used in the PSM model.

- Abstraction level: Enum (CIM level, PIM level, PSM level)

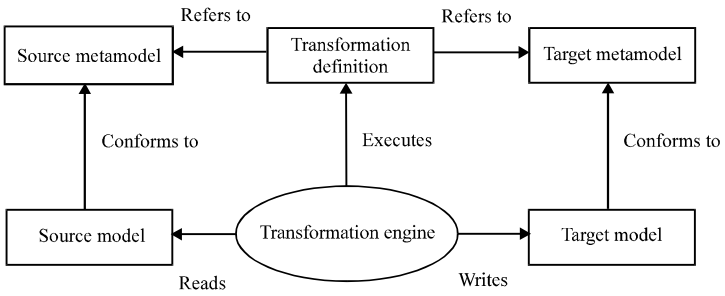

Fig. 5: Model transformation process (TOPCASED-WP5 Staff, 2008)

Standardized notation: The standardized notation determines the type of notation used. This facet is interested if the notation is conform to a standard like UML or its own notation not standardized like software product line (in this case we name it proprietary). For some models we can find the proprietary notation with injection of some standardized elements, we consider this notation as mixed.

- Standardized notation: ENUM (standard, proprietary, mixed)

User manipulation: This facet was inspired from the classification of the transformation model approach presented in (Di Ruscio, 2007). We define two types of model generation approahes, the direct manipulation approach and the operational approach.

Direct manipulation approach: Users specify the characteristics of transformation with a direct interaction. They may implement transformation rules, scheduling, etc.

Operational approach: We can consider it as a direct manipulation approach with a dedicated support for the users. To ensure this transformation we need to extend the meta-model with some new concepts.

- User manipulation: ENUM (Direct manipulation approach, operational approach)

Development world: This view focuses on different activities of model generation method's development in terms of process adopted to generate models.

Life cycle cover: According to Diaw *et al.* (2008), the model generation methods process involves two phases. The first identifies the correspondences between the concepts of source and the target model in their meta-models. It indicates the existence of a transformation function applicable to all instances of the meta-model source. The second phase applies the transformation of the source model to automatically generate the target model by a program called processing motor or execution motor. Figure 5 illustrates these two phases of model generation.

- Life cycle cover: SET (transformation definition, transformation engine)

Generation model approach: There are three types of generation model approach: The programming approach, the template approach and the modeling approach.

Programming approach: Uses programming languages and especially object-oriented language. In this approach, the transformation is described as a computer application. This approach is widely used. It exploits the accumulated experience and equipment of existing languages (Diaw *et al.*, 2008).

Template approach: The objective is to define the canvas target models desired. These approach use targets parameterized models or models template. The execution of the model generation is to take a model template and replace its parameters by the values of the source model.

Modeling approach: Is based on the model to model engineering concepts. The objective is to generate model from a source model and make models sustainable and the transformation productive (Diaw *et al.*, 2008).

Generation approach: The SET (the programming approach, the template approach, the modeling approach).

Model validate: As any construction process, we need to validate the final product to verify if it's conforming to initial objectives. In the model generation method, we validate if the model is conforming to the meta-model target, expresses the idea and the semantic aimed for. So this facet presents the validate phase in the generation of models.

- Validate model: BOOL

EXPERIMENTATION

The purpose of the comparison is to highlight the generation of UML fragments by using SPL strategy. To accomplish this mission, we have chosen in this experimentation only tools that use the UML standard. So in this section we give a brief overviews of some tools that allows the generation of models and we will use them to make a comparison with our framework. We will detect the variations between these tools and the differences between it.

In this sub-section, we represent four model generation tools: FUML, UMLAUT, FUJABA and IRSM. As we said previously, we have chosen tools that use UML diagram.

FUML: Foundational subset for executable UML models: The FUML was created by the Object Management Group (OMG) to resolve the UML ambiguity problem. In fact the ambiguity on UML is provided from the behavior aspect. We can not validate the semantic of the UML diagram especially in design if it was satisfied at the end of the process development. The solution proposed by the FUML is to define the semantic with a specific language and to generate code from a validated diagram.

One of the purposes of using UML as a graphical modeling language is to abstract ideas, facilitate them and understand them. The final target in the process development is to construct a program that satisfies the specification and implements the design. One of the bases of UML is the semantic aspect. This aspect is based in UML on behavior modeling by an activity diagram, state machine, etc. FUML aims to validate the semantic aspect presented in general by the activity diagram and generate the corresponding code. But these programming languages are not enough at a high level of abstraction to facilitate discussions about design.

For this reason, we need an automatic mapping from one level of abstractions to another. This would be useful in the way that they could accelerate the development of software systems, helping to validate models and generating code at the same time the system is designed.

To do this, OMG needs to express this semantic with a specific langage, named DSML (Domain Specific Modeling Langage). All the concepts introduced by the DSML in FUML must be conform to MOF. Also we need in this case to define the semantic with a new meta-model . The OMG chooses to use KERMETA, executable meta-modeling language that incorporates aspect-oriented features UMLAUT.

UMLAUT: In UML there are many tools that help designer to extend his diagrams with some operations. Those tools are based on a syntactic generation from a simple class diagram to another with methods from the basic diagram but still difficult to add some complex operations like the design patterns. This type of tools allows the transformation from a model to a more expressive and detailed one. However, didn't control or validate the elements generated in the new model like methods. It is focalized on the syntactic aspect but not the semantic. So in general the result is syntactically correct but semantically not (Ho *et al.*, 1999). In this context UMLAUT appears as a tool to generate design model from another and to express and validate that this transformation is conformed to the initial specifications.

UMLAUT is based on algebraic to generate the transformation. The specification is represented by graphs. Each node represents a diagram and from each node we can re-visit the same node that means a "Refinement" or visit another and that means a transformation. To ensure the syntactic aspect, UMLAUT associates for each node its corresponding UML metamodel.

IBM Rational Software Modeler (IRSM): IBM has three tools of modeling: Web Sphere Business, Modeler Rational Software Modeler/Architect and Rational Rose. We are interested in this section, Rational Software Modeler. It aims to represent different UML diagrams and to transform them. With RSM, users can elaborate any metamodel and define the rules of transformation. So it offers an extern generation of a model and an operational manipulation approach.

From UML to java and back again (FUJABA): FUJABA aims to generate a java code from an UML diagram or the retro-design by generating UML diagram from java code. It is classified as an open source CASE tool providing developers with support for model-based software engineering and re-engineering (http://www.fujaba.de/about-fujaba.html).

RESULTS AND DISCUSSION

We apply the comparison framework on tools presented in the previous subsection. We represent the comparison between UMLAUT, FUML, RSM and FUJABA in the four worlds, represented separately in the Table 1-4.

Table 1: Nature world

Tool/Facet	Nature of model	Nature of model generation	Generation axe
UMLAUT	False	Local	Vertical
FUML	False	Local	Vertical
RSM	False	Extern	Vertical
FUJABA	True	Extern	Vertical

Table 2: System world

Tool/Facet	Approach	Formalism	Fragmentation goal	Orientation
UMLAUT	Model to model	Semantic (static semantic, dynamic semantic)	False	Aspect oriented
FUML	Model to model	Syntax (abstract syntax), Semantic (static semantic)	False	Aspect oriented
RSM	Model to model	Syntax (concret syntax, abstract syntax), Semantic (static semantic, dynamic semantic)	False	Intention orientated and aspect oriented
FUJABA	Model to code	Syntax (concret syntax, abstract syntax), Semantic (static semantic, dynamic semantic)	False	Intention orientated and aspect oriented

Table 3: Usage world

Tool/Facet	Abstraction level	Standardized notation	User manipulation
UMLAUT	PIM, PSM	Standard	Direct manipulation approach
FUML	PIM	Standard	Direct manipulation approach
RSM	PIM, PSM	Mixed	Operational manipulation
Fujaba	PSM	Mixed	Operational manipulation

Table 4: Development world

Tool/Facet	Life cycle cover	Generation model approach	Model validation
UMLAUT	Transformation engine	Programming approach and modeling appoach	True
FUML	Transformation engine	Modeling approach	True
RSM	Transformation definition, transformation engine	Template approach programming approach	False
FUJABA	Transformation engine	Programming approach	False

Table 5: Nature world (Present study)

Nature of model	Nature of model generation	Generation axe
False	Local	Vertical, horizontal

Table 6: System world (Present study)

Approach	Formalism	Fragmentation goal	Orientation
Model to model	Semantic (static semantic, dynamic semantic))	True	Intention oriented

Table 7: Usage world (Present study)

Abstraction level	Standardized notation	User manipulation
PIM, PSM	Standard	Direct manipulation approach

Table 8: Development world (Present study)

Life cycle cover	Generation model approach	Model validation
Transformation engine, transformation definition	Modeling appoach	False

The result of the comparison framework application on the four tools UMLAUT, FUML, RSM and FUJABA was illustrated in the previous tables. To locate our contribution, we applied the four world framework on our generation model method. The result is presented in Table 5-8. We detected many differences and commonalities. As presented in Table 2, those tools do not aim to decompose models and generate them as fragments. In some cases, users need just to generate a part of a model and this deficiency will be our contribution. Our study is based on fragmentation (Table 6), we develop method that automates the composition of reusable fragments of UML model. The use of the Software Product Line guides both the composition and the fragmentation of the UML diagram. As we presented in (Bouhaouel *et al.*, 2013), we defined an approach to cover the

transformation engine and the transformation definition (Table 8). As shown in Table 6, our method is oriented intention, thus we used the map to guide the semantic knowledge of UML diagram and SPL for the syntax knowledge (notation). The four methods presented in the study are restricted in a vertical transformation. Our method covers the vertical and the horizontal transformation (Table 5). It's based on generation diagram using a set of fragmentation criteria, one of them is the goal of fragmentation. That can be a refinement goal (horizontal generation) or a vertical transformation goal.

CONCLUSION

This study defines a comparison framework of model generation methods. We have focused to present a state of the art of MDE and to elaborate a comparative framework based on four worlds: Usage, nature, system and development. Then we have used it to classify four model generation methods (UMLAUT, FUML, RSM, FUJABA) according to facets. This comparison framework allows us to define explicitly our contributions. Indeed, we attempt to construct a method that generates UML fragments. We will use the software product line as a reuse approach. Therefore, we will elaborate in our future studies the product line model of UML diagrams. This approach refers to model transformation specially the model to model transformation approach.

ACKNOWLEDGMENT

The study was supported by The Research Council (TRC) Oman.

REFERENCES

Andova, S., 2012. MDE basics with a DSL focus. Proceedings of the 12th International School on Formal Methods for the Design of Computer, Communication and Software Systems, June 18-23, 2012, Bertinoro, Italy, pp: 21-57.

Bezivin, J. and O. Gerbe, 2001. Towards a precise definition of the OMG/MDA framework. Proceedings of the 16th Annual International Conference on Automated Software Engineering, November 26-29, 2001, Coronado, CA, USA., pp: 273-280.

Bezivin, J., 2005. On the unification power of models. Software Syst. Model., 4: 171-188.

Bezivin, J., 2006. Model Driven Engineering: An Emerging Technical Space. In: Generative and Transformational Techniques in Software Engineering, Lammel, R., J. Saraiva and J. Visser (Eds.). Springer, Berlin, Heidelberg, ISBN: 978-3-540-45778-7, pp: 36-64.

Bouhaouel, R., N. Kraiem and Z. Al-Khanjari, 2013. Labeled UML model fragments composition by the SPL strategy. Int. J. Comput. Technol., 10: 2049-2056.

Czarnecki, K. and S. Helsen, 2006. Feature-based survey of model transformation approaches. IBM Syst. J., 45: 621-645.

Di Ruscio, D., 2007. Specification of model transformation and weaving in model driven engineering. Ph.D. Thesis, University of L'Aquila, Italy.

Diaw, S., R. Lbath and B. Coulette, 2008. [State of art: Model driven engineering] TSI-X/2008. The Model-Driven Engineering.

Ho, W.M., J.M. Jezequel, A. Le Guennec and F. Pennaneac'h, 1999. UMLAUT: An extendible UML transformation framework. Proceedings of the 14th IEEE International Conference on Automated Software Engineering, October 12-15, 1999, Cocoa Beach, FL., pp: 275-278.

Jarke, M., J. Mylopoulos, J.W. Schmidt and Y. Vassiliou, 1992. DAIDA: An environment for evolving information systems. ACM Trans. Inform. Syst., 10: 1-50.

Jarke, M., K. Pohl, S. Jacobs, J. Bubenko and P. Assenova *et al.*, 1993. Requirements engineering: An integrated view of representation, process and domain. Proceedings of the 4th European Software Engineering Conference, September 13-17, 1993, Germany, pp: 100-104.

Kleppe, A.G., 2009. Software Language Engineering: Creating Domain-specific Languages Using Metamodels. Addison-Wesley, New York, ISBN: 9780321553454, Pages: 207.

Lee, K.W.K., 2002. An introduction to aspect-oriented programming. COMP 610E 2002 Spring Software Development of E-Business Applications, The Hong Kong University of Science and Technology, Hong Kong.

Metzger, A., 2005. A Systematic Look at Model Transformations. In: Model-Driven Software Development, Beydeda, S., M. Book and V. Gruhn (Eds.). Springer, Germany, ISBN: 978-3-540-25613-7, pp: 19-33.

OMG., 2003. MDA guide version 1.0.1. Document No. OMG/2003-06-01, Object Management Group, June 12, 2003.

Prieto-Diaz, R., 1991. Implementing faceted classification for software reuse. Commun. ACM, 34: 88-97.

Rolland, C., 1997. A primer for method engineering. Proceedings of the Conference INFORSID (Informatique des Organisations et Systemes d'Information et de Decision), June 10-13, 1997, Toulouse, France, pp: 1-26.

Rougemaille, S., F. Migeon, T. Millan and M.P. Gleizes, 2009. Methodology fragments definition in spem for designing adaptive methodology: A first step. Proceedings of the 9th International Workshop on Agent-Oriented Software Engineering, May 12-13, 2008, Estoril, Portugal, pp: 74-85.

Schmidt, D.C., 2006. Model-driven engineering. IEEE Comput., 39: 25-31.

Soley, R. and The OMG Staff Strategy Group, 2000. Model driven architecture. Object Management Group White Study, Draft 3.2-November 27, 2000. http://www.geocities.ws/pravin_suman/Resources/00-11-05.pdf.

TOPCASED-WP5 Staff, 2008. [Methodological guide for model transformations]. In IRIT/MACAO.

Van Deursen, A., P. Klint and J. Visser, 2000. Domain-specific languages: An annotated bibliography. ACM SIGPLAN Notices, 35: 26-36.

Voelter, M. and I. Groher, 2007. Product line implementation using aspect-oriented and model-driven software development. Proceedings of the 11th International Software Product Line Conference, September 10-14, 2007, Kyoto, Japan, pp: 233-242.