ISSN:2320-0790

An International Journal of Advanced Computer Technology

Model-Based Software Engineering (MBSE) and Its Various Approaches and Challenges

Reema Sandhu

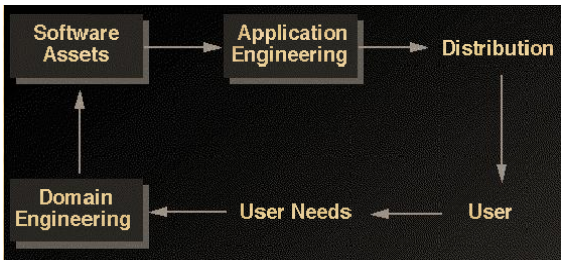
Assistant Professor, Dept. of Computer Science
Dr. B.R.A Govt. College Kaithal (Haryana)

Abstract: One of the goals of software design is to model a system in such a way that it is easily understandable. The use of model-based software development is increasingly popular due to recent advancements in modeling technology. Nowadays the tendency for software development is changing from manual coding to automatic code generation thus relieving the human from detailed coding. This is a response to the software crisis, in which the cost of hardware has decreased and conversely the cost of software development has increased sharply. This paper presents the drastic changes related to modeling, different approaches and important challenging issues hat recur in MBSD. New perspectives are provided on some fundamental issues, such as the distinctions between model-driven development and architecture-centric development, code generation, and Meta modeling. Achieving a positive future will require, however, specific advances in software modeling, code generation, and model-code consistency management.

Keywords: Model-Based Software Development, Model-Driven Development, Architecture-Centric Development.

I. Introduction

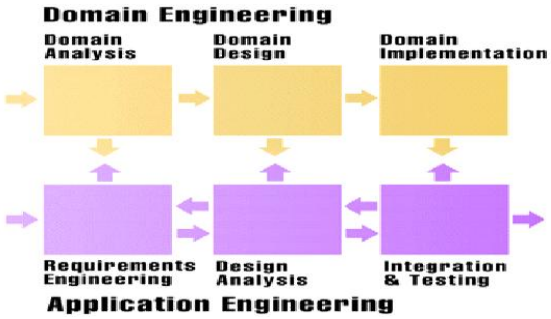
Model-Based Software Engineering (MBSE) is defined as reusing the code and performing maintenance and development of software through the use of software modeling technology. It improves the quality and productivity of the software, given that software models follows the principle of abstraction by hiding away certain implementation details and defines the problem domain relative to programming languages .The system described by a model may or may not exist at the time of model creation. Models are thereby created to serve particular purposes, for example, to present a human understandable description of some aspect of a system or to present information in a form that can be mechanically analyzed. This process is being successfully implemented in many domain areas and is evolving continuously. Many tools and technologies available today use and support MBSE. It splits the production of software into two parallel engineering processes namely *domain engineering* and *application engineering*. One area of MBSE delivers applications already made to the customers and the other uses the synthesized features to build new customized applications for other customers. The development cycle is shown in figure 1 below.



Domain Engineering: - Domain Engineering (DE) is defined as the process of developing the reusable components of software and organizing its architecture after analyzing the requirements of a particular domain. Domain refers to the set of functional areas covered by a group of application systems that have similar software requirements. The process of domain engineering is shown in figure 2. It comprises of three main stages, the first being domain analysis, then domain design and lastly domain implementation. DARE-COTS tool is used

for Domain Analysis. At an initial stage, it is mandatory to get the universal and variable characteristics of group systems in a particular domain. Domain analysis model can be generated by abstracting certain characteristics. The domain specific software architecture can be designed based on this model and then the reusable components will be generated and organized. Thus, when developing a new system in a new domain, we have to identify the system's requirements and specifications as per the domain model, and can generate the new design as per the Domain Specific Software Architecture (DSSA), and then select the particular components to assemble the new system.

Application Engineering: - The process of developing the single application system is called Application Engineering. It shows how to develop an open and reusable product quality tracking system on the basis of domain engineering. It reuses the major functionality of the system for the application developed in a similar domain for which the reusable components are available. In the case of application engineering we identify the following three steps: Requirements Engineering, Design Analysis and Integration & Testing.



Approaches to MBSE:- Model-based development approaches to MBSE can be roughly classified on the primary abstraction level of their focal software model as follows.

i) **Model Centric Software Development** (MCSD):- This model is a knowledge hub for the SDLC. It lays emphasis on the use of concise and expressive models in the development process to express the relevant concepts of each area such that they become transparent and can be used in other areas. This approach to software development is being used since many years but generation of executable code from implementation-level models is an area of special interest. MCSD has a much broader scope and areas such as business process modeling, architectural models, or enterprise-wide federated repositories etc.

ii) **Model Driven Development (MDD) and Architecture Centric Development (ACD):-** This approach typically focuses on software design models. MDD and ACD both rely on the machine for generation of complete code from software artifacts of a higher-level abstraction. There lies a difference between the MDD and ACD. The rationale behind MDD is to make software design models compliable and executable, so that software developers can solely focus on abstract models. To achieve this goal, software models must have sufficient detail to enable full code generation. Whereas, ACD uses software architecture as the blueprint where principal design decisions are laid out. Its code generation process is primarily about generating architecture-prescribed code. This application code needs software developers to fill in details. The uses of UML in these two approaches are actually in different modes: UmlAsProgrammingLanguage in MDD and UmlAsBlueprint in ACD.

iii) **Specification-driven development:-** It uses requirements specifications for creation and direct execution of applications. It appears in two forms: transformational programming and application generation. Transformational programming is a methodology of constructing a program by successive applications of transformation rules. It starts with a formal statement of a problem, and ends with an executable program. Application generators are tools used for creating a family of applications, and are deeply rooted in domain engineering. An application generator translates a highly-particularized specification in a domain into a complete implementation. To change or modify a product, one simply changes input specifications and reruns the generator.

iv) **Generative and Component-based approaches:-**This model generates code called glue code to combine existing components into the final artifact. The composition of components combines two or more software components and yields a new component behavior at a different level of abstraction. Functional composition and multi-dimensional composition are two distinguished approaches to composition of components. The functional composition breaks up a complex software system into smaller components with functional relationships as the

primary criterion, while the latter emphasizes separation of overlapping concerns along multiple dimensions of decomposition. A typical example of functional composition is generative software development, which focuses on automating the selection and assembly of components. Multi-dimensional composition distinguishes the notion of core components from concerns.

Challenges faced by MDSE:-

i) **Multi-Aspect Modeling : -** In MBSD software models not only have to contain enough details to generate relatively complete code, but also need to be simple than the software programs created during the process. Existing behavioral modeling methods are those which are based on formal notations and those that are more informal, but with a practical bias. None however, provides an appropriate form for MBSD. Formal behavioral modeling methods use the process of algebras like CSP and the pi-calculus. Providing a basis for automatic analysis is one of their main purposes. Formal behavioral modeling methods are more appropriate for software development because of their limited expressiveness and in most of the cases, developers would rather write code directly. Examples of more informal methods include interaction diagrams, state diagrams, and activity diagrams of UML Informal methods were used traditionally used for communication and system comprehension. Due to their incompleteness properties they cannot be used alone for behavioral modeling in MDD, which emphasizes complete modeling. In many cases where only executions of significance are concerned, such as architecture-centric development, practical methods like sequence diagrams may be a good choice after some form of extension .

ii) **Code Generation:-**. MBSD faces another big challenge which requires structural code, behavioral code, or even non-functional code to be automatically generated from source models. This is difficult because nonstructural modeling in MBSD is not yet mature and also system dynamics are involved. Many more variations need to be considered as compared with static structural code generation.

iii) **Model-Code Consistency Management:-** After the process of code generation either the source model has to be modified again or the developers should do additional editing in the generated code. These changes endanger the conformance established between the model and code. Successful solutions are already available to handle changes in the model, guaranteeing that extra work done on the generated implementation remains as such when the system is regenerated. This is usually done through code markers in the form of comments. There are two types of approaches based on inconsistencies that occur during this process - correct-by-construction and correct-by-detection. Some inconsistencies may be too expensive to be detected and resolved. They can further be divided into one-way mapping and two-way mapping, depending on which artifact can be manually changed. The correct-by-detection approaches are usually used to map updated code to model, and assume the relative constancy of model. This explains why there are no two-way mappings of correct-by-detection. Correct-by-construction approaches are extensively used in MBSD to avoid inconsistency from the very beginning. One-way mapping approaches among them try to generate complete code, so that manual modification of code is not a necessity and chances of inconsistency can be reduced. Two-way mapping include separation of generated and non-generated code, architecture frameworks, and the adoption of new implementation strategies. These can only enforce structural conformance between model and the code. The use of round-trip engineering is a new trend in this area, where traceability links between model and code are used to automatically propagate updates in derived code back to the model. In particular, a successful utilization of round-trip engineering in complex software development is still missing. Correct-by-detection approaches address the conformance issue through after-the fact consistency checking done either through reverse engineering based static analysis or runtime monitoring verification. Reverse engineering abstracts source models from modified implementations, and compares the original source model with the generated one. It can be expensive for complex systems; moreover, it is hard to guarantee that the generated model captures the same aspects that the original source model contains, since they may represent two different abstractions of the same implementation. Runtime monitoring approaches infer the system architecture from execution traces or system events that are collected at runtime. They are favorable in terms of being able to check the system behaviors against the original architecture. To do this, the availability of executable software is usually required. Some approaches also demand certain forms of code instrumentation. This prevents dynamic verification from being used at development time, when programs are often not complete enough to be executed.

REFERENCES

[1] Janos Sztipanovits. "Model-based Software Development". ESMD-SW Workshop, NASA, March, 2007.

[2] Youxin Meng, Xinli Wu, Yuzhong Ding," Research and Design on Product Quality Tracking System Based on DomainEngineering", IEEE, 2010.

[3] H. Stachowiak. Allgemeine Model ltheorie. Springer- Verlag Wien, 1973.

[4] D. Harel and B. Rumpe. Modelling languages:Syntax, Semantics and all that stuff , IEEESoftware, 2004.

[5] France, R. and Rumpe, B. 2007. Model-driven Development of Complex Software: A Research Roadmap. In 2007 Future of SoftwareEngineering (May 23 - 25, 2007). IEEEComputer Society, Washington, DC, 37-54.

[6] Balzer, R. 1985. A 15 Year Perspective onAutomatic Programming. IEEE Trans. Software Engineering. 11, 11 (Nov. 1985), 1257-1268.

[7] Selic, B. 2003. The Pragmatics of Model-Driven Development. IEEE Softw. 20, 5 (Sep. 2003), 19-25.

[8] N Md Jubair Basha, Salman Abdul Moiz, A.A Moiz Qyser, " Performance Analysis of HR Portal Domain Components Extraction ", International Journal of Computer Science & Information Technologies (IJCSIT), Vol2 (5),

1.

[9] William Fakes, Ruben Prieto- Diaz, Christopher Fox, "DARE-COTS: A Domain Analysis Support Tool", IEEE, USA,
1997.

[10] Massimo Fenarlio, Andrea Valerio, "Standardizing Domain- Specific Specific Components: A Case Study", ACM, Vol. 5, No.2,June, 1997.

[11] P. Clements, F. Bachmann, L. Bass et al., Documenting Software Architectures: Views and Beyond: Addison Wesley, 2002.

[12] Matinlassi, M., Niemelä, E, Dobrica, L. 2002. Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture. Espoo, VTT Publications

[13] Kleppe, A. G., Warmer, J., and Bast, W. 2003 MDA Explained: the Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc.

[14] Czarnecki, K. and Eisenecker, U. W. 1999. Components and generative programming (invited paper). SIGSOFT Softw. Eng. Notes 24,6 (Nov. 1999), 2-19.

[15] Kelly, S., Tolvanen, J-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Society Press, 2008.

[16] ArchStudio 4: http://www.isr.uci.edu/projects/ archstudio/

[17] Hailpern, B. and Tarr, P. 2006. Model-drivendevelopment: the good, the bad, and the ugly. IBM Syst. J. 45, 3 (Jul. 2006), 451-46