Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 April 2021 doi:10.20944/preprints202104.0028.v1

A Conceptual Introduction to Model Driven Development: From Softness to Complexity

Mohammad Reza Besharati[1], PhD Candidate, Sharif University of Technology, Tehran, Iran

Mohammad Izadi, Sharif University of Technology, Tehran, Iran

Abstract

In Model-Driven Development (MDD), the models, their generation, and imposing changes on them (model transformation) are used for the development of software. Models provide a framework to start from the imagination and abstraction to create and accomplish the final system. Models create a slow and steady transition from whatness to howness, i.e. from the natural path of the generation of software. For supporting this path, the Logic and Functionality of software must be changeable during its evolution. Here we provide a brief introduction to the concept of Model Driven Development.

Keywords

Software Engineering, Model, Model-Driven, Model Driven Development, MDD, MDA

Introduction

In Model-Driven Development (MDD) [1], the models, their generation [2], and imposing changes on them (model transformation) are used for the development of software. Models create a slow and steady transition from whatness to howness [3], [4], [5], i.e. from the natural path of the generation of software. In the general sense, even code is a kind of model [2]. Once more in the general sense, a model can be described and imagined as the surroundings of an entity (a niche in the context [6] world), which is located in a specific level of abstraction. Model and modeling are intertwined with the concept of abstraction.

An entity cannot be acknowledged or created if not imagined. Models provide a framework to start from the imagination and abstraction to create and accomplish the final system. Therefore, it is only natural to use models and modeling to generate systems.

The industry is a collective and social phenomenon. The proverb “many hands make light work” does not apply to the industry. Thus, common terms, tools, procedures, and methods must be defined for the activists of each industrial field so that they can interact synergistically. There are a variety of standards for MDD and the one which is most popular is MDA. The relation of

[1] besharati@ce.sharif.edu

[2] Platform-Specific Model (PSM)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 April 2021 doi:10.20944/preprints202104.0028.v1

MDA to MDD is that MDA is a standard in the field of MDD, which defines a conceptual architecture to achieve model-driven development.

There are three levels of the MDA Model. The first level is Computational Independent Model (CIM), the second level is the Platform-Independent Model (PIM), and the third level is the Platform Specific Model (PSM). In the development of a system, the work starts in the models of the CIM level. Then, it is transformed into the PIM level. Finally, the PIM models are transformed into PSM (the most important of which is the final code of the system).

If imposing changes on a model leads to the transformation in its level, then, there will be a vertical transformation of the model. In the horizontal transformations, there is no level transformation, i.e. there is merely a rearrangement of the model in the same level.

The reverse transformations - for instance, from PSM to PIM and from PIM to CIM - are crucial in software engineering as well (both in the creation of a system and in the re-engineering of an existing system). Their least advantage is the determination of the line and relation of the system components (i.e. different sections of design and code) to the requirements. Therefore, being reciprocal is a crucial concern regarding model transformation.

Model-Driven Development is not limited to software Development Phase. Any software (in all of its evolution life-cycle stages) experiences change and due to these changes “path of love seemed easy at first, what came was many hardships” (everything seemed easy at first but then there was the hardship). The multitier architecture of MDA together with the model transformation concept is a great framework for supporting change.

Besides, by automatizing the process of model transformation, the Automatic Code Generation and Generative Software Development can be achieved. Thus, model-driven development can lead to the automatic generation and development of software. It’s a very important issue for today’s software development (for example: for cyber physical systems [8], smart contracts [9], mobile computing [10], legacy to cloud-migration [11], microservices [12], etc.).

To enumerate the development in the field of model-driven development, it must be noted that at first, MDD was proposed as a general approach, then, MDA was suggested as a standard for achieving MDD by OMG. Then it was expanded. Meaning that the researchers stated that we need to surpass this stage and focus on the MDE, i.e. Model-Driven Engineering. The last development was proposing MD*. * can be replaced by any significant concern and concept in the field of software engineering (for example: model driven self-adaptation [7]).

Some of the Applications of MDD

One of the most evident applications of MDD in the industry is resolving problems pertinent to API evolution and the soft framework of software. You must have seen a message when installing software that asks you to install the latest version from a software framework or platform (e.g. .Net Framework). The software’s dependence on a soft platform to be developed

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 April 2021 doi:10.20944/preprints202104.0028.v1

on is a tangible reality. However, the problems get serious when you put yourself in software developers' place.

The software in a company or organization, even in case they have not lost their maintenance capability, might turn into worthless codes within only a few months. Since there might be a technological change in the market and industry and our unfortunate software, will become the clear example of the Ugly Duckling. (The rapid growth of the Server-side JavaScript technologies such as Node.js and the pains that will be suffered by PHP is an example of this problem, i.e. the disaster that awaits PHP in the future). I assume that you too are of the opinion that "the only constant in life is change" and this constant change has turned into a real nightmare for a company that invested all its capital and years of work in developing a single product or a software solution. So what can we do? The question is not "Who Moved My Cheese?" However, do we have to beware of the changes so that we can have an unfair competition with the changes? Or we can use MDD? The second solution is close to engineering.

Nothing is constant in the software world but change itself. Meanwhile, there is no reason to succumb so soon and ruin everything. Some changes occur in the long-term and some in the short-term. The short-term changes can be managed through the relative stability resulting from the long-term changes. What is that mean? It means that humans are selectors. We must rely on the dependencies of our software on the long-term changes and not on the transient technologies and concepts. Do you need to ask again what that means? That means MDD. It means that the concepts and rules scarcely change in the field of the problem domain. At least, their change rate is slower than the concepts and technologies pertinent to the soft platform of software.

An accounting software, audit and controller, and .Net Framework 4.5… Which concepts become outdated faster and change? The last one if you ask me.

No soft platform has ever remained constant and unchanged. Since it is soft plus it is alive as long as it can change. When it loses its ability to change, it will be replaced by a newly emerged rival.

It is not just the soft platforms that change. Most modern systems are a combination of software and hardware components. The software is designed based on the hardware. Therefore, there is a strong dependency between the hardware and software platforms. What happens if the hardware changes and updates every day and passes an evolutionary path? It is a reality that exists in the modern system and hardware components can be upgraded and updated several times during the lifespan of a system. Does the change of changes have to continue with each update to lead to the software changes of the system? It can be a good solution. However, it can result in extra and considerable costs. You can fix what ain't broke, but why?
The three-layer MDA and transformation of the models can be helpful in this regard. If the software layer of the system is considered and designed on the basis of MDA architecture, there should be no worries regarding updating the hardware: the domain of changes will exceed up to Platform Specific Model, and the Platform Independent and Computation Independent will face no changes. Only the transformers between layers 2 and 3 (i.e. PIM into PSM transformers) are required to be updated by each update, which is not difficult if the modularity teachings and

object-orientation are observed in the development of these models: with a modular or object-oriented software (i.e.PIM into PSM transformer), not only there should be no concerns regarding change but it can be considered as a good opportunity. Since it was designed and developed to be changed and based on the idea of changeability.

If the interlayer transformer software is not under complexity management [2] (=not modular and object-oriented), they will not have the ability to support changes. Software that lacks complexity management is harder than iron:

If you wish to see the beautiful face of your love, clear your heart
Otherwise, no flower or daffodil grows in iron or zinc

The logic [3], meaning [4] and functionality [5] of software must be changeable like the picture on the mirror. A mirror becomes smooth by polishing and software, by object-orientation and complexity management.

References

[1] OMG. MDA Success Stories (web page). <http:// www.omg.org/mda/products_success.htm>

[2] Besharati, M.R.; Mostafazadeh, M.; Izadi, M. Complexity Management Patterns and Generative Software Development. Preprints 2021, 2021030651 (doi: 10.20944/preprints202103.0651.v1).

[3] Spivey, J. Michael, and J. R. Abrial. The Z notation. Hemel Hempstead: Prentice Hall, 1992.

[4] Putnam, Hilary. "The meaning of 'meaning'." Philosophical papers 2 (1975).

[5] Van Lamsweerde, Axel. Requirements engineering: From system goals to UML models to software. Vol. 10. Chichester, UK: John Wiley & Sons, 2009.

[6] Moradi, Hossein, Bahman Zamani, and Kamran Zamanifar. "CaaSSET: A framework for model-driven development of context as a service." *Future Generation Computer Systems* 105 (2020): 61-95.

[7] Feyzi, Farid. "Model-driven development of self-adaptive multi-agent systems with context-awareness." *International Journal of Computer Aided Engineering and Technology* 12, no. 2 (2020): 131-156.

[8] Mohamed, Mustafa Abshir, Geylani Kardas, and Moharram Challenger. "A Systematic Literature Review on Model-driven Engineering for Cyber-Physical Systems." *arXiv preprint arXiv:2103.08644* (2021).

[9] Azzopardi, Shaun, Christian Colombo, and Gordon Pace. "Model-Based Static and Runtime Verification for Ethereum Smart Contracts." In International Conference on Model-Driven Engineering and Software Development, pp. 323-348. Springer, Cham, 2020.

[10] Gharaat, Mohammadali, Mohammadreza Sharbaf, Bahman Zamani, and Abdelwahab Hamou-Lhadj. "ALBA: a model-driven framework for the automatic generation of android location-based apps." *Automated Software Engineering* 28, no. 1 (2021): 1-45.

[11] reza Bazi, Hamid, Alireza Hassanzadeh, and Ali Moeini. "A comprehensive framework for cloud computing migration using Meta-synthesis approach." Journal of Systems and Software 128 (2017): 87-105.

[12] Di Francesco, Paolo. "Architecting microservices." In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 224-229. IEEE, 2017.