UDC 20.53
IRSTI

https://doi.org/10.55452/1998-6688-2023-20-4-48-54

[1]*IBRAGIM G.K., [2]UMAROV T.F.
[1]Kazakh-British Technical University,050000, Almaty, Kazakhstan
[2]British Management University, Tashkent, Uzbekistan
*E-mail: g.ibragim@kbtu.kz

THE CONSIDERATION OF SEMANTIC GAP BETWEEN DESIGN AND CODE

Abstract

In project management on creation of program application, specialists from different subject areas are involved, who include their contributions, for instance, UI/UX designers who create mock-ups of the future application or developers who write the code according to the prototype. The design conception may go beyond the possibilities of interpreting it from a technical point of view of implementation. The realization of such idea could not to be able to collect on only one defined program platform or language, and accordingly the problem is appeared. To eliminate semantic gap between the designers' concepts and opportunity of program developers in technical affordance, released methodology, Model Driven Architecture (MDA), which is, on the one hand, a concept for implementation of software, on the other hand a standard. In paper, considerate the MDA and its transformation levels with determine a pragmatical semantics of mapping, reasons of chosen a class diagram as model of transformation and Java language for code generation.

Key words: MDA, UML, transformation, mapping, semantic gap, pragmatics, class diagram, Java.

Introduction

Model Driven Architecture (MDA) is a methodology for the implementation processes of program applications. The MDA was realized in 2001 year by the Object Management Group (OMG). OMG is an international, open membership, not-for-profit technology standards consortium. Founded in the 1989 year. Figure 1 illustrates the inherent pragmatics of the relationship between Model Driven Engineering, Model Driven Development and Model Driven Architecture.

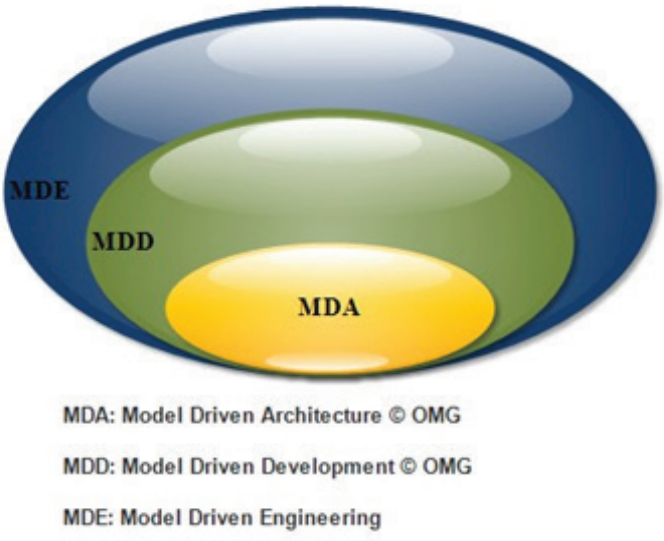

Figure 1 – The structure of MDE

MDA is a core of Model Driven Engineering (MDE). MDE is a software development methodology that basis on defining and utilizing domain models.

The syntax of a programming language defines the processes of microprocessor, so it defines semantics of functioning the microprocessor by executed program. Semantics in UML diagram defines abstract functioning of executable program. And these two semantics is distinct. So, semantic gap between UML diagram and the code representations of future application are emerged.

Semantics of programming language consist of domain semantics and semantic mapping. Semantic mapping relates the syntactic expression to the components of the semantic domain. Semantic domain is an ontology, description of concepts of construction of programming language, in this context. The notion semantic domain may use in describing the semantics of design part of development. UML diagrams have components which describe the processes of functioning of implementing application. Hence it follows that UML diagram components depict semantic domain.

A semantic gap is a discrepancy in the logical connection of elements of one model in another, transformed because of the first. The basic importance to Model Driven Architecture is a notion of metamodel, above that it obtains model transformations. Metamodels determined using the Meta Object Facility (MOF) standard. OMG defined also a specific standard language for models' transformation which called Query/Views/Transformations (QVT). And defined mechanism based on XML, which provide interchange between models.

QVT is not success concept, it has no complete implementation, no industrial support, and not used much by developers. For MDA implementation released Eclipse Metamodel Framework (EMF) tool. EMF is used for a research project and affords to support a metamodel.

Materials and methods

Transformation

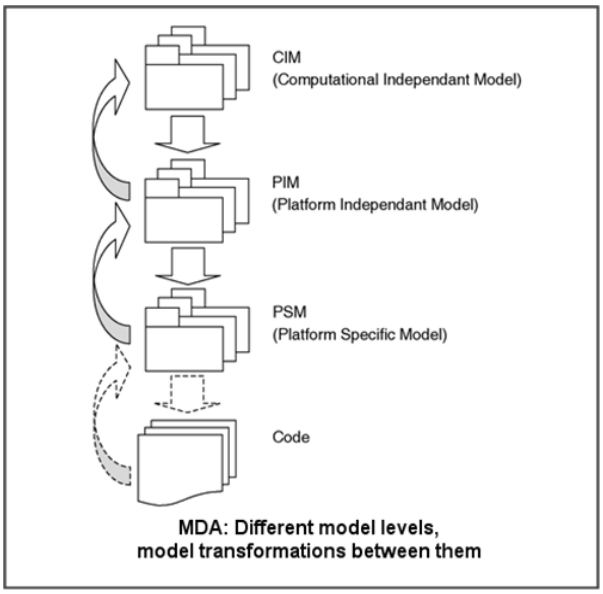

Figure 2 – Sequential transformations between models of MDA

Figure 2 explains the sequential transformations up Computational Independent Model (CIM) level to executed code. The CIM level consider specifications, scenario, and requirements to software application. It contains semantics of domain elements of conceptual model of application. Platform Independent Model (PIM) is transformed version of CIM level. On transformation between CIM and PIM levels the scenario of implementing application turn into UML diagram, which is understandable design not only for domain specialist. On transformation of PIM to Platform Specific Model (PSM) level, which demonstrated on Figure 3, the model of application turns into detailed version of PIM. On that level all UML diagram elements acquire more detailed features, which give opportunity to

transform it to code. Behind all that transformations hidden notion which identifies metamodels and contain domain semantics.

Main provisions

Mapping is not easy part of MDA. The semantics of domain specification should not change on transformation levels. Mapping feasible based on NLP and Graph theory.

Mapping based on matching. The notion of second suppose the use of Model Management Algebra (MMA). The match in MMA presented as an operator which takes two models as input and returns a mapping of them. Mapping identifies combinations of objects in the input models that are either equal or similar, based on the external meaning of equality and similarity. That definitions set leads to two versions of the operator: Elementary and Complex matches.

Elementary is when one element is a modified version of another. The complex is based on the complex meanings of the equation. That match, in its case, should distinguish sets of equal objects from similar ones. Similarity implies that the object is related, but with uncertainty, how are objects interconnected. Elementary and complex matching are not algorithms. But these matching's rules have their usage in graph isomorphism to detect structural similarity in complex models, and not only. In NLP to identify, analyze similarity in text of a model.

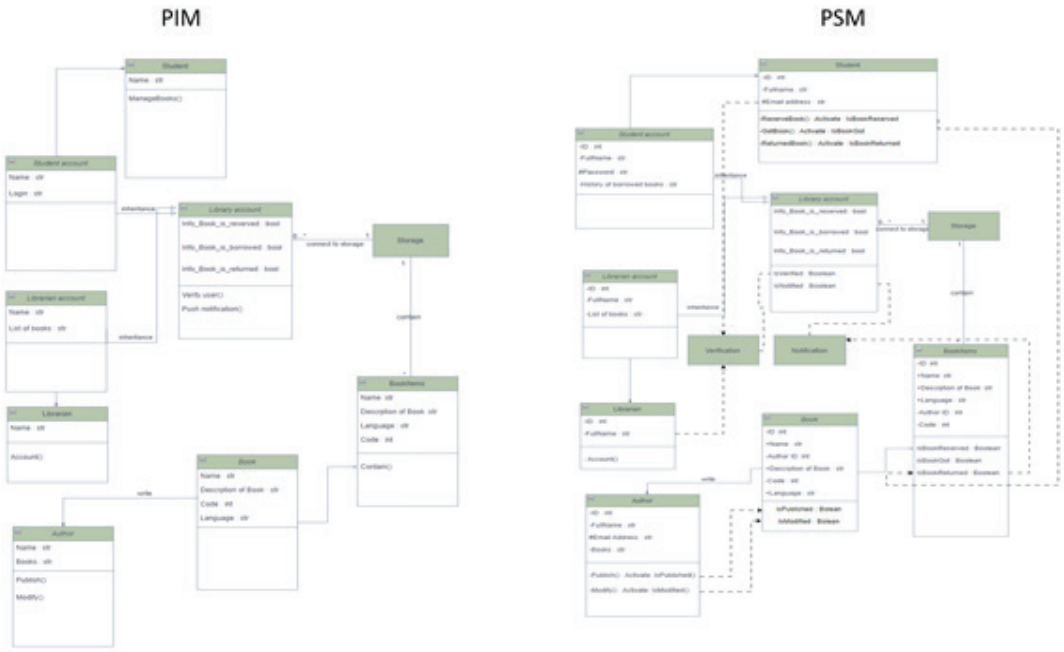

Figure 3 – Transformation PIM to PSM

Results and discussion

Pragmatics of mapping

In sequential of all written above may conclude following definition which contained in the notion of matching according to software development:

Semantic pragmatics is a comparison of two models by separating the essential from the non-essential. For instance, semantic pragmatics between model programming language and the model of PSM level, which in case the transformed model of previous levels of MDA, reveals in cutting off elements do not correspond to the syntax of programming language and domain-driven design specifications.

Semantic pragmatics is embedded in matching, which means it is embedded in mapping, which in case embedded in transformations of levels of MDA.

Literature review

MDA reviewed in papers [1, 2, 3, 4]. In articles considered MDA methodology in detailed position from statement to levels transformation description.

In the paper of Peter D. Mosses el at. [5] a semantics of programming language is determined as conceptual meaning of a program. It means that semantics provides abstract version of how the application will work in real. The form and structure of semantics of any program are determined by their syntax. So, the syntax has the defining role in collecting a semantics of implementing application.

In [6], considered the solution of bridging translating problems between pseudo-code and code with using NLTK library functions. NLP is a developing sphere of information technology. Nowadays, most applications based on trained "AI", the abbreviation AI in parenthesis, because it is not complete version of human brain, it works similar and with human written algorithms. It is not existing by itself. NLP use machine learning methods and related to data science, because from the namespace, it processes the text. Data is textual and symbolic information. NLP use in automatic word detection, words translator. Tokenization and summarization are the main parts of NLP. In paper, the primary objective in research was to translate the pseudo-code to code automatically. The method to solve was using seq2seq technique. The prevented technique solves the 26% blank pseudo-code problem of SPoC dataset.

In [7], authors present an approach to automatically transform textual business rules to an SBVR model, Semantics of Business Vocabulary and Business Rule is a standard of OMG. The approach sate on NLP and SBRV model, which include semantic notations of each rule. The semantics contained as XMI file.

In paper [8], presented approach of automatic generation of code using smart contract code examples from Solidity PSM. then the generated smart contract code compile on the Ethereum blockchain JavaScript virtual machine, compare with original contract code in terms of Solidity code metrics, similarity scores and execution costs. Authors elaborate on how the Solidity PSM is used for Solidity smart contract code generation by employing model-to-text transformations.

In [9], proposed transformation from PIM to PSM as a process. Authors extend it as separating mapping specification and transformation definition. The proposed process involves a metamodel based on MOF and Ecore, a UML metamodel, a mapping and transformation language model, and a transformation engine.

The mapping model specifies a relationship between the source and target metamodel, which is an UML.

A transformation model generates from a mapping model. The transformation program implements on the base of the transformation model. Transformation accomplishes according to the transformation engine, which executes the transformation program. Then the transformation engine on output produces the target model.

Three categories of mapping given in the article based on the concept of similar structure and semantics between the elements of metamodels. There are: one – to – one, one – to – many, many – to – one.

A one - to - one mapping is defined by one element from a target metamodel that equal to similar structure and semantics of one element from a source metamodel. A one – to – many mapping is defined by non-empty and non-unitary set of elements from a target metamodel with similar semantics to one element from a source metamodel. The last mapping is opposite definition of the one – to – many mapping.

Article [10] describe tool and approach of automatic generation code from UML class diagram in software development, consequently. Authors in their article describe the Eclipse modeling tool in concrete and Java code generation from UML diagram file. In [11] given approach of automatically generating Java code. Authors created GenCode named tool as solution for mobile application development. GenCode is open access and generate Java code from UML only. The algorithm of GenCode tool is as follows: First, the diagram is fixed and sorted into the "structure" and "sequence" packages. The structure package contains a class diagram, and sequence contains a sequence diagram. After that, the "models' generator" package will generate code for Android generator and CSharpgenerator for the selected one. First, the structural code is generated, then the behavioral code.

In article [13], the authors research focuses on identification of significance of class diagram in software development. And formulated the class diagram description.

In article [14], a metamodel of Java language and model-based code transformations are touched upon. There are presented the scratch of Java meta-based code as an example of applying modeling tools like QVT. Besides, marked the definition blackbox.

Class diagram and Java

Unified Model Language is easy implementable to any Independent Development Environment (IDE) for its paradigm. In software engineering there are some IDE to generate code from UML directly: Eclipse Metamodel Framework IDE, NetBeans IDE, IntelliJ IDEA, Visual studio, Android Studio.

IDE transform the PSM level to code not from the initial CIM level. There are many research in using that IDEs but one is not yet described. It is Android studio. All these IDE based on class diagram and implement transformation from PIM to PSM, then to code. To get such result it requires Visual Paradigm plugin. The reason of using Visual Paradigm by developers is that the Visual Paradigm is an aggregate of design, analysis, management tools, which provide code generation. The noticed detail in research of software development that most code transformations based on class diagram and interpretation on Java language. The question why class diagram chosen as optimal variant arises, and why code interpreted in Java not Python or C++. The class diagram relates to structural type, so it is static and used to model static view of a software application. The static view describes the vocabulary of the prototype of application. Beside that the class diagram is a consideration for component and deployment diagrams and used to build the executable code. UML diagrams not entirely based on OOP, but exactly the class diagram present the mapping of object-oriented languages. One of these object-oriented languages is Java. The choose exactly that language comes from its semantic modelling. Because of the class diagram is static, and static semantics are sufficient to construct the most used Java refactoring. Java metamodel reflects static semantics.

Conclusion

The implementation of any software application is based on semantics. Semantics in MDA is presented in the form of a CIM level specification, which interpreted as a UML diagram at the PIM level. Then, the PIM model converts to the PSM model. By comparing models, the semantic gap is eliminated. The mapping is based on semantic pragmatics. Each transformation is a set of rules that is determined by semantic pragmatics. The article also presents a literature review of articles by other researchers related to this topic. Many software developers use IDE to automatically generate code from UML diagrams. And each of them is associated with class diagrams and the Java language. The reason of chosen is their similar semantic structure. Due to that, the class diagram correctly converts into the Java code, and thereby the semantic gap between design and code is eliminated.

REFERENCES

1 Silega N., Noguera M., Rogozov Y. I., Lapshin V. S. & Gonzalez T. (2022) Transformation from CIM to PIM: A systematic mapping. IEEE Access, 10, 90857-90872.

2 Niepostyn S.J. (2015) Consistent model driven architecture. Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2015.

3 Natek H., Elmounadi A. & Guerouate, F. (2022). Overview in the eclipse model-driven architecture tools. ITM Web of Conferences, 46, 02001.

4 Safitri A. G. & Atqiya F. (2022). Automatic model transformation on multi-platform system development with model driven architecture approach. Computer Science and Information Technologies, 3(3), pp. 157–168.

5 Mosses P. D. (2006). Formal semantics of programming languages. Electronic Notes in Theoretical Computer Science, 148(1), pp. 41–73.

6 Acharjee U.K., Arefin M., Hossen K.M., Uddin M.N., Uddin M.A. & Islam L. (2022) Sequence-to-sequence learning-based conversion of pseudo-code to source code using neural translation approach. IEEE Access, 10, 26730-26742.

7 Haj A., Jarrar A., Balouki Y. & Gadir T. (2021) The semantic of business vocabulary and business rules: An automatic generation from textual statements. IEEE Access, 9, 56506-56522.

8 Jurgelaitis M., Ceponiene L. & Butkiene R. (2022) Solidity code generation from UML state machines in model-driven smart contract development. IEEE Access, 10, 33465-33481.

9 Lopes D., Hammoudi S., Bézivin J. & Jouault, F. (2006) Mapping specification in MDA: From theory to practice. Interoperability of Enterprise Software and Applications, pp. 253–264.

10 Fouquet F., Nain G., Morin, B., Daubert E., Barais, O., Plouzeau N., & Jézéquel J. (2012) An eclipse modelling framework alternative to meet the Models@Runtime requirements. Model Driven Engineering Languages and Systems, pp. 87–101.

11 Parada A., Marques M. & Brisolara L.B. (2015) Automating mobile application development: UML-based code generation for Android and Windows phone. Revista de Informática Teórica e Aplicada, 22(2), 31.

12 Yang S. & Sahraoui H. (2022) Towards automatically extracting UML class diagrams from natural language specifications. Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings.

13 Yashwant W. (2014) Significance of class diagram in software development. Conference Managelization at Osmanabad.

14 Hamioud S. & Atil F. (2015) Model-driven Java code refactoring. Computer Science and Information Systems, 12(2), pp. 375–403.

[1]*ИБРАГИМ Г.К., [2]УМАРОВ Т.Ф.
[1]Қазақстан-Британ техникалық университеті, 050000, Алматы қ., Қазақстан
[2]Британ менеджмент университеті, Ташкент қ., Өзбекстан
*E-mail: g.ibragim@kbtu.kz

ДИЗАЙН МЕН КОД АРАСЫНДАҒЫ СЕМАНТИКАЛЫҚ АЛШАҚТЫҚТЫ ҚАРАСТЫРУ

Аңдатпа

Жобаларды басқару кезінде, бағдарламалық қосымшаны құруға әртүрлі салалардың мамандары қатысады және әр маман жобаға өз үлесін қосады. Мысалы, болашақ қосымшаның макеттерін жасаушы – UI/UX дизайнерлер мен сол прототипке сәйкес код жазатын бағдарламалаушылар. Дизайн тұжырымдамасы іске асырудың техникалық тұрғысынан, жүзеге асырылу мүмкіндіктерінен тыс болуы мүмкін. Жүзеге асыруды тек белгілі бір бағдарламалық платформа немесе бағдарламалау тілінде жинау мүмкін емес, сәйкесінше дамытуда да қиындықтар туындайды. Дизайнерлердің тұжырымдамалары мен бағдарламалық жасақтама жасаушылардың техникалық қол жетімділік мүмкіндіктері арасындағы семантикалық алшақтықты жою үшін Model driven Architecture (MDA) әдістемесі шығарылды, бұл бір жағынан бағдарламалық жасақтаманы енгізу тұжырымдамасы, екінші жағынан стандарт. Мақалада MDA және оның түрлендіру деңгейлерінің арасындағы прагматикалық семантика, түрлендіру моделі ретінде класс диаграммасы мен кодты құру үшін Java тілін таңдау себептері зерттелді.

Тірек сөздер: MDA, UML, трансформация, сәйкестендіру, семантикалық алшақтық, прагматика, класс диаграммасы, Java.

[1]*ИБРАГИМ Г.К., [2]УМАРОВ Т.Ф.
[1]Казахстанко-Британский технический университет, 050000, г. Алматы, Казахстан
[2]Британский университет менеджмента, г.Ташкент, Узбекистан
*E-mail: g.ibragim@kbtu.kz

РАССМОТРЕНИЕ СЕМАНТИЧЕСКОГО РАЗРЫВА МЕЖДУ ДИЗАЙНОМ И КОДОМ

Аннотация

В управление проектами при создании программного приложения вовлечены специалисты разных предметных областей, которые делают свой вклад. Например, дизайнеры UI/UX, которые создают макеты будущего приложения, или разработчики, которые пишут код в соответствии с прототипом. Концепция дизайна может выходить за рамки возможностей ее интерпретации с технической точки зрения реализации. Реализа-

цию невозможно собрать только на одной определенной программной платформе или языке, и, соответственно, появляются проблемы в разработке. Для устранения семантического разрыва между концепциями дизайнеров и возможностями разработчиков программ в технической доступности была выпущена методология Model Driven Architecture (MDA), которая является, с одной стороны, концепцией внедрения программного обеспечения, с другой – стандартом. В статье рассматривается MDA и его уровни преобразования с определением прагматической семантики отображения, причин выбора диаграммы классов в качестве модели преобразования и языка Java для генерации кода.

Ключевые слова: MDA, UML, трансформация, сопоставление, семантический разрыв, прагматика, диаграмма классов, Java.

About authors

Ibrahim Gulnur Kuandykizi
Master's degree, Kazakh-British Technical University, 59, Tole bi str.,
050000, Almaty, Kazakhstan.
ORCID ID: 0000-0003-0974-106X
E-mail: g.ibragim@kbtu.kz
Timur Faridovich Umarov
Professor, British Management University, Tashkent, Uzbekistan.
ORCID ID: 0009-0008-0044-7159
E-mail: t.umarov@bmu-edu.uz

Авторлар туралы мәліметтер

Ибрагим Гүлнұр Қуандыққызы
Магистр, Қазақстан-Британ техникалық университеті, Төле би көш., 59,
050000, Алматы қ., Қазақстан
ORCID ID: 0000-0003-0974-106X
E-mail: g.ibragim@kbtu.kz
Умаров Тимур Фаридович
Профессор, British Management University, Ташкент қ., Өзбекстан
ORCID ID: 0009-0008-0044-7159
E-mail: t.umarov@bmu-edu.uz

Информация об авторах

Ибрагим Гүлнұр Қуандыққызы
Магистр, Казахстанско-Британский технический университет, ул. Толе би, 59,
050000, г. Алматы, Казахстан
ORCID ID: 0000-0003-0974-106X
E-mail: g.ibragim@kbtu.kz
Умаров Тимур Фаридович
Профессор, British Management University, г. Ташкент, Узбекистан
ORCID ID: 0009-0008-0044-7159
E-mail: t.umarov@bmu-edu.uz