Transformation of Class Hierarchies During Software Development in UML

Himesha Wijekoon, Vojtěch Merunka
Department of Information Engineering
Faculty of Economics and Management
Czech University of Life Sciences Prague
Prague, Czechia
email: wijekoon@pef.czu.cz, merunka@pef.czu.cz

Vojtěch Merunka
Department of Software Engineering
Faculty of Nuclear Sciences and Engineering
Czech Technical University in Prague
Prague, Czechia
email: vojtech.merunka@fjfi.cvut.cz

***Abstract*—This article discusses support for the Unified Modelling Language (UML) standard in Model Driven Architecture (MDA) style software development. There are described some of the weaknesses of the UML standard that software developers should know about, to take full advantage of this otherwise very good and desirable standard. Specifically, it is a hierarchy of object classes, which belongs to the basic concepts of the object-oriented paradigm. This hierarchy is considered well known, but in fact there are three slightly different hierarchies that fortunately fit well with the MDA philosophy. The problem is mainly that all these three hierarchies appear in UML in the same way, as if they were just one type of hierarchy. The article describes and explains these differences and suggests a refinement to the UML using stereotypes. The conclusions written in this article are a summary of the authors' experience of software projects for the international consulting company Deloitte and of university education.**

Keywords-UML; software development life cycle; transformation of concepts; MDA; class hierarchies.

I. INTRODUCTION

The objective of Unified Modelling Language (UML) has been and is to replace older methodologies by one methodology that is a combination of the best of the older ones. Likewise, the Model Driven Architecture (MDA) philosophy is a synthesis of previous best experiences in the creation of large-scale software, where there is a semantic gap between programmers and people in the area of the modeled problem.

The history of software engineering could be simply described as a human struggle with complexity. The solution is to split a complex task into a set of many smaller and therefore simpler tasks that one can already handle. Incidentally, this idea, which is the basis, for example, of the programming of computers is not new. It was probably first pronounced by a Persian scientist Muhammad ibn Musa al-Khwarizmi in his book "The Compendious Book on Calculation by Completion and Balancing" which became the basis of modern mathematics and was the forerunner of software engineering [1][2].

Authors, based on their practical experience in an international consulting company, have experienced that the same UML diagram is understood differently by different development team members (e.g., problem domain experts, IT architects, data analysts, programmers). This increases the semantic gap between users and developers and makes software development more complicated, expensive, and error prone.

This paper discusses about using UML standards in the MDA approach for software development. More precisely, the paper discusses how different types of hierarchies can be expressed in UML class diagrams.

This paper is organized as follows:

- The Introduction is followed by Section II on UML and its problems.
- This is followed by Section III on the MDA approach.
- Section IV is central because it contains our own research, which is described in a concrete example.
- Section V is a discussion and suggested solution.
- The Conclusion of this article.

II. OBJECT-ORIENTED APPROACH AND THE ORIGIN OF UML

Before the arrival of UML, in early 1990s, we had several competing object-oriented methodologies with mutually different notations. These were so called first generation object-oriented methodologies. Many software companies used a combination of several methodologies instead of just one methodology – mostly object models from Object Modelling Technique (OMT) along with interaction diagrams from the Booch method and the Use-Case approach of the Jacobson Object-Oriented Software Engineering (OOSE) method [3][4][5]. Most of these methodologies have later become the foundation for UML [6]. UML has brought along a unification of the previous notations. The UML notation is mostly based on OMT and has become a recognized standard. UML includes many different elements from the original methodologies. There is, for example, the so called "business extension" from the original Jacobson method that has been added in version 1.x, or the absorption of the Specification and Description Language (SDL) methodology for supporting real-time processes in version 2.x [7].

Obviously, the UML is not a method. UML is "only" a modelling language [8]. That, itself should not be a problem as it is good that since 1996, we have had a standard for object modeling. The problem, however, is the fact that for the "universal" language there are more methodologies (e.g., Rational Unified Process) and even mere knowledge of UML is often considered a methodology [9].

 ISBN: 978-1-61208-981-2

A. Is UML a Method?

Experience proves that it is not a method. UML is definitely not a method that could be understood by a layman in reasonable time (for instance in 15 minutes at the beginning of a meeting with analysts), to be able to read and understand the diagrams. This is not an unrealistic requirement, because in the past it was possible to work like this with entity-relational and data-flow models. Unfortunately, in object-oriented modeling we do not have such an elegant and simple method. Instead, we send customers to attend long training sessions on UML, where we make them work with Computer Aided Software Engineering (CASE) tools.

B. Some Issues of UML

Most criticism at UML is directed at its complexity and inconsistency. It is, for example, the direction of the arrows of different links that sometimes draws in reverse with reality. Another criticism is the varying level of detail. For example, terms directly related to C++ or Java and similar programming languages have beautiful distinguishable symbols, but concepts are also very important but not supported in Java-like programming languages have very little support or only optional textual stereotype. The third and last part of the criticism speaks of complicated or even no UML support for the decomposition and generalization of diagrams that no longer have the elegance of the old Data Flow Diagram (DFD). A good publication on this topic is an article by Simons and Graham [10].

However, we know many of these things also from other areas of science. As a typical example, let's look at the direction of the flow of the electric current that is drawn from the positive pole to the negative pole in electric circuit diagrams since Michael Faraday's time, which is the opposite of reality, as every bright student knows today.

Individuals who are not familiar with programming find UML too difficult, and then they incorrectly interpret the entire object-oriented approach [10][11][12]. It is possible to pick an acceptable set of concepts out of UML for non-programmers; nevertheless, most professional books and training sessions are too often unnecessarily based on programmer experience. Comprehensibility and simplicity of UML is corrupted by the following facts:

1. UML models contain too many concepts. The concepts are at different levels of abstraction, and sometimes they semantically overlap (e.g., relations between use-cases); and even their concepts sometimes differ. The same model can therefore be interpreted differently by an analyst and a programmer (the typical example is associations between objects).
2. There are several ways in the UML diagrams to show certain details in models (e.g., qualifiers and link class objects or state diagrams that are a mix of Mealy and Moore automata). It is up to analysts, which option they choose.
3. Some concepts are insufficiently defined such as events in state diagrams. One UML symbol covers several different concepts (e.g., in sequence diagram the data flow between objects blends with control flow).
4. Although UML is generally good from the graphics aspect, some analysts do not like for example the same symbol of a rectangle for instance and class (they are differentiated only by internal description), as well as the direction of the inheritance arrow that leads toward the parent object in spite of the fact that in the codes of programming languages (even in users interpretations) inheritance is represented by opposite direction (i.e., from the parent object toward the descendant).

C. UML Support of Object-Oriented Approach

Although UML has the ambition to be truly versatile and is also registered as a universal International Standards Organization (ISO) standard [6], it is true that the largest field of application is object-oriented analysis and programming. UML supports many object-oriented concepts, and there is currently no other “more” object-oriented as well as standard modelling language. The success of UML in practical usage is based on many successful projects where the software has been developed in C++ or Java (i.e., languages that use object-oriented approach).

Practically speaking, UML is associated with object-oriented software creation for many users, who do not even know that UML has an overlap with other areas of software engineering, such as relational database modelling.

III. MDA Approach

MDA is an Object Management Group (OMG) specification based on fixed standards of this group [13]. The main idea behind MDA is to separate business and application system from the technology platform. This idea is not new as the need to create a separate analytical and design model has existed for quite some time. What MDA brings are procedures and ways to transform these models. The primary objectives of this approach are to ensure portability, interoperability, and reusability through a separate architecture [14].

The MDA approach advises a complex system to evolve as a gradual transformation of three large models:

1. Computer-Independent Model (CIM): This model, also known as the domain model, focuses exclusively on the environment and general requirements of the system. Its detailed structure and specific computer solution are hidden or unspecified at this stage. This model reflects customer's business requirements and helps to accurately describe what is expected of the system. Therefore, they must be independent of technical processing and describe the system in a purely factual and logical way. It does not require to know any details of computer programming, but rather requires knowledge of the real target environment.
2. Platform Independent Model (PIM): This model deals with the part of the complete system specification which does not change according to the particular type of computer platform chosen. In fact, PIM mediates a certain degree of independence of a particular solution to a given problem area to suit different platforms of a similar type. It describes the behaviour (algorithms) and structure of the application only within those limits that

 ISBN: 978-1-61208-981-2

will ensure its portability between different technological solutions. Compared to the CIM model, it is supplemented with information (algorithms, principles, rules, constraints, etc.) that are essential for solving the problem area through information technology. The big advantage of the PIM model is its reusability and therefore it can serve as a starting point for various assignments when it is necessary (e.g., to change to another programming language, the need to reuse some legacy component or data, etc.). It's like abstract programming in an ideal programming environment. At this stage of development, the so-called expansion of ideas is also taking place, as the target environment has not yet restricted us.

3. Platform-Specific Model (PSM): The latest MDA model, which is already platform dependent, combines PIM with a specific technology solution. There is a so-called consolidation where the previous ideas must be realized in a specific target computer environment with all the shortcomings and limitations of the version and configuration of the technology used.

IV. THREE DIFFERENT TYPES OF CLASS HIERARCHIES IN THE PROCESS OF SOFTWARE DEVELOPMENT

Conceptual hierarchy of classes, hierarchy of data types, and hierarchy of inheritance do not necessarily mean the same thing regardless all three hierarchies are drawn in the same way in UML. We can only use UML stereotypes to distinguish among them in detail. These hierarchies have a strong connection with MDA ideas and can be recognized as follows:

1. From the perspective of the user/analyst: The instances of lower-level classes then must be elements of the same domain that also includes the instances of the classes of the superior class. It means that a lower-level domain is a sub-set of a higher-level domain. This hierarchy is also called the IS-A hierarchy or also taxonomy of classes. In specific cases, it can differ from the hierarchy of types because it does not deal with the behaviour of the objects at the interface; rather it deals with the object instances as a whole including their internal data structure. Formally, we can define this hierarchy of a superclass A and subclass B as

$$A \prec B \;=\; \text{extent}(A) \supset \text{extent}(B) \qquad (1)$$

This hierarchy corresponds to the CIM phase of MDA.

2. From the perspective of polymorphism: This is a view of an application programmer who needs to know how to use the objects in the system but does not program them. The object in lower levels of hierarchy then must be capable of receiving the same messages and serve in the same or similar context, such as high-level objects. Therefore, this hierarchy is the hierarchy of types. Formally, we can define this hierarchy of a superclass A and subclass B as

$$A \prec B \;=\; \text{interface}(A) \subseteq \text{interface}(B) \qquad (2)$$

This hierarchy corresponds to the PIM phase of MDA.

3. From the designer's perspective: new object designer. This is a view of a system programmer who needs to create these objects. This hierarchy is a hierarchy of inheritance because inheritance is a typical tool for the development of new classes. Formally, we can define this hierarchy of a superclass A and subclass B as

$$A \prec B \;=\; \text{methods}(A) \subseteq \text{methods}(B) \qquad (3)$$

This hierarchy corresponds to the PSM phase of MDA.

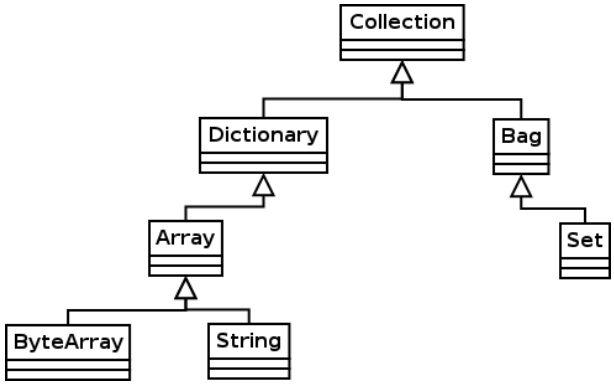

Figure 1. IS-A Hierarchy (Class Taxonomy)

In simple problems, it is obviously true that these three above-mentioned hierarchies are identical. However, this is not true in more complex problems (e.g., in the design of system libraries that are often re-used when developing specific systems).

A. *An Example - Library of Object Collections*

Figure 1 is a good example showing IS-A hierarchy, hierarchy of types and hierarchy of inheritance which is a part of a system library of the Smalltalk language concerning collections of objects. A similar library can be found in each object-oriented programming language, of course. There are the following classes:

- Collection: This is an abstract class from which the individual specific classes are derived. A common quality of all these objects is the ability to contain other objects as their own data.
- Dictionary: This is a collection where each value stored has a different value assigned to it (therefore forming a pair), which serves as an access key to the specific value. Dictionaries can be really used as dictionaries for simple translations from one language to another. Another frequently used example of the use of object dictionaries is a telephone book (i.e., the key is the names of the people and the values connected with the keys are the telephone numbers).
- Array: Simply said, an array is a dictionary where the keys can only be natural numbers from 1 to the size of

 ISBN: 978-1-61208-981-2

the array. So, the array values are also accessed as if through keys.

- Byte Array: It is an array where the permitted scope of values is limited to whole numbers in the interval from 0 to 255.
- String: A string of characters can be also viewed as an array where the permitted scope of values is limited to characters.
- Bag: This is a collection in which internal objects can be stored inside without any accessing key.
- Set: This is a special type of a bag where, in addition, the same value can occur only once. If the set already contains a specific value, another input of the same value is ignored unlike the above-mentioned bag, which allows multiple occurrences of the same value. The objects which are elements of the set are functionally corresponding with mathematical concept of sets. Therefore, they have this name.

This description of the classes from Figure 1 follows the IS-A hierarchy (or class taxonomy) as we know it from natural sciences. But we may define a slightly different perspective as it is presented at Figure 2, but equally important as first one from Figure 1. If we concentrate on behaviour of objects, we obtain a bit different hierarchy that is defined by the scope of permissible messages. Or we can also declare this hierarchy as a hierarchy of object interfaces. It is the hierarchy of types corresponding with the PIM phase of MDA. This supertype-subtype hierarchy has following differences from previous IS-A hierarchy:

- Because Dictionaries can receive the same messages as Sets, they can be therefore viewed as sub-types of Sets. The same applies also for Bags.
- Arrays and String are interpreted as almost independent classes because each of them supports very specific operations (messages) with very little common intersection.

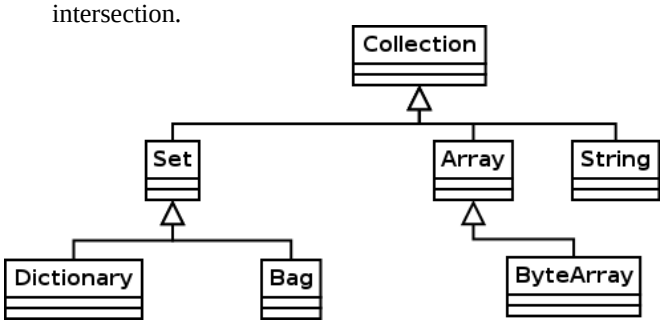

Figure 2. Hierarchy of Types (Supertype - Subtype Hierarchy)

This second hierarchy is not the last one. We can create yet one more hierarchy to match the PSM phase of MDA. See Figure 3. This hierarchy of inheritance is very important for the programming when programmers have to create their objects in some programming languages. Again, we will have some differences from previous hierarchies:

- Strings can be implemented as a special kind of ByteArrays (e.g., inherited subclass), because separate character elements are typically encoded into bytes of tuples of bytes.

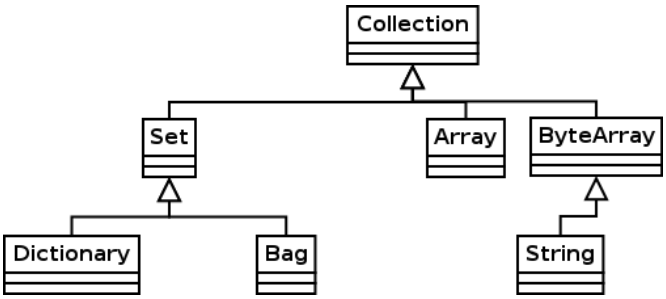

Figure 3. Hierarchy of Inheritance

- Implementation of Arrays and ByteArrays has nothing in common and therefore it makes no sense to inherit anything together. Arrays are implemented using pointers which point to the internal objects that make their elements, but ByteArrays are contiguous sections of computer memory, where their elements are stored directly in these bytes. Although these two classes have much in common and can receive the same messages in terms of external behaviour (that is, they have polymorphism), the code of their methods cannot be shared and it is necessary to program each method separately, although they seem very similar.

V. DISCUSSION - UML SUPPORT FOR SOFTWARE DEVELOPMENT PHASES

In Section IV, we have just explained the need for different class hierarchies. The problem remains to be resolved is, how to express them in the UML class diagrams. Fortunately, the UML standard includes an extension mechanism that allows new concepts to be introduced in a standard way. They are so-called stereotypes. All we have to do is select some graphic element, and we can give it a different interpretation by typing the text in double angle brackets « ». The result is in Figure 4.

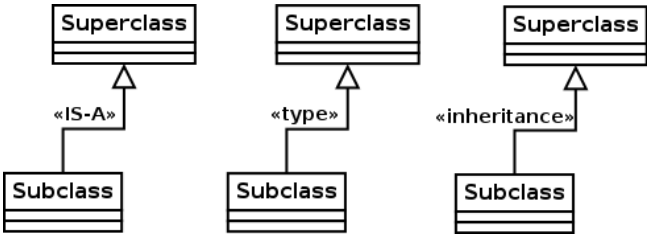

Figure 4. UML Extension Proposal

Of course, if each UML class diagram clearly indicates what phase of the model it is (CIM, PIM, or PSM in the style of MDA, for example), then this stereotype is unnecessary.

A. The Need of MDA Way of Thinking

During system development it is necessary to gradually transform the system model into a condition that is necessary for physical implementation of the system in program form in the specific programming language.

According to our experience, initial objects cannot be viewed only as initial simplification of the same future software objects, as the common error of the analysts in

 ISBN: 978-1-61208-981-2

UML [15]. The initial business model is simpler, but at the same time it contains concepts which are not directly supported by current programming languages.

In the work on major projects, IS analysts face problems when not all system requirements are known at the start of the project and the customer expects that discovery and refinement thereof will be part of the project. These problems are even more complicated because the function of the major systems built has impact on the very organizational and management structure of a company or organization where the system is implemented (e.g., new, or modified job positions, management changes, new positions, new departments, etc.). Therefore, it is desirable to also address the change of these related structures during the work on information systems.

VI. CONCLUSION

In this paper, we demonstrated the need of precise interpretation of modelling concepts on an example of gradually transforming object class hierarchy. This approach is a practical realization of MDA ideas in UML.

Underestimation of the model differences in the individual phases of development of an information system happens, when the analysis using UML is viewed by programmers as the sole graphical representation of the future software code. Analytical models are then used not to specify the problem formulation with the potential users of the system who are also stressed by the complexity of the models that are presented to them. In our practical experience, many projects in UML suffer from this problem. In response to that, there are two "remedial" approaches used in practice: Extreme Programming [16] and Domain Specific Methodologies [15]. But it is as if also the baby itself had been spilled with dirty water from the bath.

The objective of this article is not to suggest that UML is a bad tool. On the contrary, UML is a good and rich tool. The fact that it is not perfect in all areas is not anything horrible. UML is the first successful attempt to introduce a reasonable object-oriented standard, and it is good to use it. We only wanted to point out some of the problems that relate to the use of the UML. We see a danger that results in the fact that the UML is taught and used incorrectly. The problems discussed can be summarized as follows:

1. UML is not a method. It is "only" a standardized tool for recording. UML needs some method, otherwise it doesn't help.
2. UML is complicated. People who are not familiar with programming have difficulty learning it. It is not easy to explain UML to laymen and non-programmers in just a few minutes at the first meeting.
3. Analysis in UML must not be a graphical representation of the future program code.
4. UML itself does not accurately emphasize which concepts are to be used in the analysis phase and which concepts to be used in the design and implementation phase. Unfortunately, many books on UML look at modelling through the eyes of implementation and are written in a language for programmers and particularly programmers in C++ or Java or a similar programming language.

The thoughts described in this article are a synthesis of our own experiences from object-oriented modelling at the international consulting company Deloitte, from own research activities and from teaching the development of information systems at the universities.

REFERENCES

[1] H. Zemanek, "Al-Khorezmi His background, his personality his work and his influence," Ershov A.P., Knuth D.E. (eds) Algorithms in Modern Mathematics and Computer Science. Lecture Notes in Computer Science, vol. 122. Springer, Berlin, Heidelberg, 1981.

[2] D.E. Knuth, "Algorithms in modern mathematics and computer science," Ershov A.P., Knuth D.E. (eds) Algorithms in Modern Mathematics and Computer Science. Lecture Notes in Computer Science, vol. 122. Springer, Berlin, Heidelberg, 1981.

[3] J. Rumbaugh, "Object-Oriented Modeling and Design," Prentice-Hall International, 1991.

[4] G. Booch, "Object-Oriented Analysis and Design with Applications (2nd ed.). Redwood City: Benjamin Cummings. ISBN 0-8053-5340-2, 1993.

[5] I. Jacobson, M. Christerson, and G. Övergaard, "Object Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley, 1992.

[6] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML) Version 2.5.1*. [Online]. Available from: https://www.omg.org/spec/UML/2.5.1/PDF, 2022.05.17.

[7] R. Reed, "Notes on SDL-2000 for the new millennium. Computer Networks," 35. No. 6, pp. 709-720, 2001.

[8] J. Hunt, "The Unified Process for Practitioners: Object-oriented Design, UML and Java," vol. 12, Springer Science & Business Media, 2000.

[9] Object Management Group (OMG). *Introduction To OMG's Unified Modeling Language (UML)*. [Online]. Available from: https://www.uml.org/what-is-uml.htm, 2022.05.17.

[10] A.J. Simons and I. Graham, "30 Things that Go Wrong in Object Modelling with UML 1.3," Kilov, H., Rumpe, B., Simmonds, I. (eds) Behavioral Specifications of Businesses and Systems, The Springer International Series in Engineering and Computer Science, vol. 523, pp. 237-257, Springer, Boston, MA. 1999.

[11] S.W.Ambler. *Toward executable UML*. [Online]. Available from: https://drdobbs.com/toward-executable-uml/184414808?queryText=scott%2Bambler%2BUML%2Bexecutable, 2022.05.17.

[12] D. Thomas, "UML - Unified or Universal Modeling Language?", Journal of Object Technology, vol. 2, no. 1, pp. 7-12, 2003.

[13] Object Management Group (OMG). *MDA - The Architecture of Choice for a Changing World*. [Online]. Available from: https://www.omg.org/mda/, 2022.05.17.

[14] A. Noureen, A. Amjad, and F. Azam, "Model Driven Architecture - Issues, Challenges and Future Directions," Journal of Software. vol. 11, no. 9, pp. 924-933, 2016.

[15] S. Kelly and J.P. Tolvanen, "Domain-Specific Modeling. Enabling Full Code Generation," John Wiley & Sons, 2008.