Rethinking Model Driven Development: Analysis and Opportunities

Nikola Tanković\textsuperscript{1}, Dražen Vukotić\textsuperscript{1}, and Mario Žagar\textsuperscript{2}
\textsuperscript{1}Superius d.o.o., Čirilometodske družbe 1, 52100 Pula, Croatia
\textsuperscript{2}Faculty of Electrical Engineering & Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia
nikola.tankovic@superius.hr, drazen.vukotic@superius.hr, mario.zagar@fer.hr

Abstract. Application modeling is gaining its share as the next generation software development methodology. Model Driven Development (MDD) uses abstract models of software systems to yield concrete implementations. This process can be achieved in two fundamentally different ways: by generating end artifacts, or by applying model interpretation.

Regardless of approach used, MDD still needs to surpass certain challenges to gain further appreciation among community. This paper classifies those challenges and proposes use of different methodologies which could enable simpler modeling solutions. Object Process Methodology (OPM) as software modeling tool is an adequate candidate yielding results in that direction.

Keywords. Generative MDD, Interpretive MDD, Object Process Methodology

1. Introduction

Software modeling is working its way as next evolutional step in software development by constantly coming with new ways to represent complex software aspects with new modeling languages and methodologies. In order to achieve those aspects, modeling languages, whether graphical or textual, together with their execution architecture must surpass all technical difficulties of targeted environments, represent and model complex software and maintain high understandability.

The primary goal of software modeling should be creation of meta-models and models that are closer to end-users and domain problems rather than to programmers and programming language specifics. Unfortunately, due to high level of complexity and technical specialties that both software and hardware systems possess, the translation between such higher level models and classical programming code is quite difficult to achieve. That is why software modeling is more oriented at surpassing those technical challenges rather than providing more intuitive ways of modeling.

Building an effective modeling solution means to choose an adequate level of abstraction to achieve simplicity, but also keeping expressivity within desirable levels. Too much abstraction can produce solutions with inadequate flexibility. On other hand, less abstract solutions can be very hard to understand and manage, but quite powerful and flexible. The key is in choosing right characteristics for each modeling task. There are two fundamentally different ways to achieve applications from models. First approach advocate transformation from high-level models to lower level artifacts like programming code, where second approach implies implementation of an execution engines capable of model interpretation at run-time. Former solutions are referred as Generative MDD, and latter as Interpretive MDD.

Figure 1 shows the main difference between running environments within interpretive and generative MDD. In Interpretive MDD case, modeling and runtime environments are the same, unlike in generative case where all three environments differ. Because of the same modeling and run-time environment, interpretive models are most accurate in producing end results, where in generative approach end results...
can differ. Both approaches will be described in following two sections together with their key representatives, positive and negative sides. Fourth section will describe a holistic approach to modeling called Object Process Methodology, and fifth section will bring the challenges that modeling solutions are facing together with suggestions that could possibly yield positive progress in tackling these challenges.

2. Generative MDD

Generative MDD endorses the idea that models should be used only to describe domain problems, and that the results should be achieved with generated programming code, which can be, if necessary, further manually tailored. Modeling environment is completely separated from execution environment. These solutions tend to generate applications by using many different models as input, and some of these models are also used to specify transformations between models and generated end-result. That is why generated solutions are considered to have additional degree of freedom, because not only that they provide modeling capabilities, but also fine-tuning of the model translation towards end result.

MDA initiative\(^1\), launched by the OMG\(^2\) group in 2001, is de facto a standard in this approach. MDA defines functionality using appropriate domain-specific language (DSL) used to compose a platform-independent model (PIM). PIM is platform agnostic and concentrates only at describing domain problem and is further transformed into many platform-specific models (PSM) which are tailored for specific platforms and technologies on which the final solution will run onto. In MDA, models are the main artifacts or “ingredients” integrated into the development process through the chain of transformations starting from PIM through many PSM’s to final generated programming code and artifacts. MDA is the most popular representative of the Generative MDD principle, it targets complex systems and it is flexible to adapt to many technologies, which comes with a fee: having high level of complexity.

Generative approaches often lead to faster execution because they translate abstract models to optimal target language candidate specific for the task at hand. E.g. BPEL\(^3\) can be chosen for process execution or SQL for query execution. In contrast, interpreted approaches tend to use their own data storage space which is rather universal, non-optimized. This can be compensated by using modern solutions like NoSQL\(^4\) in which a schema can be adapted dynamically in run-time, or can even not be needed at all \([2]\).

Generative MDD is more appropriate for deployment which requires protection of intellectual property, because once a model and transformations are defined to produce end-results, these results can be delivered without the ability of modeling solution being exposed. In interpretive MDD, final deployment also includes installation of executing engines with models for interpretation. This makes them exposed so they must seek another way of intellectual protection.

The downside of generative approach is the difficulty to extract model back from generated code to continue developing through model. Only solutions whose models are expressive enough with code generation fast enough do not require manual corrections of the programming code. These solutions are sometimes referred with term complete code generation tools\(^5\). With complete code generation, there is rarely need to examine or modify the generated program, just like there is rarely a need to examine or modify compiled machine code from programming languages.

3. Interpretive MDD

Using MDA terms, Interpretive MDD can be observed as a set of PIM models which are directly interpreted instead of translated further to PSM’s and source code. There is no additional executing language; the model itself is the execution language. A well-known representative of this MDD group is Executable UML \([3]\). In general, a model compiler can

---

\(^1\) Model Driven Architecture, available at \(\text{http://www.omg.org/mda}\)
\(^2\) OMG is an open-membership computer industry specifications consortium, founded in 1989 by 11 companies including Hewlett-Packard, IBM, Sun Microsystems and Apple Computer, \(\text{http://www.omg.com}\)
\(^3\) Business Process Execution Language is a standard executable language for specifying actions within business processes with web services. Available at \(\text{http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wbpel}\)
\(^4\) NoSQL stands for database management system solutions that differ from classic relational database systems. These data stores may not require fixed table schemas, they usually avoid costly join operations, and typically scale horizontally. They are also known as structured storage.
\(^5\) An example of such system is Evolve, available at \(\text{http://www.intrinsarc.com/evolve}\), and Eclipse EMF, available at \(\text{http://www.eclipse.org/modeling/emf/}\)
compile several executable UML models, each capturing a single cross-cutting concern to yield the running system. In this sense, executable UML makes use of the concepts in aspect-oriented programming by having specific UML models each describing certain aspect [4]. Some approaches also tend to provide a single model describing whole application like Open Process Graph [5]. OPG is a general purpose executable graph model that incorporates every aspect of an application, including process, user interface, and database in a single graph structure. Interpretive MDD principles are being met because there is no code generated since the graph itself is treated as the code. Instead, a server-side engine interprets the graph alongside user executing the application online.

The key advantage of Interpretive MDD is the ability for changes to be immediately effective and visible. Since executing engine interprets model one part at a time, system can be instantly adapted. Interpretive approach is often criticized for being more platform specific [1], because it is considered easier to implement additional generators in Generative MDD then whole executing engines for specific platform, but using platform independent technologies like Java or JavaScript can leverage this problem.

The end verdict between generative and interpretive solutions relies on specific needs. Modern solutions tend towards hybrid approaches [6] with generation and interpretation techniques integrated and combined. This can be achieved by either improving the interpreters with partial code generation or by improving the generators in executing speed so that end-artifacts can be held in main memory and generated on request.

4. OPM Methodology

This section introduces a different kind of modeling methodology called Object-Process Methodology (OPM) [7] together with a simple assessment of different positive aspects which are introduced with OPM and their possible application in field of software modeling. OPM has been successfully applied to modeling generic web applications [8], as well as to modeling data-driven web and mobile applications [2]. It is primarily used as conceptual design tool for complex systems in large industries such as space, electric supplier, military and telecommunication industry\(^6\).

OPM is a universal modeling approach which identifies certain aspect [4]. Some approaches also tend to provide a single model describing whole application like Open Process Graph [5]. OPG is a general purpose executable graph model that incorporates every aspect of an application, including process, user interface, and database in a single graph structure. Interpretive MDD principles are being met because there is no code generated since the graph itself is treated as the code. Instead, a server-side engine interprets the graph alongside user executing the application online.

The key advantage of Interpretive MDD is the ability for changes to be immediately effective and visible. Since executing engine interprets model one part at a time, system can be instantly adapted. Interpretive approach is often criticized for being more platform specific [1], because it is considered easier to implement additional generators in Generative MDD then whole executing engines for specific platform, but using platform independent technologies like Java or JavaScript can leverage this problem.

The end verdict between generative and interpretive solutions relies on specific needs. Modern solutions tend towards hybrid approaches [6] with generation and interpretation techniques integrated and combined. This can be achieved by either improving the interpreters with partial code generation or by improving the generators in executing speed so that end-artifacts can be held in main memory and generated on request.

4. OPM Methodology

This section introduces a different kind of modeling methodology called Object-Process Methodology (OPM) [7] together with a simple assessment of different positive aspects which are introduced with OPM and their possible application in field of software modeling. OPM has been successfully applied to modeling generic web applications [8], as well as to modeling data-driven web and mobile applications [2]. It is primarily used as conceptual design tool for complex systems in large industries such as space, electric supplier, military and telecommunication industry\(^6\).

OPM is a universal modeling approach which identifies certain aspect [4]. Some approaches also tend to provide a single model describing whole application like Open Process Graph [5]. OPG is a general purpose executable graph model that incorporates every aspect of an application, including process, user interface, and database in a single graph structure. Interpretive MDD principles are being met because there is no code generated since the graph itself is treated as the code. Instead, a server-side engine interprets the graph alongside user executing the application online.

The key advantage of Interpretive MDD is the ability for changes to be immediately effective and visible. Since executing engine interprets model one part at a time, system can be instantly adapted. Interpretive approach is often criticized for being more platform specific [1], because it is considered easier to implement additional generators in Generative MDD then whole executing engines for specific platform, but using platform independent technologies like Java or JavaScript can leverage this problem.

The end verdict between generative and interpretive solutions relies on specific needs. Modern solutions tend towards hybrid approaches [6] with generation and interpretation techniques integrated and combined. This can be achieved by either improving the interpreters with partial code generation or by improving the generators in executing speed so that end-artifacts can be held in main memory and generated on request.

4. OPM Methodology

This section introduces a different kind of modeling methodology called Object-Process Methodology (OPM) [7] together with a simple assessment of different positive aspects which are introduced with OPM and their possible application in field of software modeling. OPM has been successfully applied to modeling generic web applications [8], as well as to modeling data-driven web and mobile applications [2]. It is primarily used as conceptual design tool for complex systems in large industries such as space, electric supplier, military and telecommunication industry\(^6\).

OPM is a universal modeling approach which identifies certain aspect [4]. Some approaches also tend to provide a single model describing whole application like Open Process Graph [5]. OPG is a general purpose executable graph model that incorporates every aspect of an application, including process, user interface, and database in a single graph structure. Interpretive MDD principles are being met because there is no code generated since the graph itself is treated as the code. Instead, a server-side engine interprets the graph alongside user executing the application online.

The key advantage of Interpretive MDD is the ability for changes to be immediately effective and visible. Since executing engine interprets model one part at a time, system can be instantly adapted. Interpretive approach is often criticized for being more platform specific [1], because it is considered easier to implement additional generators in Generative MDD then whole executing engines for specific platform, but using platform independent technologies like Java or JavaScript can leverage this problem.

The end verdict between generative and interpretive solutions relies on specific needs. Modern solutions tend towards hybrid approaches [6] with generation and interpretation techniques integrated and combined. This can be achieved by either improving the interpreters with partial code generation or by improving the generators in executing speed so that end-artifacts can be held in main memory and generated on request.

5. Modeling Challenges

There are quite a few problems that modern software modeling is facing. Despite its ability to significantly shorten development time and costs; there are some problems that are preventing the modeling tools from breaking into wider usage. They will be analyzed in three following subsections.

5.1. Modeling Complexity

For a model to completely specify a useful application that meets users’ expectations, it would mean too much detail and complexity for an average user to understand. This results in

\(^6\) More information can be found at OPCAT website: http://www.opcat.com/
difficulties with modeling yielding additional education.

Rather than introducing a separate model for each system aspect such as behavior, structure and function, in OPM complexity is handled within single model through abstracting-refining mechanisms. Such mechanisms enable viewing the modeled system at various levels of detail by zooming in and out of diagrams, exposing or hiding lower object and process details. In such way details are easily exposed with maintaining “big picture” view at all times. This enables easier complexity management than classical UML approaches, because it allows modeling from any level of detail working your way up or down, or slowly building on several levels of detail in parallel with gathering facts of system at study.

5.2. Modeling Language

Modeling language is an artificial technical language used to specify knowledge and information about systems or processes in consistent and structural way following a set of rules defined by meta-model specification. Certain modeling tools allow meta-models to be customized, which makes modeling approach called meta-modeling [10]. Expressiveness and nature of modeling language is a direct outcome of its meta-model. Finding an adequate modeling language, or meta-model for a specific modeling domain is the key to easier and natural modeling that requires less symbols to define a system.

OPM methodology uses a single common meta-model which can also be defined by OPM itself. This is why OPM is called reflective: it is complete and expressive enough to define own symbols using nothing but its own symbols. A modeling language that is complete, comprehensive and expressive enough to describe itself, promises enough expressiveness to model almost any system [9].

Another important aspect of modeling language is its representational form which can be graphical or textual. Research showed that graphical languages are not inherently superior to textual languages [11] and that both types of languages have their benefits. Textual form greatly enhances our understanding and communication about systems, thus programming modeling languages often use natural English language as starting base.

OPM methodology enables both completely equivalent forms of modeling: Object Process Diagram as graphical language, and a complete textual counterpart of the graphic language called Object Process Language (OPL) with automatic translations in-between.

5.3. Modeling static and dynamic aspects

The process of modeling involves manual input of desired model using desired modeling tool and language. The most common way of defining the model among available tools includes drawing and connecting symbols on canvas, together with providing names for relationships and objects, as well as choosing some parameters (e.g. cardinality) from in-front defined set of properties. The problem with this approach is that it requires acquaintance with objects and symbols defined in meta-model, as well as with their meaning and visual or textual representation. Further, semantic value of elements is also important to define the model behavior and structure. These characteristics make modeling process a difficult task for average end user, targeting only domain experts that are educated for using these appropriate tools.

MDD is in need for a better form of model input. Some solutions go as far as extracting models from pencil sketches [12]. Main idea is to enhance the process of modeling with instruments that end-user is familiar with, based on more appropriate metaphors. One solution could be exposing domain objects with well-known common user interface elements like text input boxes, drop down menus, elements that end-user is familiar with. Similar principle used in Domain Driven Development methodology is called Naked Objects [13]. The philosophy behind Naked Objects is automatic interface generation by extracting information from object-oriented classes that define domain objects. In [2] an automatic interface generation from OPM model has been achieved which can be seen as similar approach to Naked Objects, but done in MDD environment. What should be done further is to invert this principle in a sense that end user should define user interface of certain domain object yielding an automatic creation of OPM model defining this object. This standardized interface elements would enable a far better metaphor for domain objects then symbols from any kind of modeling tool.

Figure 2 explains this approach on a set of isomorphic representations of exemplary object called “Tviit” used to transfer messages: Figure
Figure 2 Different model representations for modeled object

(a) User interface as modeling form
(b) OPM graphical form
(c) OPM textual form

For modeling dynamic aspects, another approach is required. To avoid classical process modeling like in UML, BPMN,[7] or even OPM, which all use graphical symbols and relationships, a more user friendly approach is necessary. Such approach is observed in PbE, Programming by Example[14], principle which emphasizes programming a software system in its own user interface, or in this example, by interacting with previously placed interface objects. In PbE, user writes a program by providing an example of what the program should do; the system remembers the sequence of actions, and can perform it again. This principle can be used to define application processes in more understandable way, with underlying OPM model composed in background automatically.

5.4. Solving complex problems

PbE and Naked Object principle are much easier to comprehend for end-user but they are difficult to model complicated applications with. MDA targeted complicated and detailed modeling with a goal to cover almost every use case, but paying a high price for it: losing average user as target market. To make our proposal applicable for solving more complex solutions, a third principle is also needed: Component Based Modeling[15] based on principles of Component Based Software Engineering.[8] It is founded on a paradigm common to all engineering practices: complex systems can be obtained by assembling simpler building blocks. For this principle to be user friendly, every component should be represented with some form of interface which makes that component self-describable. Internally, components can be represented by OPM Process blocks.

To recapitulate, this approach proposes a new modeling direction based on three isomorphic forms: (1) Graphical representation with OPM diagrams, (2) Textual representation with OPM language, (3) “By example” representation based on PbE using components exposed as common user interface elements. This is a starting point for a new field of “Modeling by example” techniques in which application model is extracted from end-user actions. This could make software modeling a very powerful solution for end-users with three parallel representation forms to choose from.

6. Conclusion

Model Driven Development is centered on usage of models - simplified representations of certain system. Models are further used to automate the process of software development, partially or fully. There are two main methods of getting end applications from models: code generation or model interpretation. Choosing the right path

---


8 CBSE is branch of software engineering concentrated on separation of concert introduced with components.
depends on modeling task: large enterprise systems are more likely to choose generative approach, while smaller solutions tend to use interpretive approach to avoid complexities. Trends also show usage of hybrid approaches that emphasize on drawing positive sides from both.

Object Process Methodology is a holistic approach to modeling enabling modeling of any kind of system, but it is especially interesting to apply in interpretive modeling approach. OPM is reflective, has solid complexity management, and is able to describe system within single diagram, both static and dynamic aspects.

Modeling solutions in general are still quite complex. Next step in achieving simplicity would be to seek better user interfaces with better metaphors to enable more user friendly modeling. One example of such solution could be modeling the domain objects by composition of common user interface elements and extracting the underlying OPM model from such compositions. To achieve modeling of dynamic aspects, composing components with “programming by example” methods could be used. These two proposals could yield positive steps in simplifying and bringing modeling process closer to end-users.

7. Acknowledgements

This work is supported in part by the Croatian Ministry of Science, Education and Sport, under the research project “Software Engineering in Ubiquitous Computing”.

8. References


