ACTA SCIENTIFIC COMPUTER SCIENCES

Volume 2 Issue 9 September 2020

Review Article

On Modelware as the 5th Generation of Programming Languages

Christian Mancas*

DATASIS ProSoft srl, Department of Computer Science and Mathematics, Ovidius University, Bucharest, Romania

***Corresponding Author:** Christian Mancas, DATASIS ProSoft srl, Department of Computer Science and Mathematics, Ovidius University, Bucharest, Romania.

Received: July 16, 2020

Published: August 19, 2020

Abstract

This paper argues that the 5th generation of programming languages will be modelware, i.e. programming will be replaced by modeling, which will significantly boost productivity and reduce errors, as there are already powerful tools that automatically translate conceptual models into programs of the 3rd and 4th generations.

Keywords: Modelware; Model-Driven Development; 5th Generation Programming Languages; Conceptual Models; The (Elementary) Mathematical Data Model; MatBase

Introduction

Some 80 years ago, the first generation of computer programming languages (machine code) was born [21]. Some 10 years later, the second generation (assemblers/autocoders) appeared. The third generation (high-level programming languages) started exactly 66 years ago, with IBM's Fortran and is still flourishing today. The fourth one, out of which only SQL remained (plus Datalog, but only academically, not commercially), was born exactly 20 years later [7] (some authors include in it object-oriented programming languages too, but, being imperative, algebraic, procedural ones like all other 3rd generation ones, while SQL is declarative, based on first-order logic, we consider them belonging to the 3rd generation as well).

Only 8 years later (i.e. some 38 years ago already), Japan launched a 5th Generation Project (soon paralleled by similar ones in the U.S., U.K., and E.U.), based basically on massively parallel computing and logic programming (KL1, based on Concurrent Prolog, just another 4th generation declarative, first-order logic programming language) over knowledge bases, including techniques from artificial intelligence (AI), fuzzy logic, and neural networks [26]. After some 10 years, despite huge funding, this project failed, mainly because, from the hardware point of view, better and much cheaper architectures emerged and, from the software one, that it was highly dependent on AI technology (which was in its infancy at that time) and on the not at all efficient resolution evaluation paradigm of Prolog.

Anyhow, this evolution from binary coding to higher and higher abstraction levels will always continue for several reasons, out of which the main ones are the general desire of mankind to ever develop better and better tools allowing people to be much more productive with less effort (i.e. "we want") and the experience got so far (i.e. "we can").

Conceptual models as programs

From antique geographic maps to the Nash Equilibrium theory, from Maxwell's equations of the electro-magnetic waves (established some 200 years before discovering of the radiophone…) to the Big Bang theory, etc., mankind always designed and used conceptual models. For simplicity, we consider conceptual models to be well-defined, purposeful, coherent, and self-sufficient abstract tools formulated in some well-structured language.

Even if still mostly used only to inspire computer programming and/or to document and understand legacy code, from algorithm workflow diagrams (flowcharts, [17]) to Entity-Relationship ones [8,22], and to UML [4] and graph [2] ones, computer science too is taught and applied with the help of conceptual models.

Conceptual modeling eventually emerged as a compulsory step in the software development process (e.g. [13,22,25]). Little by little, it became clear that, by associating to it corresponding code generators, conceptual modeling will become the 5th generation of programming languages (e.g. [29,33]).

For example, since some 15 years ago already, model-driven programming was considered (e.g. [11,32]). Then, for example, Broy., *et al*. [5] proposed a unification of modeling and programming. This idea was not at all new: 18 years earlier, for example, Hürlimann [18] introduced modeling languages that combine de-

clarative and algorithmic aspects of programming. One such example, designed for mathematical programming, is AMPL [15]. Then, for example, Völter [34] discussed it again in that technological time and model-driven context.

Moreover, several steps were already taken for automatically generating code in 3rd and 4th generation programming languages from such models. For example, a plethora of model-driven tools (e.g. [31]) were already developed, e.g., Xpand [12], PathMATE [30], SMARTGenerator [3], Acceleo [1], Integranova [20], MagicDraw [27], openMDX [28], Rhapsody [19]. Except for MagicDraw, all of them do not support databases (dbs), but only software applications. Dually, products like the famous erwin suite [14] only support dbs.

"Modelware" has long been used for the building of models and associated modelling tools for software systems [6,35,36]. The term "modelware" is also used in AI, but with a different meaning, namely: a reused object that makes the machine learning model reusable and reproducible [37].

More and more of us understand today by modelware a declarative executable problem domain model of what (not how) the computer should do. Its input should be a model expressed in some data and workflow formalism and its output a corresponding db application for managing data according to the workflow and the business rules formalized in the model as constraints.

It was proven that, through the use of sets of innovative modeling techniques, supported by integrated model-driven development (mdd) tool suites, orchestrated by mdd processes and methodologies that enable scalable model-driven development software, productivity increases, while error possibilities (and hence, debugging) dramatically decreases (e.g. [10]).

For example, the (Elementary) Mathematical Data Model ((E)MDM, [23,25]) is such a formalism – an example of a 5th generation programming language. With both its Entity-Relationship Data Model and, especially, (E)MDM graphic user interfaces (GUI), MatBase is one of the modelware tools that saves programming, letting its users to solely focus on conceptual data modeling [24]. Starting from a model, both of its versions (MS Access and C# and SQL Server, respectively) automatically generates both the corresponding db, GUI, and event-driven methods in object-oriented classes for enforcing the non-relational constraints [23,25].

Dually, however, it is also true that some authors advocate the contrary: for example Cleaveland [9], in the context of embedded-software development, states that programming should rather be considered a form of modeling, with the programming language constituting an executable metalanguage, in which models of dynamical systems are encoded and simulated, or "run". Modeling instead plays a design role, with some models used as a basis for the generation of code. It is true that the embedded-software context is a very peculiar one, due to its drastically limitations in both available memory size and required execution speed, but also to its significant degree of simplicity as compared to general information systems.

Conclusion

The strive for efficient software development, both in productivity gain and in the reduction of errors, needs a higher-level programming paradigm than the current standard ones (belonging to the 3rd and 4th generations). The first attempt in this direction, done some 40 years ago, failed as it was too early both conceptually and technologically.

Latest developments in this field, both academical and commercial, strongly suggest that the 5th generation level will be the modelware in its latest coordinates, i.e. programming will solely consist in modeling, with model-driven system software automatically translating conceptual models into 3rd and 4th language generation programs.

For example, [16] reports that in a project developed using such an approach, productivity increased more than 10 times and, as an expected bons, there has also been a 90% reduction of the errors.

Bibliography

1. Acceleo. Acceleo (2019).
2. Bang-Jensen J and Gutin GZ. Digraphs: Theory, Algorithms and Applications (2nd edition). Springer-Verlag, London (2009).
3. BITPlan. SMARTGenerator (2018).
4. Booch G., *et al*. The Unified Modeling Language User Guide 2nd ed. Addison-Wesley Professional/Pearson Education, NJ (2005).
5. Broy M., *et al*. "Towards a Unified View of Modeling and Programming". In: Margaria T., Steffen B. (eds) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications. ISoLA 9953 (2016): 238-257.
6. Bryant BR. "Grammarware, Semantics and Modelware" (2019).
7. Chamberlin DD and Boyce RF. "SEQUEL: A Structured English Query Language". In: Proc. ACM SIGFIDET Workshop on Data Description, Access and Control (1974): 249-264.

8. Chen PP. "The entity-relationship model: Toward a unified view of data". ACM TODS 1.1 (1976): 9-36.

9. Cleaveland R. "Programming is modelling". In: Margaria T., Steffen B. (eds) Leveraging Applications of Formal Methods, Verification and Validation. Modeling. ISoLA 11244 (2018): 150-161.

10. Cordis. Modelware: a new approach to model-driven development" (2005).

11. Cretu LG and Dumitriu F. Model-Driven Engineering of Information Systems. Principles, Techniques, and Practice. Apple Academic Press, Palm Bay, FL (2014).

12. Eclipse Foundation. Xpand (2016).

13. Embley DW and Thalheim B. Handbook of Conceptual Modelling. Theory, Practice and Research Challenges. Springer, Berlin, Germany (2011).

14. erwin. erwin EDGE Portfolio (2020).

15. Fourer R., *et al*. AMPL. A Modeling Language for Mathematical Programming 2nd ed. Duxbury/Thomson (2003).

16. Genicore AB. Modelware Is the New Software. Digitization of Project Lazarus (2017).

17. Goldstine H and von Neumann J. Planning and coding of problems for an electronic computing instrument, Part II, Volume 1. In: Taub, A. (Ed., 1963), John von Neumann Collected Works. 5. Macmillan. (1974): 80-151.

18. Hürlimann T. "Modeling Languages: A New Paradigm of Programming" (1998).

19. IBM. IBM Rational Rhapsody family (2020).

20. Integranova Software Solutions. Integranova (2020).

21. Knuth DE and Pardo LT. "Early development of programming languages". In: Encyclopedia of Computer Science and Technology. Marcel Dekker, NY 7 (1977): 419-493.

22. Mancas C. Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume I: The Shortest Advisable Path. Apple Academic Press/CRC Press (Taylor and Francis Group), Palm Bay, FL (2015).

23. Mancas C. "MatBase Constraint Sets Coherence and Minimality Enforcement Algorithms". In: Benczur, A., Thalheim, B., Horvath, T. (eds.), Proc. 22nd ADBIS Conf. on Advances in DB and Inf. Syst., LNCS. Springer, Cham, Switzerland 11019 (2018): 263-277.

24. Mancas C. "MatBase - a Tool for Transparent Programming while Modeling Data at Conceptual Levels". In: Proc. 5th Int. Conf. on Comp. Sci. and Inf. Techn. AIRCC Pub. Corp., Chennai, India (2019): 15-27.

25. Mancas C. Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume II: Refinements for an Expert Path. Apple Academic Press/CRC Press (Taylor and Francis Group), Palm Bay, FL (in press) (2021).

26. Moto-oka T. Fifth Generation Computer Systems. North Holland (1982).

27. No Magic. MagicDraw (2020).

28. openMDX. openMDX (2020).

29. Pastor O and Ruiz M. "From Requirements to Code: A Conceptual Model-based Approach for Automating the Software Production Process". *International Journal of Conceptual Modeling* (2018): 274-280.

30. Pathfinder Solutions. PathMATE (2017).

31. Soukaina M., *et al*. "Model Driven Engineering (MDE) Tools: A Survey". *American Journal of Science, Engineering and Technology* 3.2 (2018): 29-33.

32. Stahl T and Völter M. "Model-Driven Software Development. Technology, Engineering, Management. John Wiley and Sons, Chichester, U.K (2006).

33. Thalheim B and Jaakkola H. "Models as Programs: The Envisioned and Principal Key to True Fifth Generation Programming". In: Proc. 29th European-Japanese Conf. (2019): 170-189.

34. Völter M. "From Programming to Modeling - and Back Again". *IEEE Software* 28 (2011): 20-25.

35. Wimmer M and Kramler G. "Bridging grammarware and modelware". In: Proc. Satellite Events at the MoDELS 2005 Conf., LNCS 3844 (2005): 159-168.

36. Withears D., *et al*. "Software/Modelware Application Requirements (Panel)". In: Proc. IEEE 1992 Winter Simulation Conf. (1992): 205-210.

37. Zhao H., *et al*. "Domain-Specific Modelware: To Make the Machine Learning Model Reusable and Reproducible". In: Proc. 12th ACM/IEEE Int. Symp. on Empirical Soft. Eng. and Measure., ACM, NY (2018).