The Conceptual Model Is The Code. Why Not?

Oscar Pastor and Vicente Pelechano

Abstract The selection of the paper entitled "OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods" for this book on Advanced Information Systems Engineering allows us to reflect on the research context where the work was developed and presented (in "CAiSE 1997") and to introduce its main contributions, how they have evolved with time and what influence the approach could have in the emergence of the Model-Driven Engineering domain. As the main goal was to provide a Software Process that should be fully Conceptual Model-based, the central message of this chapter is still the same 16 years later: the Conceptual Model must be the key software artefact of a sound, correct and complete Software Production Process. Novel approaches were required to generate a sound software production process, and they should use conceptual models as the key software artefact. The model should be the code of the application, and a conceptual modelling programming style should become a reality. While historically Software Engineering is in practice focused on programs, we have always tried to provide methods and tools to achieve the objective of make modelling the essential activity of programming. Why not making true the statement that "the model is the code?". This was our point when we published our referred CAiSE paper, and it is still our position now, with many more results and experiences to support it, that we introduce throughout this work.

O. Pastor (✉) • V. Pelechano
Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
e-mail: opastor@pros.upv.es; pele@pros.upv.es

J. Bubenko et al. (eds.), *Seminal Contributions to Information Systems Engineering*,
DOI 10.1007/978-3-642-36926-1_12, © Springer-Verlag Berlin Heidelberg 2013

1 Introduction

Among many other significant improvements, last century nineties was the time of CASE methodologies. Providing a complete software process intended to correctly support analysis, design and implementation became a priority. Some proposals that stemmed from Structured Analysis and Design [1, 2] started to apply notions drawn from Object-Oriented (OO) programming to conceptual modelling. A plethora of OO methods were proposed (e.g. [3–6]), with different methodological backgrounds and a diversity of notations, paving the way for the creation of a Unified Modeling Language called UML [7].

Much effort was devoted to investigating and providing a software process capable to guarantee the quality of the final application code. The CASE tools that were constructed constituted a serious attempt to automate, in some degree, the software production process. The strategy was to define a set of models in a conceptual modelling step that should be properly transformed first in a software design, and then in a software code. But unfortunately, too often these CASE tools generated a frustrating experience in practice. Instead of providing a more effective and efficient solution to the software development process, as they were committed to do, users perceived that the tools were adding an additional burden to the problem of programming. Programming was still a big challenge, but additionally now a new method and its modelling language had to be learnt to create models that still had to be converted into code.

In any case, it was becoming clear that the ideas stated already in 1971 in [8] about the automation of systems building were more alive than never: "the size, importance and cost of systems building provides an opportunity for the investigation of ways to improve the process." These new ways had an increasingly conceptual model-oriented perspective, and eventually conceptual models were playing the wished role of essential software artefact.

It was in this historical context where two CAiSE papers [9, 10] were introducing an approach that had the following original and relevant aspects:

- The proposal of a formal, OO specification language –OASIS- that contained the conceptual primitives required for specifying an organizational system, with a precise semantics.
- The definition of an ontology for information systems, based on the FRISCO proposal [11], to characterize those basic concepts that should be present in a modelling language.
- The creation of a methodological background clearly distinguishing between Problem Space (conceptual model-based, focusing on "what" the system is) and Solution Space (final software product, centred on "how" an appropriate support is going to be provided), together with the specification of an Execution Model intended to link the conceptual primitives of the Conceptual Model (Problem Space) with their corresponding software representation in the final software application (Solution Space).

The pair constituted by the OO specification language (OASIS) and its methodological support (called the "OO-Method") conformed a rigorous contribution that will be presented with full historical detail in Sect. 3.

This work did not come alone. In the following years, a set of proposals, methods, and tools were generated following the same direction, creating a family of approaches that shared a common goal. Instead of having a Software Engineering approach based on the principle that "the code is the model", the new conceptual modelling approaches promoted just the contrary: the model should just be the code. All these proposals have made the dream of automating systems development closer to truth than ever. The most relevant works are summarized in Sect. 2.

Finally, projections of the results reported in this work in other challenging domains will be discussed as further work in Sect. 4. Conclusions and a list of references complete the chapter.

2 Model-Driven Development in Practice: The "Model Is the Code" Versus "The Code Is the Model"

Assuming that programs are models of implementations, one may argue that the main challenge of software engineering is to see Conceptual Models as higher-level programs and to provide sound transformations to convert those conceptual models (Problem Space representation) into code (Software Solution representation). Such a full software process should start with the elaboration of a Requirements Model, and continue with its subsequent transformation into its associated Conceptual Schema that should be executable through a Conceptual Model Compilation process.

The essential principles behind the OO-Method proposal [10] were turn into reality by the implementation of the Integranova Conceptual Model Compiler [12], which was developed and used in an industrial environment.

Morgan introduced in [13] the notion of "Extreme Non-Programming (XNP)", opposing Extreme Programming to highlight that XNP programmers should have a conceptual modelling perspective. This means that they should not do programming at all –at least they should not program in the traditional programming sense-. Instead, they should follow the motto "the model is the code".

Olivé proposed in [14] the concept of "Conceptual Schema-Centric Software Development", proposing a precise criteria to support it: to design an Information System, it is necessary and sufficient to create its Conceptual Schema". Not only necessary, but necessary and sufficient.

In the same line of argument, Model-Driven Engineering (MDE), which is also referred to as Model-Driven Development (MDD) or Model-Driven Architecture (MDA) [15], advocates in the recent years the creation of software systems by model specification. This movement has supposed a strong push to all the ideas that are discussed here, and a plethora of methods and tools have started to appear under the common, accepted assumptions that (i) models ought to be used as the key

software artefacts, and (ii) models are to be seen as abstract conceptualizations of particular domain concepts, rather than algorithmic specifications written in a high-level language. Conceptual modelling becomes then the primary means of software production.

More recently, the "Conceptual Modelling Programming" manifesto [16] puts together all these principles, focusing on the importance of three basic ideas: (i) conceptual modelling is programming, (ii) the conceptual model, with which modellers program, must be complete and holistic, and conceptual but precise, and (iii) application evolution must occur at the level of the model.

This selection of approaches provides a solid basis to understand the potential of the ideas discussed in [10, 17] to show how effectively they influenced the advances that lead to the existing MDE approaches, and to analyse how fruitful their evolution has been and is still being.

3 The OO-Method Approach: Past, Present and Future

Let us focus now on the most relevant ideas that conformed the contribution presented in the OO-Method Approach [9, 10, 17], how they have evolved, and what is their intended projection for the very next future.

In a context where the terms MD* (Model-Driven Development, Model-Driven Architecture, Model-Driven Engineering, etc.) and Model Transformations did not exist yet, the OO-Method introduced the following remarkable features [10]:

(a) The use of a formal specification language as a support to characterize the modelling primitives that are required for designing Organizational Information Systems. This provided an ontological commitment for the precise conceptual characterization of the building units of a Conceptual Schema. Since that moment, Ontology Engineering, Metamodelling-based approaches and Conceptual Modelling-based techniques have evolved towards the challenge of elaborating a sound and full Software Process based on Conceptual Modelling..
(b) A strategy for executing Conceptual Schemas –so called Execution Model- that basically defined a set of mappings between the conceptual primitives of the Conceptual Model and their corresponding software representation counterpart in the selected target software development environment.

These two contributions together paved the way to the implementation of a Conceptual Model Compiler, as it happened with the design and implementation of Integranova, a Conceptual Modelling Programming tool created by CARE Tech [12] that created an industrial tool to put into practice all the ideas behind the OO-Method approach.

Since then, the approach has had to be adapted to the appearance of new software development environments, which means that the Conceptual Model Compiler must be always ready to evolve in two ways: finding out better software representations and adapting to diverse software architectures that guide the software generation

process and that require to extend the Conceptual Model Compiler offer. As the Conceptual Model level is stable, whenever a new development environment (e.g. a new programming language) is targeted, new mappings between the conceptual primitives and their software representation counterpart in the "new" environment are to be properly designed and implemented.

The future of the approach is related to the "Requirements Engineering" (RE) connection that should provide a full Software Process coverage for the method. This will be more detailed in the next section.

4 What Is Next?

Several lines of both, theoretical and applied research, have given a challenging continuity to the results that were originated by the work presented in [10].

Firstly, the ideas originally applied to the context of Organizational Systems were extended to other IS domains. A set of works designed and implemented a similar type of Conceptual Modelling-based Software Process to: (1) Specify and Implement Web Applications (by building the OOWS methodological approach [18]), (2) Specify and Generate code for AmI systems (by providing the PervML methodological approach [19] and (3) Specifying and Generating Business Process Driven Web Applications [20]). New conceptual primitives have conformed new conceptual models, and the subsequent Conceptual Modelling Programming environments have been designed and implemented. Currently, we are providing solutions in the Software Engineering field to tackle with the new technological and engineering challenges such as those introduced by the development of the Internet of Things (integration of the physical and logic worlds) [21] and the Autonomic Computing (reconfiguration, adaptability at run-time of services and user interfaces) [22].

Secondly, once the transformation of a Conceptual Schema (PIM in MDA terms) into code has been defined by constructing a Conceptual Model Compiler (that contains the PSM logic in MDA terms), the process is to be extended with what is was called the Requirements Engineering (RE) connection above. This means that the Conceptual Schema must be seen as the output of a higher-level model –the Requirements Model (RM), the CIM in MDA terms)- This RM must be defined, together with a sound transformation intended to create its corresponding Conceptual Schema with as much automation as possible. This is probably not a fully automated process, because the Conceptual Schema must add some information that is not present yet at the requirements modelling step. But the metaphor of moving from Requirements to Code through a precise, well defined set of models and model transformation is closer than ever to become a reality. Some steps in this direction have been already taken (see [24]), but much work is still to be done to answer the questions (i) what RM should be selected (ii) how to define the corresponding model transformation.

Thirdly, a very interesting perspective is to think about further domains were all these ideas could be used to improve the current software development process

and obtain better results. Some challenging candidates can be aircraft control weather prediction, vehicle mobile clouds, digital TV, video-games, etc. But there is one especially appealing that is the modelling of life. Conceptual Modelling of the Human Genome can provide a different perspective of the same problem: considering alive beings "implementations" of a (genetic) code, the problem is to understand the modelling primitives that could make feasible to define models and to understand how these models are converted into the final code (the human being). The clinical projection of this challenge is especially interesting, intended to apply a conceptual modelling-oriented approach to find out and manage the "bugs" (illnesses) that are a consequence of a (genetic) code mistake. Some previous promising results have been reported in [23].

5 Conclusions

Producing a sound information system design and implementing such design into a software product of high quality sounds simple, but it is still a nightmare for Software and Information Systems Engineering. The well-know problems often referred to as the *crisis of software* remain alive. In most of the complex software projects, the design, programming and testing activities still require a substantial manual effort and are keep being error-prone. From the point of view of conceptual modelling and the role of models, we claim that the software development process has not changed much over the past 40 years. We mean that the "program" has been and still is often considered the essential software artefact. Trying to prioritize conceptual modelling over programing,, many attempts have promoted that "the model should be code" instead of insisting that "the code is and will ever be the model". Assuming that looking for a different way for producing software was worth to be explored, we presented in [10] an approach that intended to fulfil that goal. Through a clear separation between Problem Space (Conceptual Schema) and Solution Space (application code), a ontologically well-founded modelling environment was presented, together with an execution strategy to transform the modelling primitives into software components through a process of conceptual-model compilation. This was one of the first works presenting a concrete solution for a domain that a few years later was extensively explored under the model-driven development paradigm, for which it could be argued that it was indeed a very significant contribution.

References

1. DeMarco, T., Structured analysis and system specification. 1979, Englewood Cliffs, New Jersey: Yourdon Press.
2. Ward, P.T, Mellor, S. Structured Development for Real-Time Systems: Essential Modeling Techniques. Prentice Hall.

3. Rumbaugh, J., Blaha. M, Premerlani. W, Eddy. F, Lorensen. W. Object-Oriented Modeling and Design. Prentice Hall. 1999.
4. Booch. G, Maksimchuk. R. A., Engel. M. W., Young. B.J. Object-Oriented Analysis and Design with Applications. Addison-Wesley
5. Jacobson. I., Christerson. M, Jonsson. P, Overgaard. G. Object-Oriented Software Engineering: A Use Case Driven Approach (ACM Press). Addison-Wesley, 1992,
6. Wirfs-Brock. R.J. Designing Object-Oriented Software, with Brian Wilkerson and Lauren Wiener, Prentice-Hall, 1990
7. Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley.
8. Teichroew, D., Sayani, H.: Automation of System Building, Datamation (1971).
9. Pastor. O, Hayes. F., Bear. S. OASIS: An Object-Oriented Specification Language. CAiSE 1992: 348–363
10. Pastor. O, Insfrán. E, Pelechano. V., Romero. J.R., Merseguer. J. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. CAiSE 1997: 145–158.
11. Falkenberg. E.D., Hesse. W., Lindgreen. P., Nilsson. B.E., Oei. J.L.H., Rolland. C., Stamper. R.K., Van Assche. F.J.M., Verrijn-Stuart. A.A., Voss. K. FRISCO : A Framework of Information System Concepts, The IFIP WG 8.1 Task Group FRISCO, December 1996.
12. Integranova Software Soluctions. Available on: http://www.integranova.com/. Last Access: January 19, 2013.
13. Morgan, T.: Business Rules and Information Systems – Aligning IT with Business Goals. Addison-Wesley, Reading (2002).
14. Olivé, À.: Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 1–15. Springer, Heidelberg (2005).
15. Booch. G, Brown. A., Iyengar. S., Rumbaugh. J., Selic. B. An MDA Manifesto. The MDA Journal: Model Driven Architecture Straight from the Masters, pages 133–143, 2004.
16. Embley. D. W., Liddle. S.W, Pastor, O. Conceptual-Model Programming: A Manifesto. Handbook of Conceptual Modeling, 2011, pp 3–16. Springer.
17. Pastor, O., Gomez, J., Insfrán, E., Pelechano, V.: The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. Information Systems 26(7), 507–534 (2001).
18. Fons. J, Pelechano. V, Albert. M, Pastor. O. Development of Web Applications from Web Enhanced Conceptual Schemas. ER 2003: 232–245.
19. Muñoz. J., Pelechano. V. Building a Software Factory for Pervasive Systems Development. CAiSE 2005: 342–356
20. Torres. V., Giner. P., Pelechano. V. Developing BP-driven web applications through the use of MDE techniques. Software and System Modeling 11(4): 609–631 (2012)
21. Giner. P., Cetina. C., Fons. J., Pelechano. V. Developing Mobile Workflow Support in the Internet of Things. IEEE Pervasive Computing 9(2): 18–26 (2010)
22. Cetina. C., Giner. P., Fons. J., Pelechano. V. Vicente Pelechano: Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. IEEE Computer 42(10): 37–43 (2009)
23. Oscar Pastor, Juan Carlos Casamayor, Matilde Celma, Laura Mota, M. Ángeles Pastor, Ana M. Levin: Conceptual Modeling of Human Genome: Integration Challenges. Conceptual Modelling and Its Theoretical Foundations 2012: 231–250
24. Oscar Pastor, Sergio España: Full Model-Driven Practice: From Requirements to Code Generation. CAiSE 2012: 701–702