Models: The Main Tool of True Fifth Generation Programming

Bernhard Thalheim[1] [0000-0002-7909-7786], Alexander Sotnikov[2] [0000-0003-2985-3704], and Igor Fiodorov[3] [0000-0003-2335-0452]

[1] Christian-Albrechts-University Kiel, Computer Science Department, Germany
[2] Joint Supercomputer Center, Russian Academy of Sciences, Russia,
[3]Plekhanov Russian University of Economics, Moscow Russia
[1]Thalheim@is.informatik.uni-kiel.de, [2]ASotnikov@jscc.ru,
[3]Igor.Fiodorov@mail.ru

Abstract. Models are one of the main and most commonly used instruments in Computer Science and Computer Engineering. They have reached a maturity for deployment as the main tool for description, prescription, and system specification. They can be directly translated to code what enables us to consider models as the main tool for modern software development. Models are the power unit towards new programming paradigms such as true fifth generation programming. This paper introduces model-centered programming as one of the main ingredients and main tool of true fifth generation programming.

Keywords: models, true fifth generation programming, model-centered programming.

1 Introduction

1.1 Towards New Programming Paradigms

Programming has become a technique for everybody, especially for non-computer scientists. Programs became an essential part of modern infrastructure. Programming is nowadays a socio-material practice in most disciplines of science and engineering. Solution development for real life complex systems becomes however an obstacle course due to the huge variety of languages and frameworks used, due to impedance mismatches among libraries and environments, due to vanishing programming expert knowledge, due to novel and partially understood paradigms such as componentization and app programming, due to the inherent tremendous complexity, due to programming-in-the-large and programming-in-the-web, and due to legacy and integration problems.

Programming languages have evolved since early 1950's. This evolution has resulted in a thousand of different languages being invented and used in the industry.

[2]The work was done within the framework of the state assignment (research topic: 065-2019-0014 (reg. no. AAAA-A19-119011590097-1)

Proceedings of the XXII International Conference «Enterprise Engineering and Knowledge Management" April 25-26, 2019, Moscow, Russia

First generation languages – although low-level and machine-oriented at micro-code level - are still used for instruction-based programming. Second generation languages are assembly languages that can be translated to machine language by an assembler. Third-generation languages provide abstractions and features such as modules, variables, flow constructs, error handling, support packages, many different kinds of statements etc. Fourth generation languages are more user friendly, are portable and independent of operating systems, are usable by non-programmers, and have intelligent default. The fifth generation project has been oriented on logic programming and did not result in a wide acceptance and usage. The main supporting feature for programming is however that programs written in these languages are translated by compilers to programs in low-level machine languages.

Programs became an infrastructure of the modern society. At the same time, we face a lot of problem for such infrastructure. Its maintenance, extension, porting, integration, evolution, migration, and modernization become an obstacle and are already causing problems similar to the software crisis 1.0. Programs are developed in a variety of infrastructures and languages that are partially incompatible, in teams with members who do not entirely share paradigms and background knowledge, at a longer period of time without considering legacy problems at a later point of time, without development strategies and tactics, and with a focus on currently urgent issues. A crucial point is the development of critical software by non-professionals. Programming has already changed to programming-in-the-large beyond programming-in-the-small and is going to change now to programming-in-the-mind. Moreover, systems become more complex and less and less understandable by team members. The software crisis 2.0 (e.g. [15]) is also be exacerbated by understandability, communication, comprehension, complexity, and provenance problems.

We thus need better and more abstract techniques for development of our programs. We envision that ***true fifth generation programming*** can be based on ***models and model suites*** which can be automatically transformed to corresponding programs without additional programming.

1.2 Models as Programs

Our notion and understanding of models and model suites is based on the compendium on models in sciences and engineering [11].

A *model* is a well-formed, adequate and dependable instrument that effectively and successfully functions in utilization scenarios. It is *adequate* if it is analogous to the origins to be represented according to some analogy criterion, if is more focused (e.g. simpler, truncated, more abstract or reduced) than the origins being modelled, and if it sufficiently satisfies its purpose. Well-formedness enables an instrument to be *justified* by an empirical corroboration according to its objectives, by rational coherence and conformity explicitly stated through conformity formulas or statements, by falsifiability or validation, and by stability and plasticity within a collection of origins. The instrument is sufficient by its quality characterization (internal quality, external quality and quality in) such as correctness, generality, usefulness, comprehensibility, parsimony, robustness, novelty etc. Sufficiency is typically combined with some as-

surance evaluation (tolerance, modality, confidence, and restrictions). A well-formed instrument is called dependable if it is sufficient and is justified for some of the justification properties and some of the sufficiency characteristics. A model comes with its background, e.g. paradigms, assumptions, postulates, language, thought community, etc. The background its often given only in an implicit form.

A model reflects only some focus and scope. We thus use *model suites* that consists of a set of models, an explicit association or collaboration schema among the models, controllers that maintain consistency or coherence of the model suite, application schemata for explicit maintenance and evolution of the model suite, and tracers for the establishment of the coherence.

A typical model suite is used for co-design of information systems that is based on models for structuring, models for functionality, models for interactivity, and models for distribution. This model suite uses the structure model as the lead model for functionality specification. Views are based on both models. They are one kernel element for interactivity specification. Distribution models are additionally based on collaboration models.

Model-centered development is used in many branches of modern computer science and computer engineering. Model-as-Programs approaches will become the traction machine characterized by slow beginning at present and a progressive increase in speed. In the sequel we discuss this change of paradigms for database development. A similar approach has already been practiced for editing systems such as literate programming and as the LaTeX environment or such as compiler-compiler approaches for domain-specific languages.

1.3 The Storyline of this Paper

Models and model suites became easy-to-use and easy-to-develop instruments that are used by everybody and therefore also by non-programmers. We envision that modern programming could be based on model suites that are translated to programs. As discussed in Section 2, this vision is already real for users that use advanced database development techniques. However, model-based database programming is used only in a less sophisticated and rather implicit form. Investigating the more advanced approach, we develop a path towards true fifth generation programming that is based on model suite development in Section 3. The entire framework is inspired by and can be considered as a generalization model-centric database development and modern specification approaches.

2 Case Study: Model Suites Direct Database Specifications

2.1 Data Specification for Database Applications with Conceptual Models

Conceptual schemata and models are widely used for database structure specification and as a means for derivation of user viewpoints [10,13]. These models are used for concept-backed description of the application domain or of thoughts, for prescription of the realization and thus system construction, for negotiation and iterative develop-

ment of the model decisions, and for documentation and explanation of the decisions made in the modelling process.

Viewpoints can be represented by view schemata that are defined on the main database schema by expressions given in an advanced algebra. Interaction models for business users are the third kind of models that are used in a database model suite. Collaboration models can be specified in a similar form and are based on viewpoints.

The usage of a conceptual model as a description model of thoughts and understanding in an application area is commonsense today. The usage for system realization must be based on specific properties of the database management platform and requires thus a lot of additional information. We thus enhance conceptual modelling by additional information. Pragmas and directives are essential elements that we use for enhancement of conceptual models for system realization. Pragmas have originally introduced for C and C++. Directives have been used as additional control units for compilation.

2.2 Transformation of Conceptual Models to Logical and Physical Models

Conceptual models and schemata are often taken as an initial structure for logical and physical schemata. The transformation is still often based on some brute-force interpreter approach that requires corrective specification for integrity maintenance and for performance management at a later stage by experienced database operators. The transformation of integrity constraints is not yet automatically enhanced by enforcement mechanisms and control techniques. Procedural enhancement on the basis of triggers and stored procedures is still a challenge for database programmers. Performance support includes at the first step CRUD supporting indexing. Support for querying can be based on hints.

The transformation approach can however be based on rule-based compilation. Essentials of rule-based transformation are in a nutshell: syntactical and semantic analysis of the models and schemata according quality characteristics within the platform setting; preprocessing of the models and schemata to intermediate normalized models and schemata; extension of the models by support models for performance support; derivation of integrity maintenance and other support schemes; derivation of association schemata for models in the model suite; derivation of tracers for coherence maintenance; rule-based transformation of models; optimization after transformation.

It does not surprise that this approach follows classical four-layer compiler technologies (lexical and syntactical analysis, derivation of intermediate models, preparation for optimization, translation, performance management) [14]. It is enhanced by a compiler configuration pragmas according to the profile of the DBMS. The models must be complete for performance consideration. Therefore, a number of directives have to be added to all models in the model suite: treatment of hierarchies, redundancy control, constraint treatment, realization conventions, and quantity matrices for all larger classes. Directives and pragmas must not be fully described. Instead we may use templates, defaults and stereotypes, e.g. realization style and tactics, default configuration parameters (coding, services, policies, handlers), generic operations, hints

for realization of the database, strategies for matching performance expectations, constraint enforcement policies, and support features for the system realization.

This transformation approach is already state-of-the-art for challenging applications. Advanced database programming is based on such techniques. Web information systems development uses such transformations [10]. However, it is currently the professional secret of database operators and administrators.

2.3 Generalizing the Approach for Database Programming

The specification and transformation approach has already becoming common practice for database structuring development. The Higher-Order Entity-Relationship Modelling (HERM) language [13] is the basis for development of conceptual database schemata and for specification of derivable view schemata. The latter are used for support of viewpoints for a given database system user community. Derivation is based on the HERM algebra that allows specification of user schemata. The specification of a database schema follows the disciplinary matrix of database development, e.g. approaches such as global-as-design and viewpoint support as derivable structures. This structuring may be enhanced by generic or reference models which are essentially package for modelling. The foundation and the models background is supported by the HERM theory. The entire schemata development is based on tools, e.g. ADOxx [4,6] as a specification environment. An essential element of this environment is a compiler for compilation of the specifications to logical schemata. The database developer specifies the schemata within this environment as HERM schemata, view schemata, and pragmas and directives as an additional description for database performance.

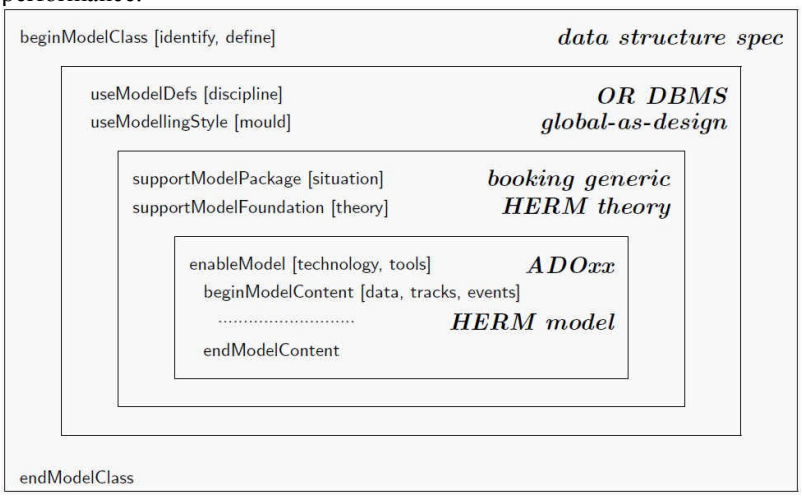

Fig. 1. The model-centric database structure development with automatic mapping of conceptual models to logical models for HERM models

Fig. 1 displays this approach. The database developer uses the development environment within the environment of the ADOxx workbench, the model definitions provided for HERM, the packages for schemata (e.g. a generic reference model for booking applications), the foundations provided by the HERM theory, and the transformation features embedded into the ADOxx generator [6].

3 Model Suites Used as Programs

3.1 Towards New Programming Paradigms

Modern programming languages provide as much as possible comfort to programmers.

Models are a universal instrument for communication and other human activities. Thought chunks can be presented to those who share a similar culture and understanding without the pressure to be scientifically grounded. Models encapsulate, represent and formulate ideas both as of something comprehended and as a plan. They are more abstract than programs. They can be as precise and appropriate as computer programs. They support understanding, construction of system components, communication, reflection, analysis, quality management, exploration, explanation, etc. From the other side, models can be translated to programs to a certain extent. So, models can be used as higher-level, abstract, and effective programs. Models are however independent of concrete programming languages and environments, i.e. programming language and environment independence is achieved. Models declare what exactly to build. They can be developed to be understandable by all main parties involved in system development. They become general enough and accurate enough. They can be calibrated to the degree of precision that is necessary for high quality.

Model-based programming can then replace classical programming based on compilation and systematic development of models as well on explicit consideration of all model components without hiding intrinsic details and assumptions. Our approach to true fifth generation programming will be extendable to all areas of computer science and engineering beside the chosen four exemplary ones (information system models; horizontally and vertical layered models; adaptable and evolving models; service line models). This paper develops a general framework to true fifth generation programming for everybody.

The framework is based on model suites since the user interface models and the collaboration models must be an integral part of modelling. Interface and collaboration treatment generalizes literate programming [5] to literate modelling as ‘holon’ programming that is combined with schemes of cognitive reasoning. Model suites enable the programmer of the future to develop their programs in a multi-facetted way. They can reason in a coherent and holistic way at the same time on representation models as the new interfaces, on computing and supporting models, on infrastructure models, on mediating models for integration with other systems, etc.

Application engineers and scientists are going to develop and to use models instead of programming in the old style. They will be supported by templates from their application area, can thus concentrate on how to find a correct solution to their prob-

lems, can manage the complexity of software intensive systems, will be supported by model-backed reasoning techniques, and will appreciate and properly evaluate the model suite at their level of abstraction. Literate modelling with model suites supports all members of a community of practice (CoP) by reflecting their needs and demands in a given situation and scenario by an appropriate model in the model suite. It becomes thus an effective and efficient means of communication and interaction for users depending on their beliefs, desires, needs, and intentions.

The generalization of the database approach is depicted in Fig. 2. We use a similar form as the experienced one that is displayed in Fig. 1.

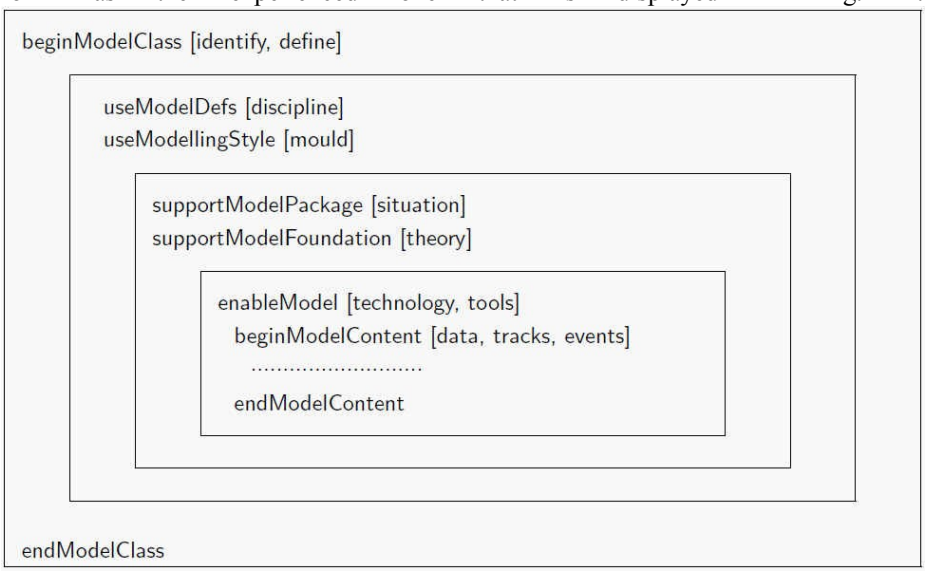

Fig. 2. The general approach to true fifth generation development based on models

Our approach proposes new programming paradigms, develops novel solutions to problem solving, integrates model-based and model-backed work into current approaches, and intents to incubate true fifth generation programming. This new kind of programming enhances human capabilities and could become the kernel of new industrial developments. Models are thus programs of the next generation.

3.2 The Layered Approach to Modelling

Our approach is based on model suites as the source, on systematic development of model suites in a layered approach, on compilers for transformation to programs in third or fourth generation, and on quality assurance for the model as a program. The notion of the model suite is based on [11]. Model suites generalize approaches developed for model-driven development from one side and conceptual-model programming from the other side. Model suite development and deployment will be based on separation of concern into intrinsic and extrinsic parts of models. Models typically consist on the one side of a normal model that displays all obviously relevant and

important aspects of a model and on the other side of a deep model that intrinsically reflects commonly accepted intentions, the accepted understanding, the context, the background that is commonly accepted, and restrictions for the model. The model suite will be layered into models for initialization, for strategic setup, for tactic definition, for operational adaptation, and for model delivery (see Fig. 3).
Model development can be layered in a form that is similar to the onion structure in Figures 1 and 2. We use essentially five layers for true fifth generation programming as shown in Figure 3:

(1) an internal layer for general initialization,
(2) an application definition language layer that includes many additional library packages,
(3) the internal supporting and generated layer with its generic and reference libraries,
(4) the input model suite that reflects the application and which is essentially the main task for an application engineer, and
(5) the generic intermediate output layer, and its delivery layer for multiple output variants depending on the target programming language.

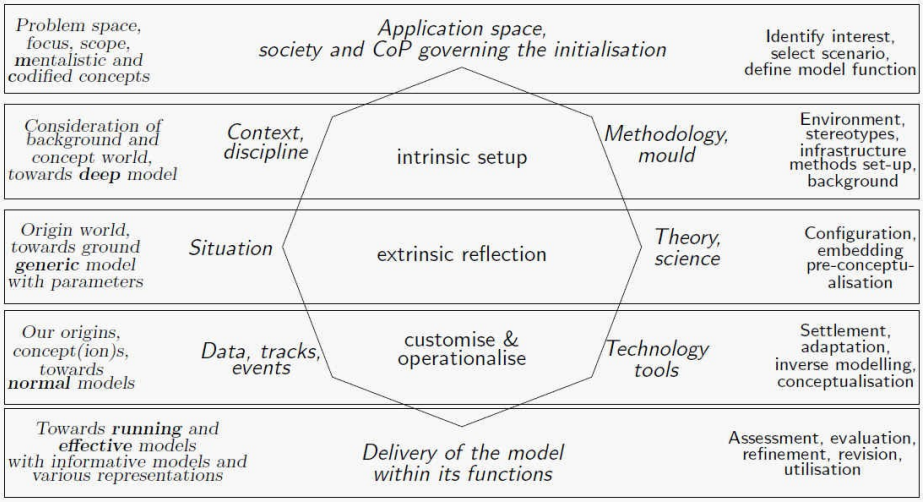

Fig. 3. The five layers to model development (initialize, setup, reflection, customize, delivery)

The model suite will be layered into models for initialization, strategic setup as an intrinsic setup, tactic definition as an extrinsic reflection, customization and operationalization as the main program development layer and as operational adaptation, and for model delivery. The complete model suite thus becomes the source for the code of the problem solution, and for the system to be built. Currently, a model is considered to be the final product. Models have their own background that is typically not given explicitly but intrinsically. Currently, methods for developing and utilizing models are accepted as to be given. The intrinsic part of a model and these methods form is called deep sub-model. The deep model is coupled with methodologies and

with moulds that govern how to develop and to utilize a model. The deep as well as the general model are starting points for developing the extrinsic or “normal’’ part of a model. Consideration of modelling is often only restricted to normal models similar to normal science. Model suites integrate however these model kinds. The main obstacle why model-driven development and of conceptual-model programming has not yet succeed is the non-consideration of the deep model and of modelling moulds.

4 Conclusion

We envision that true fifth generation programming can be based on development of high-level program descriptions that can be mapped to third-generation or fourth-generation programs. These programs may then be directly executed within the corresponding environment. This approach has already been the essential idea and its generalization behind a system for translation of domain-specific languages in the 80ies. The DEPOT-MS (DrEsdner PrOgrammTransformation) [7] was a compiler-compiler for domain-specific languages (historically: little languages, application-domain languages (Fachsprache)) that has been used to compile specific language programs to executable programs in the mediator language (first BESM6/ALGOL, later PASCAL, finally PL/1 [2]). The approach integrates the multi-language approach [1], the theory of attribute grammars [9], and theory of grammars [3,12].

A second source for true fifth generation programming is literate programming [5] that considered a central program together with satellite programs, especially for interfacing and documenting. This approach can be generalized and extended by new paradigms of programming (e.g. GibHub, 'holon' programming, schemata of cognitive semantics, and projects like the Axiom project or the mathematical problem solver [8] have already shown the real potential of literate programming. Our approach extends literate programming to model suites which are sets of models with well-specified and maintainable associations.

The developed framework, its theoretical underpinning and the realization approach is novel, targets at new programming styles, supports programmers from applications without requiring from them a deep program language knowledge and skills, and is going to overcome current limitations of programming. Layering is one of the great success stories in computer engineering. Already early languages such as COBOL used layered programs (division-section-paragraph-sentence-statement-command; ICCO: initialize-configuration-content_enhancement-operationalisation; environment-declaration-program). Our approach continues and generalizes this approach and will be thus the basis for true fifth generation programming.

A model in the model suite is used for different purposes such as communication, documentation, conceptualization, construction, analysis, design, explanation, and modernization. The model suite can be used as a program of next generation and will be mapped to programs in host languages of fourth or third generation. Models will become programs of true fifth generation programming.

Models delivered include informative and representation models as well as the compilation of the model suite to programs in host languages. Models will thus be-

come executable while being as precise and accurate as appropriate for the given problem case, explainable and understandable to developers and users within their tasks and focus, changeable and adaptable at different layers, validateable and verifiable, and maintainable.

References

1. Ershov, A.P.: The transformational machine: Theme and variations. In Proc. *MFCS 1981*, LNCS 118, pp. 16-32. Springer (1981).
2. Grossmann, R., Hutschenreiter, J., Lampe, J., Lötzsch, J., and Mager, K.: DEPOT 2a Metasystem für die Analyse und Verarbeitung verbundener Fachsprachen. Technical Report 85, *Studientexte des WBZ MKR/Informationsverarbeitung der TU Dresden*, Dresden (In German) (1985).
3. Hutschenreiter J.: *Zur Pragmatik von Fachsprachen*. PhD thesis, Technische Universität Dresden, Sektion Mathematik (In German) (1986).
4. Karagiannis, D., Mayr, H.C., and Mylopoulos, J., editors: Domain-Specific Conceptual Modeling, Concepts, Methods and Tools. Springer (2016).
5. Knuth, D.E.: Literate programming. *Comput. J.*, 27(2) pp. 97-111 (1984).
6. Kramer, F.F.: *Ein allgemeiner Ansatz zur Metadaten-Verwaltung*. PhD thesis, Christian-Albrechts University of Kiel, Technical Faculty, Kiel (In German) (2018).
7. Lötzsch, J.: *Metasprachlich gestützte Verarbeitung ebener Fachsprachen*. Advanced PhD thesis (habilitation), Dresden University of Technology, Germany (In German) (1982).
8. Podkolsin, A.S.: *Computer-based modelling of solution processes for mathematical tasks*. ZPI at Mech-Mat MGU, Moscov (In Russian) (2001).
9. Riedewald, G. and Forbrig, P.: Software specification methods and attribute grammars. *Acta Cybern.*, 8(1):89—117 (1987).
10. Thalheim, B., Schewe, K.-D., Prinz, A., and Buchberger, B. editors: *Correct Software in Web Applications and Web Services*. Texts & Monographs in Symbolic Computation. Springer, Wien (2015).
11. Thalheim, B. and Nissen, I. editors: Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung. De Gruyter, Boston (In German) (2015).
12. Thalheim, B.: Theorie deterministischer kontextfreier Grammatiken. Diplomarbeit, Technische Universität Dresden, Sektion Mathematik (In German) (1975).
13. Thalheim, B.: Entity-relationship modeling - Foundations of database technology. Springer, Berlin (2000).
14. Wirth, N.: *Compiler construction*. International computer science series. Addison-Wesley (1996).
15. Werthner, H. and Van Harmelen, F., editors: Informatics in the Future: Proceedings of the 11th European Computer Science Summit (ECSS 2015), Vienna, October 2015. Springer (2017).