Softw Syst Model (2011) 10:441–446
DOI 10.1007/s10270-011-0207-y

EXPERT'S VOICE

UML formal semantics: lessons learned

Manfred Broy · María Victoria Cengarle

Received: 20 May 2011 / Accepted: 23 May 2011 / Published online: 23 June 2011

Abstract The article below presents the insights gained during a number of years of research dedicated to the formalisation of the Unified Modeling Language.

Keywords Formal semantics · Compositional semantics · Multiple system views · All-encompassing UML semantics · Formal model-driven system development

1 Introduction

The Unified Modeling Language (UML [1]) is a general-purpose modelling language, created and managed by the Object Management Group (OMG). UML offers elements of a graphical concrete syntax to create visual models of software-intensive systems. UML synthesised notations of the Booch method [2], the Object-Modeling Technique (OMT [3]) and Object-Oriented Software Engineering (OOSE [4]) by fusing them into a single, common and widely usable modelling language; several other methods have then further influenced the UML, as for instance the Statecharts [5] and the Object Constraint Language (OCL [6]). UML aims to be a standard modelling language for concurrent and distributed systems. Though UML is not a development method by itself, it was designed to be compatible with the leading object-oriented software development methods of its time, for example the already mentioned OMT and Booch method as well as Objectory [7]. Development methods have been created based on UML, the best known is the Rational Unified Process (RUP [8]) of IBM. To provide more specific solutions, or achieve different objectives, further UML-based methods have been designed such as, e.g., the Dynamic Systems Development Method [9].

The UML rapidly gained popularity in industry, as it facilitated the communication between most diverse stakeholders about a system at different phases of development, on one hand, and, on the other, several points of view onto that system, including its context.

In contrast to its popularity, there are severe reservations about the UML in science and various domains of application. The most fundamental criticism of UML made by the models-based engineering community has been its lack of informal and formal semantics, i.e., the colloquial nature of UML's semantic description in natural language or in terms of UML itself; see [10,11]. Therefore, as pointed out in [12], multiple and potentially contradictory interpretations of one and the same model are not excluded, and automatic interpretation must be hard coded in some way or another in the tool chain. Regarding the former issue, in [13] it is claimed that perception and interpretation of reality is strongly shaped by our mental, cognitive concepts and structures. In [14], it is observed that modellers typically ignore any UML semantics and invent their own; in [15] it is moreover emphasised that a "cognitive" semantics of the UML, as opposed to an objectivist one, ultimately has its origin in political rather than objective considerations. Another matter is whether object-oriented programming concepts provide the right modelling concepts.

Communicated by Bernhard Rumpe.

The insights reflected in this article are the corollary of the rUML project financed by the German Research Foundation (DFG).

M. Broy (✉) · M. V. Cengarle (✉)
Software and Systems Engineering,
Technische Universität München, Munich, Germany
e-mail: broy@in.tum.de

M. V. Cengarle
e-mail: cengarle@in.turn.de

However, according to [16], the UML specification is *highly technical, terse, and very difficult for beginners to understand*, and the question of [17] *Why are some specifications so hard to read?* is therein puzzlingly answered as follows: *Most OMG specifications are written for programmers who implement compliant software products.* Needless to say, an informal description unavoidably involves ambiguities and lacks rigour, and thus precludes the reasoning, early simulation and automated analysis of a system design expressed by a UML model. This issue was speedily tackled by the research community as soon as the potential of UML as a de facto standard was recognised.[1] While (a large portion of) the static semantics of UML seems to have promptly reached a consensus,[2] the dynamic semantics of the UML sub-languages, such as activities, interactions and state machines, poses a major challenge. No reliable quality assurance can be undertaken as long as the very language employed to describe the system under scrutiny is deprived of a precise semantics. One consequence of these open issues is the low level of automation in tool support for the UML.

2 UML sublanguages

The UML sublanguages are complicated as many constructs and features have been put at disposal without the necessary fine tuning with each other. The Object Constraint Language (OCL [18]), for example, is used to formulate well-formedness rules in UML models, and is also intended to be an adjunct to UML for modellers who wish to add to their models more precision that cannot be otherwise (e.g., graphically) stated. OCL can be used as a navigation language, to write class invariants and pre-/post-conditions of methods within classes of a class diagram, guards of transitions of state machines, and guards of communication between instances in a sequence diagrams, among other uses. OCL offers (a) higher-order constructs as, e.g., `iterate`, (b) non-determinism, (c) non-termination, and (d) a three-valued logic with truth values `true`, `false`, and `undef`. These features necessarily make a semantics for OCL intricate; see [19], where it is moreover demonstrated that the changes introduced in OCL 2.x restrict its expressivity to primitive recursion while OCL 1.4/5 was computationally complete. In the end, the decision in favour of a three-valued logic in OCL is not state of the art.

The UML interactions describe possible message exchanges between system instances, and make up a powerful language which, besides integrating the standard operations like sequential, parallel, and iterative composition of interactions, provide means to specify recursive behaviour and negative behaviour, i.e., behaviour forbidden in system implementations. Previous versions of the interaction language did not provide for the specification of forbidden scenarios, what had been considered a fundamental flaw. However, it is just the operator for negation, or more precisely its meaning informally stated, that provoked more than one interpretation. Three possibilities are listed in [20], namely the "loose," the "strict" and the "flip" variants, the latter being the chosen one although admittedly the "strict" interpretation seems to be the closest to the intended meaning of [1].[3] These three interpretations of interaction negation are based on manipulations of sets of event traces, such that the universe of possible traces is partitioned into "valid," "invalid" and "contingent" traces. In [21], it is argued that the universe needs be partitioned in four kinds of event traces, for there exist traces that are simultaneously valid and invalid for an interaction expression; such an expression is therefore called "overspecified." The usefulness of a construct for the description of negative behaviour is emphasised in [22], where the focus is on secure systems and negative traces are those for which a negative (possibly non-continuous) subtrace exists. A further interpretation for negation is given in [23], which treats negation as a modality rather than an operator. UML interactions also put an operator for assertion at disposal, whose meaning is closely related to that of negation; the semantic discussion in relation to negation also applies to assertion.

Concepts semantically difficult to treat are those of `AND`- and `OR`-states of state machines and interlevel transitions in combination with the mechanism of deep and shallow history; see, e.g., [24,25]. As an illustration on how possibly desirable features may be obstructive, the combination of interlevel transitions, state references and the history mechanism prevents the definition of a compositional semantics for statecharts; see [26] and the references therein. As a matter of fact, these three features are still available in the UML syntax for state machines. Connection point references of UML state machines (see [1, p. 544ff]) are the present mechanism allowing interlevel transitions. The history mechanism, in its variants deep and shallow, is also a feature of UML state machines; see [1, p. 556ff]. State references can be made within the guard constraint (i.e., precondition) of a state transition as detailed in [1, p. 591]. Compositionality amounts to the meaning of a complex expression being determined by the meanings of its constituent expressions and

[1] Meanwhile UML is a de jure international standard, managed by the OMG; see also ISO/IEC 19501:2005 Information technology—Open Distributed Processing—Unified Modeling Language (UML) Version 1.4.2.

[2] Some static issues as, e.g., links are still subject of debate.

[3] The version of the UML referred to in [20] is older than [1], but it is still true that both the "loose" and the "flip" interpretations of interaction negation do not correspond to the intended meaning.

the rules used to combine them. But not only the respective meanings of the parts, depending on the composition operator used, imply the meaning of the whole; also properties locally proved of the parts may be inferred of the whole, thus compositionality provides for modular verification moreover accompanied by stepwise refinement.

The examples above let us identify the following problems. On the one hand, a vague description of the UML sublanguages allows for a "free" interpretation of them (e.g., negation of interactions). This is in line with the observation of [15] cited above, that modellers typically ignore any normative UML semantics and concoct a one which is considered correct in general or convenient for their own purposes. On the other hand, the overabundance of features, termed eclecticism in [27], results in languages that are technically difficult to treat (e.g., OCL). It can moreover lead to a definition that contradicts common sense and possibly the intention (e.g., state machines vs. compositionality), or dualities of intent (e.g., interactions switching between a dataflow and a method invocation perspective).

These foundational results show how important it is to carefully design a modelling language to avoid problems regarding its expressivity as well as its interpretation, problems which strongly impact on practical applicability. Much of the complexity of UML arises from lack of insight into experiences collected over years of research concerning those features.

3 UML as a whole

Perforce, complicated languages combined give rise to a very complicated modelling language when it comes to their integration and the definition of an integrative semantics. Aware of this situation, the OMG made a request for proposal in order to enable a chain of tools that support the construction, verification, translation, and execution of computationally complete executable models expressed in a *subset* of UML; see [28].[4] The current draft version of the official adopted specification is called Foundational UML (fUML [29]). The semantics of fUML is defined in fUML itself and using a base UML (bUML) in order to break circularity. A base semantics provides an interpretation of bUML as a set of first-order axioms over possible execution traces; the base semantics specifies when particular executions conform to a model defined in bUML, it does not generate executions. The subset addressed by this proposal most noteworthy disregards state machines and interactions (i.e., sequence diagrams).

Nevertheless, an all-encompassing formal semantics for the UML can be devised. We have approached this undertaking from two sides. On the one hand, a semantic core denominated the *System Model theory* was defined. This theory defines transition systems that, though complex, constitute a semantic domain suitable for sophisticated modelling languages and not just for UML 2. In general, the System Model forms a foundation for structure, behaviour and interaction of object-oriented, possibly distributed systems. The System Model is abstract enough to be of general value, but also sufficiently detailed to allow simulation and execution of UML models. Among others, central concepts of the UML have been formalized as theories of the "System Model;" see [30–33].

On the other hand, a heterogeneous approach to the semantics of UML is proposed where the semantics of each sublanguage is described in its natural mathematical domain (e.g., sequence diagrams define sets of positive, negative, and inconclusive traces), and where the relations between diagram types are expressed by appropriate translations. More formally, the UML family of sublanguages is represented as a "heterogeneous institution environment;" see [34]. Roughly speaking, an institution is a logical framework for the definition of theories (representing system designs) in which valid theorems implied by a theory (representing properties of a system design) remain valid even after theories are variously combined; an heterogeneous institution environment allows moreover the combination of theories defined over *different* logical frameworks. The advantage of this heterogeneous approach is that properties verified locally have global validity, a sine qua non for compositional verification; however, not for all of the UML sublanguages an institution has been devised.

What these efforts inevitably left untreated is the fact that a particular language may have different intended semantics when used at different stages of development. Indeed, on the one hand, there are concepts (e.g., transition systems) and description techniques (e.g., state machines) and, on the other, there are the stages of development. For instance, interactions can be used both for analysis purposes, to produce sample scenarios, as well as for integration test, to define test sequences. But these interaction specifications, at different stages, may be devised with a different intended semantics in mind: during integration test, they are (at least desirably) exhaustive, whereas during analysis they not necessarily are striven for exhaustiveness. Indeed, as pointed out in [27], the claim of *universality* of UML, meaning that UML is a notation suitable for analysis, design and documenting the implementation, necessarily entails multiple interpretations. This claim, put in perspective, could be considered an a priori reason for the failure of the UML as an Esperanto for modelling real world as well as systems in different stages of development. The conjecture is whether a formal semantic foundation (one or more approaches, depending on the development phase) defined beforehand for each sublanguage would allow a formal, universal and integrative at once, modelling language.

[4] Emphasis added.

4 UML in practice

The UML acceptance in industry seems circumscribed at its use, regardless of its semantics, in very early and/or very informal phases of development, mainly for the description of structure and seldom for the specification of behaviour. As surveyed in [14], the reasons for this situation are that the notations provided by the UML for describing behaviour are complex, poorly defined and poorly integrated, that round-tripping between code and model is far too lossy and error prone, and that tools in general are poor in how they integrate modelling artefacts into the lifecycle. Interestingly, [27] argues that a universal notation fosters naïve seamless development since, as the Gestalt theory postulates, initial concepts obtained by perception radically affect how subsequent constructs are formed; there moreover is shown how this fact unfavourably influences development, leading to overly coupled system with poor modular structure.

There have been efforts within academia, besides those at equipping UML with a formal semantics, that attempted at alternative but in spirit equivalent definitions of a modelling language, which are devoid of those handicaps copiously itemised in the literature. The proposal of [35], for instance, after showing deficits of UML as, e.g., circular and contradictory definitions, introduces an option that has an internally consistent structure supported by Russell's theory of types and defines a declarative semantics à la Tarski; moreover, the choice is justified on the basis of philosophical and natural science foundations, in contrast to UML which is a result of tries, failures and successes that were never theoretically justified.

However, those endeavours at either a formal semantics for the UML or at flawless alternatives equivalent to UML in intent, provoked desultory repercussion. First of all, the plethora of proposals result in a combinatoric explosion of possible combinations, of which many make simply no sense. The formal exploration of those accomplishments appear to be of no interest to the tool vendors possibly for political reasons. End-users, on their side, systematically disregard any formal semantics for UML because of the accompanying intrinsic complexity and lack of proper (i.e., at industrial scale) support. Unsurprisingly, the reaction from academia, besides questioning model-driven development methods for which UML was thought as a vehicle (see [36,37]), is a shift of focus to other challenges.

5 The bottom line

One of the pioneers of modelling languages, Doug Ross, developed the Structured Analysis and Design Technique (SADT [38]) far ahead of his time. In his old days, Doug spent most of his professional energy dealing with these complex ideas, giving meaning to and extending his modelling concepts to a complete *Technology for Understanding*. He moreover pursued the foundations for all of his technical work since the 1950s—a scientific philosophy called Plex. At a meeting of the IFIP Working Group 2.3 on program methodology held in Pouilly-en-Auxois, France, in September 1991, the group members had an intensive discussion with him about a formal model for SADT. Doug disapproved these attempts; literally, he said "You try to spoil my approach." He had the feeling that striving to give formal meaning to his graphical formalisms would ruin the whole proposal.

What do we learn from that? On the one hand, graphical notations are undoubtedly very useful, even when they are only informal. They allow us to sketch ideas in early stages of development, when many details need not be considered. Had those notations a formal meaning, many of the graphical descriptions would be actually false at least with respect to details in which nobody is interested at that stage of development. Perhaps, this might be an argument for *not* having a precise formal meaning.

On the other hand, a purely graphical notation with possibly many different semantic interpretations is not a good vehicle to support communication, in particular among engineers. What is needed is a robust set of concepts with unambiguous meaning, clearly directed towards the concepts used in engineering systems.

However, this characterisation does not apply to the UML. In many respects, UML is far too much triggered by specific concepts of programming like object-oriented constructs, which are overly close to a explicit execution model with some express operational meaning. The UML sublanguages are not properly related to each other and integrated, such that many semantic questions arise that are difficult to solve. Finally, many of the ideas expressed informally in the UML do not call for a straightforward formalisation. In the end, some quite unbaked ideas can be found in the UML. Moreover, some decisions like a three-valued logic for the OCL are certainly not state-of-the-art.

So, was it a failure to formalise the UML? We do not think so! We believe that an attempt at a formalisation is a scientific approach that promotes a deep understanding of the good as well as the bad aspects of the modelling language scrutinised. Trying to do the formalisation uncovered a lot of properties of the UML and led to considerably many fruitful questionings.

A final conclusion could be as follows: If one is interested in a formal modelling language, which supports the seamless development process including round-trip engineering and code generation with a great deal of automation by means of tools, a formal semantics is indispensable. Such a modelling language is thus close to a high-level design language and also to a programming language. For that

modelling language, all formal semantic questions have to be solved in order to achieve the essential goal of portability and independence of particular implementations of tools. A completely different goal, in accordance with the above mentioned Gestalt theory, is a language for sketching designs and grasping conceptional ideas. At early stages, less details are needed, just a simple structure is required with a basic, plain and informal meaning. Such languages are necessary, too. UML is none of both, unfortunately.

After all, the UML and the attempts to furnish it with a formal meaning, led to invaluable insights. We have learned how to do it better. We have understood that designing a full-blown modelling language as an engineering tool has to be done from foundational principles and not out of the amalgamation of a number of not completely understood concepts. Moreover, we have seen that we also need informal languages, and for them completely different principles have to be used.

Acknowledgments The authors would like to thank all colleagues who contributed to the development of the rUML project, in particular those from the Chair of Software Engineering in Aachen (formerly at the Chair of Software Systems Engineering in Brunswick). Bernhard Rumpe provided us with richly interesting feedback to previous draft versions of this account. Bran Selic pointed out many inaccuracies in particular regarding the history and evolution of the UML.

References

1. Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure—Version 2.3. Technical Report Document Number formal/2010-05-05, OMG. http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ (2010). Retrieved 2011-01-18
2. Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Connallen, J., Houston, K.A.: Object-oriented analysis and design with applications, 3rd edition. ACM SIGSOFT Software Engineering Notes, vol. 33 (2008)
3. Rumbaugh, J.E., Blaha, M.R., Premerlani, W.J., Eddy, F., Lorensen, W.E.: Object-Oriented Modeling and Design. Prentice-Hall, New Jersey (1991)
4. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Longman Publishing Co. Inc., Redwood City (2004)
5. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Progr. **8**, 231–274 (1987)
6. Warmer, J.B., Kleppe, A.G.: The Object Constraint Language: Precise Modeling With UML. Object Technology Series. Addison-Wesley Longman Publishing Co. Inc., Redwood City (1998)
7. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley Longman Publishing Co. Inc., Boston (1999)
8. Gornik, D.: IBM Rational Unified Process: best practices for software development teams. Technical Report TP026B, Rev 11/01, IBM. ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/rup_bestpractices.pdf (2004). Retrieved 2010-07-01
9. Stapleton, J.: DSDM: Dynamic Systems Development Method. In: 29th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Europe 1999), IEEE Computer Society, vol. 406 (1999)
10. Rumpe, B.: A note on semantics (with an Emphasis on UML). In: Kilov, H., Rumpe, B., (eds.) 2nd ECOOP Workshop on Precise Behavioral Semantics. Technical Report TUM-I9813, Institut für Informatik, pp. 177–197. Technische Universität München (1998)
11. Richters, M.: A Precise Approach to Validating UML Models and OCL Constraints. PhD thesis, Universität Bremen, Logos, Berlin, BISS Monographs, No. 14 (2002)
12. Cuccuru, A., Mraidha, C., Terrier, F., Gérard, S.: Enhancing UML Extensions with Operational Semantics. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., (eds.): Model Driven Engineering Languages and Systems (MoDELS'07, Proceedings). Lecture Notes in Computer Science, vol. 4735, pp. 271–285. Springer, Berlin (2007)
13. Evermann, J.: A cognitive semantics for the association construct. Requir. Eng. **13**, 167–186 (2008)
14. Cook, S.: UML2.0—Trying to have it both ways, pp. 4–7 of [39]
15. Cook, S.: UML Semantics. Steve Cook's WebLog. http://blogs.msdn.com/b/stevecook/archive/2004/12/08/278507.aspx (2004). Retrieved 2010-07-01
16. Object Management Group: Introduction to OMG's Unified Modeling Language (UML). Technical report, OMG. http://www.omg.org/gettingstarted/what_is_uml.htm (2009). Retrieved 2010-07-01
17. Object Management Group: Getting Specifications and Products. Technical report, OMG. http://www.omg.org/gettingstarted/specsandprods.htm (2009). Retrieved 2010-07-01
18. Object Management Group: Object Constraint Language. Technical Report Document Number formal/2010-02-01, OMG. http://www.omg.org/spec/OCL/2.2/PDF (2010). Retrieved 2011-01-18
19. Cengarle, M.V., Knapp, A.: OCL 1.4/1.5 vs. OCL 2.0 expressions: formal semantics and expressiveness. Softw. Syst. Model. **3**, 9–30 (2004)
20. Störrle, H.: Assert, Negate and Refinement in UML-2 Interactions. In: Jürjens, J., Rumpe, B., France, R., Fernandez, E.B. (eds.) 2nd International Workshop on Critical Systems Development with UML (CSDUML'03, Proceedings). Technical Report TUM-I0323, pp. 79–93. Institut für Informatik, Technische Universität München (2003)
21. Cengarle, M.V., Knapp, A.: UML 2.0 Interactions: Semantics and Refinement. In: Jürjens, J., Fernandez, E.B., France, R., Rumpe, B. (eds.) 3rd International Workshop on Critical Systems Development with UML (CSDUML'04, Proceedings). Technical Report TUM-I0415, pp. 85–99. Institut für Informatik, Technische Universität München (2004)
22. Seehusen, F.: Specifying enforcable high level policies with UML sequence diagrams. Telektronikk **105**, 126–134 (2009)
23. Harel, D., Maoz, S.: Assert and negate revisited: modal semantics for UML sequence diagrams. Softw. Syst. Model. **7**, 237–252 (2008)
24. von der Beeck, M.: A structured operational semantics for UML-statecharts. Softw. Syst. Model. **1**, 130–141 (2002)
25. Simons, A.J.H.: On the compositional properties of UML statechart diagrams. In: Rigorous Object-Oriented Methods (ROOM 2000, Proceedings). Workshops in Computing, BCS (2000)
26. von der Beeck, M.: A Comparison of Statecharts Variants. In: Langmaack, H., de Roever, W.P., Vytopil, J. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems (3rd FTRTFT, Proceedings). Lecture Notes in Computer Science, vol. 863, pp. 128–148. Springer, Berlin (1994)
27. Simons, A.J.H., Graham, I.: 30 Things that go wrong in object modelling with UML 1.3. In: Kilov, H., Rumpe, B., Simmonds, I. (eds.) Behavioral Specifications of Businesses and Systems, pp. 237–257. Kluwer Academic Publishers, Dordrecht, Chapter 17 (1999)
28. Object Management Group: Semantics of a Foundational Subset for Executable UML Models—Request For Proposal. Technical

Report Document Number ad/2005-04-02, OMG. http://www.omg.org/cgi-bin/doc?ad/05-04-02.pdf (2005). Retrieved 2011-01-18
29. Object Management Group: Semantics of a Foundational Subset for Executable UML Models (fUML), version 1.0. Technical Report Document Number formal/2011-02-01, OMG. http://www.omg.org/spec/FUML/1.0/PDF (2011). Retrieved 2011-01-18
30. Broy, M., Cengarle, M.V., Grönniger, H., Rumpe, B.: Considerations and Rationale for a UML System Model. In: Lano, K. (ed.) UML 2 Semantics and Applications, pp. 43–60. Wiley, Hoboken, Chapter 3 (2009)
31. Broy, M., Cengarle, M.V., Grönniger, H., Rumpe, B.: Definition of the System Model. In Lano, K. (ed.) UML 2 Semantics and Applications, pp.61–93. Wiley, Hoboken, Chapter 4 (2009)
32. Cengarle, M.V., Dingel, J., Grönniger, H., Rumpe, B.: System-model-based simulation of the UML. In: Nordic Workshop on Model Driven Engineering (5th NW-MoDE 2007, Proceedings), pp. 112–126. Blekinge Tekniska Högskola, Research Report 2007:8. ISSN 978-91-7295-985-9. http://www.sse-tubs.de/publications/CDGR07NWMODE.pdf (2007). Retrieved 2011-06-14
33. Crane, M.L., Dingel, J.: Towards a formal account of a foundational subset for executable UML models. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) 11th International Conference Model Driven Engineering Languages and Systems (MoDELS'08, Proceedings). Lecture Notes in Computer Science, vol. 5301, pp. 675–689. Springer, Berlin (2008)
34. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A Heterogeneous Approach to UML Semantics. In: Degano, P., Nicola, R.D., Meseguer, J. (eds.) Concurrency, graphs and models: essays dedicated to Ugo Montanari on the Occasion of His 65th Birthday. Lecture Notes in Computer Science, vol. 5065, pp. 383–402. Springer, Berlin (2008)
35. Naumenko, A., Wegmann, A.: Triune continuum paradigm and problems of UML semantics. Technical Report IC/2003/44, Swiss Federal Institute of Technology, Lausanne, Switzerland. http://www.triunecontinuum.com/documents/tr03_044.pdf (2003). Retrieved 2010-07-16
36. Picek, R., Strahonja, V.: Model driven development—future or failure of software development? In: 18th International Conference on Information and Intelligent Systems (Proceedings), Faculty of Organization and Informatics, pp. 407–413. Varaždin (2007)
37. Kapteijns, T., Jansen, S., Brinkkemper, S., Houët, H., Barendse, R.: A comparative case study of model-driven development vs traditional development: the tortoise or the hare. In: Bailey, T., Vogel, R., Mansell, J. (eds.) From code centric to model centric software engineering: Practices, Implications and ROI (4th European C2M Workshop, CTIT Proceedings), pp. 22–33. University of Twente (2009)
38. Ross, D.T.: Applications and extensions of SADT. IEEE Comput. **18**, 25–34 (1985)
39. Henderson-Sellers, B.: UML—the Good, the Bad or the Ugly? Perspectives from a panel of experts. Softw. Syst. Model. **4**(1), 4–13 (2005). doi:10.1007/s10270-004-0076-8

Author Biographies

Manfred Broy is a full professor of computer science at the Technische Universität München. His research interests are software and systems engineering, comprising both theoretical and practical aspects. This includes system models, specification and refinement of system components, specification techniques, development methods and verification. Manfred Broy is leading a research group, working in a number of industrial projects that try to apply mathematically based techniques and to combine practical approaches to software engineering with mathematical rigour. The main topics are ad hoc networks, software architectures, componentware, software development processes and graphical description techniques. In his group, the CASE tool AutoFocus was developed. Throughout his academic career, Manfred Broy has maintained strong contacts with industry, through consultancy, teaching and collaborative research projects. He has published more than 330 scientific publications. His main field is software and systems engineering and his current research interests are system development processes and tool support, system modelling, concurrent and embedded systems, theoretical foundation of informatics, quality and requirements engineering.

María Victoria Cengarle studied computer science at the Universidad de Buenos Aires and the Escuela Superior Latinoamericana de Informática (ESLAI), Argentina. She worked for the United Nations Development Program as a consultant and simultaneously as assistant lecturer at the Computer Science Department of the Universidad de Buenos Aires. At the Ludwig-Maximilians-Universität München, Germany, she cooperated in the DIN-Lisp standardization process and participated in interchanges with the Monash University, Melbourne, and the Pontifícia Univeridade Católica do Rio de Janeiro. She defended her PhD thesis entitled "Formal Specifications with Higher-Order Parameterization". Since then, she has worked in applied research at the Bayerische Landesbank and knowledge transfer at the Fraunhofer Institut Experimentelles Software Engineering, given lectures at the Fachhochschule München, been research assistant at the chair of Software and Systems Engineering of the Technische Universität München, and is now research fellow at Fortiss. Her research interests include foundations of computer science, formal systems development as well as semantics of specification and programming languages.