Scandinavian Contributions to Object-Oriented Modeling Languages

Birger Møller-Pedersen

Scandinavian Contributions to Object-Oriented Modeling Languages

Birger Møller-Pedersen

Department of Informatics, University of Oslo, Norway
birger@ifi.uio.no

Abstract. The history of Scandinavian contributions to modeling languages is interesting in many respects. The most interesting part of the history is that some of mechanisms were conceived very early, years before modeling became mainstream. It is well-known that object-orientation started with SIMULA in 1967, but it is less known that SIMULA formed the basis for a modeling language already in 1973, and that the Ericsson AXE software structure (1976) was one of the foundations (via SDL) for composite structures in UML2. It is also interesting that there has been a development towards making mechanisms less particular: while early modeling languages had special structuring mechanisms, UML2 now cover this by composite classes. In addition, early modeling languages were executable, in fact they were combined modeling – and programming languages. After a period where modeling was just for the purpose of analysis and design, the trend is now towards executable models, i.e. almost going back to the original Scandinavian approach.

Keywords: Languages, modeling, programming

1 Introduction

This paper tells part of the story of the Scandinavian contribution to modeling languages. We restrict ourselves to the contribution to the de facto standard language for modeling, UML in its current version UML2 [1]. Instead of just making a chronological account of what happened when and who were involved, the story is told with emphasis on language mechanisms, and the paper is organized correspondingly.

For the account of when things happened, we have decided to do it based upon published material, although we are aware that activity often started earlier, see Fig 1.

2 Object-Orientation

Object-orientation as part of UML has a straightforward history. The main sources are the Booch [2] and OMT [3] methods, and these are in turn based upon the concepts of object-oriented programming originating with SIMULA. Booch and OMT were not

the first methods for object-oriented modeling, but they directly influenced UML. The OOSE method [4] contributed with Use Cases.

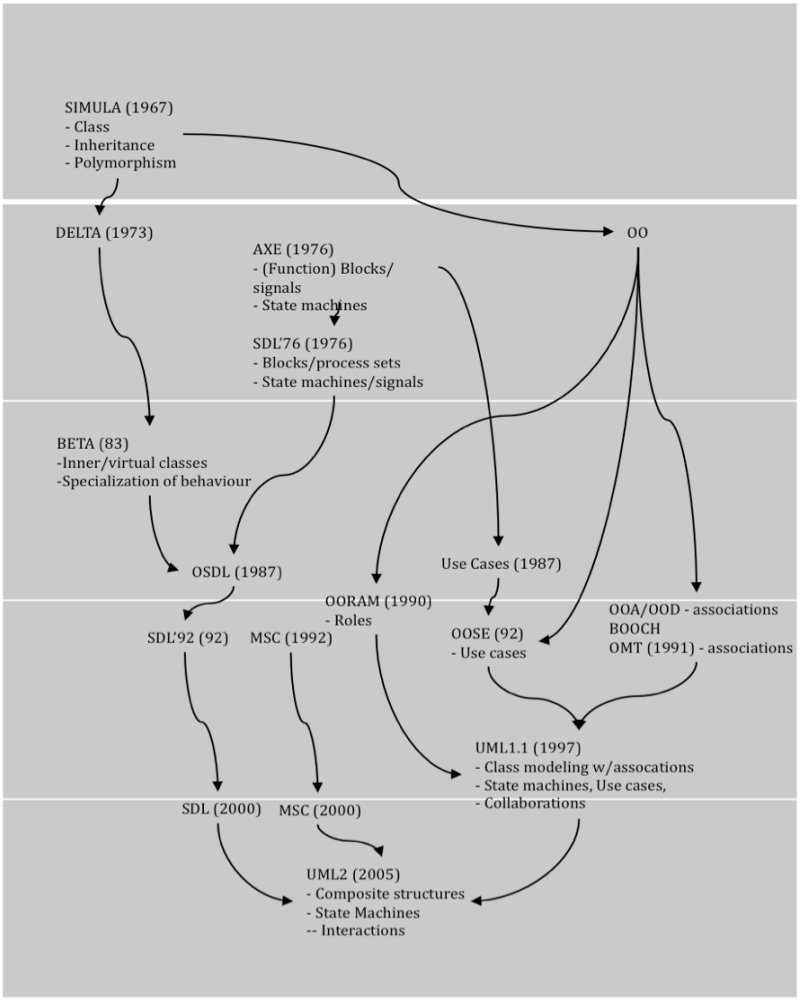

Fig. 1. Contributions to UML2.

SIMULA was developed by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Centre, Oslo, in the 1960s [5, 6]. A comprehensive history may be found in [7] and in [8].

SIMULA I was a simulation language. It was realized that the concepts in could be applied to programming in general, and the result was the general-purpose language SIMULA 67 – later on just called SIMULA. The SIMULA I language report of 1965 opens with these sentences:

The two main objectives of the SIMULA language are:

- To provide a language for a precise and standardized description of a wide class of phenomena, belonging to what we may call "discrete event systems".
- To provide a programming language for an easy generation of simulation programs for "discrete event systems".

Thus, SIMULA was considered as a language both for system description and for programming.

It turned out that many users of SIMULA seemed to get more understanding of their problem domain by developing a description than from the actual simulation results. The modeling approach of SIMULA was therefore refined in **DELTA** [9], a pure description language ('description language' was the term for 'modeling language' at that time).

DELTA supported the description of true concurrent objects and used predicates to express state changes and continuous changes over time. DELTA had a goal similar to that of the object-oriented analysis and design (OOA/OOD) methodologies (e.g. Coad and Yourdon [10]) that appeared subsequently in the mid-80s, followed by OMT and Booch.

After the initial ideas of SIMULA, there were two different developments of object-oriented programming: the 'modeling approach' and the 'programming approach'. The main difference between these is on the understanding of objects, classes and subclasses. The modeling approach regards objects as models of phenomena from the application domain, classes as models of domain concepts and subclasses as models of special concepts. SIMULA defined the modeling approach. The programming approach puts emphasis on subclassing as a language mechanism for reuse of code. A subclass does not have to define a subtype of the type of the superclass, and the subclass is free to redefine properties of the superclass. Object-oriented methods in the 1980s followed the modeling approach.

The approach to language design used for **BETA**[1] [11] was highly influenced by the SIMULA tradition. BETA should be a combined modeling and programming language. The first account of BETA is [11]. The history is described in [12].

It was realized already with SIMULA that the class/subclass mechanism was useful for representing concepts including specialization, but there was no formulation of a conceptual framework for object-oriented programming. For BETA it was therefore necessary do develop such a conceptual framework. [13] provides a definition of object-oriented programming with a modeling approach.

It was from the beginning decided that BETA should support for full concurrency. As will be seen below, this early Scandinavian work on object-orientation had similarities with the AXE software structure in that it supported concurrent, active and communicating objects.

The primary goal of the OMT++ [14], OMT+ [15], and Octopus [16] efforts by Nokia Research Center was improved methods for embedded systems, and not new language mechanisms.

---

[1] The BETA-project started as a research project (Bent Bruun Kristensen, Ole Lehrmann Madsen, Birger Møller-Pedersen, and Kristen Nygaard). The main implementation effort was done as part of the Nordic Mjølner project.

3 Structuring Mechanisms

3.1 Blocks and Block Structure

In the telecom industry, specifying systems as a set of interconnected blocks was common practice already in the seventies as reported in [17]. SDL was one of the first standardized notations for specification with support for such structuring mechanisms [18].

AXE Blocks. In 1976 the paper [19][2] in Ericsson Review presented for the first time the software structure of the AXE system. This was a result of some years of applying software technology to the area of switching, and in line with most similar efforts, it was based upon non-synchronized communication (by means of signals) between processes. The new thing was the organization of the software in terms of blocks with well-defined interfaces.

The experience with the AXE system brought into development the SDL'76 and SDL'80 by Ivar Jacobson, Nils Lennmarker, and Anders Rockström [3]. SDL used blocks with possible substructures of blocks and concurrent, non-synchronized processes with behavior defined in terms of state machines and with well-defined interfaces in terms of signals.

SDL'84 and SDL'88 saw the formalization of both the communication part and the data modeling part [20][21]. The reason was that different tools then would have the same interpretation, and that SDL could be used not only for documentation, but also for making executable specifications.

While **block structure** in AXE and SDL was a direct specification of what an execution consists of in terms of instances (composition), block structure in programming languages means pure *nesting* of definitions. Early programming languages had procedures/functions nested within procedures/functions. SIMULA introduced nesting of classes. From being a purely technical mechanism, nesting ended up being conceived as a means (in [22] coined *localization*) for describing concepts that depend on an enclosing object.

Nesting in terms of inner classes has made it into main object-oriented programming languages and into UML2.

Object-oriented SDL (OSDL/ SDL'92) was proposed in 1987 [23][4]; in 1992 these concepts became part of the language [24][5]. Object-oriented SDL introduced block

---

[2] The reason for one author on this paper is probably that Hemdahl was the key person behind this, although many persons were involved in this undertaking.

[3] Work on SDL started already in 1968, when CCITT decided to find out what to do about telecom and software technology. Telecom operators were the driving forces (e.g. the Swedish Televerket, Gösta Lindberg (CTO)). Anders Rockström joined this effort in 1974.

[4] Started as part of the Mjølner project (Dag Belsnes, Hans Petter Dahle, Birger Møller-Pedersen), where the idea was to develop a new language. After discussions, especially between EB Technology and the Swedish partners (where Anders Rockström and Ferenc Belina were involved), it was decided to go for an extension of SDL.

[5] The standardization work of OSDL was done as part of the Norwegian SISU-project, where Øystein Haugen and Rolf Bræk joined, and as part of the EU-projects SPECS and FORTE, where Ove Færgemand and Anders Olsen from TeleDanmark joined. Anders Ek was a key person involved from Telelogic.

types and process types, and gates as connection points for communication links between blocks and processes. In addition, specialization (subtyping) was defined for block and process types. For block structuring, the implication was that block subtypes would inherit the internal block structure of the super block type.

OOSE 1992. From the AXE software structure OOSE had blocks with internal structure, and each block had an own behavior in terms of a state machine.

ADL/ROOM. Work on Architecture Description Languages (ADLs) began in the early nineties. This area has produced a number of modeling languages. In 1994, ROOM [25] combined statecharts and structuring mechanisms like those of SDL (by capsules, ports and connectors).

Subsystems. When UML1[6] was adopted in 1997, none of these structuring mechanisms had become part of the language. It had no notion of block (AXE/SDL/OOSE) or capsule (ROOM); an object could only have behavior in terms of methods and not a main behavior as an active object (process).

UML1 had, however, the notion of Subsystem, being a mixture of a Classifier and a Package. As a package, it could contain (among other things) a State Machine, but this was nothing like the behavior of a block. In addition, there are no communication links between subsystems. The paper [26] gives a description of subsystems.

ADLs had Components as the parts of a structure. For some reason Components in UML1 was not a mechanism for structuring systems, but more a mechanism for deployment. This has, however, been changed in UML2.

3.2 Composite Classes

When OMG in 1999 embarked upon making the second version of UML (UML2), a group of telecom users (Alcatel, Ericsson, and Motorola) of SDL handed in joint requirements [27]. Structuring of large, complex systems was one of the main requirements.[7]

SDL had block diagrams with the only purpose of specifying the architecture of systems ([28] and [29]). In contrast to this, UML1 had class diagrams with classes and relations between classes. Although composition was one of these relations, this could not be used for the structuring of systems, with the implication that UML users had to use packages and subsystems for this purpose, see [30].

The 1992 version of SDL supported object orientation by block types and process types, and corresponding subtypes. These types would correspond to classes in UML (when it arrived in 1997), so the obvious requirement for UML2 was that classes should be able to have an internal structures.

ROOM capsules had much of the same properties as SDL'92 blocks. In 1999 the ROOM concepts became parts of a UML profile UML-RT [31]. Fortunately the

---

[6] UML1.1, the first version of UML with Scandinavian contribution. The Swedish company Objectory, founded on the OOSE method in 1987, was acquired by Rational in 1995 and contributed with Use Cases and predefined method stereotypes. UML versions up to 1.5 – Karin Palmkvist and Gunnar Övergård contributed to Use Cases, Interactions, Collaborations, and Subsystems.

[7] Scandinavian contributors to UML2: Øystein Haugen and Birger Møller-Pedersen (at that time with Ericsson).

UML2 saw the SDL2'92 and ROOM concepts as structuring concepts within UML proper and not in a profile. As pointed out in [32], the way in which UML-RT combined UML with ROOM was not the way to do it; they advocated what more or less became the reality: that structuring mechanisms became integrated with class models.

The discussions described above lead to the notion of composite structures and in turn, these were used to define composite classes and collaborations. A composite structure consists of parts (sets of objects of a given type) connected by connectors. Connectors either connect parts directly or connect ports on parts. The composite structures of collaborations also form the contexts for Interactions, so these parts are well integrated in UML2. This is illustrated in [33].

The following citation from [34] is summing up structuring mechanisms of UML2:

> The basis for this set of features comes from long-term experience with various architectural description languages, such as UML-RT, ACME, and SDL. These languages are characterized by a relatively simple set of graph-like concepts: basic structural nodes called parts that may have one or more interaction points called ports and that are interconnected by communication channels called connectors.

A detailed correspondence between SDL and UML2 is described in [35].

Languages with support for modeling of architectures typically have language mechanisms designed especially for this purpose. One may easily imagine that the same thing could have happened when introducing this in UML2, i.e. making up a new language mechanism just for this purpose. Fortunately, both SDL and ROOM had been there before. In UML2, therefore, it was recognized that the composite structures, which were needed in order to model architectures, could be seen as a special case of composition of classes. This implied that composite structure became an inherent property of objects, and by being defined for classes, inheritance also applied to structure: inheritance implied inheritance of parts, ports and connectors.

4 Object Behavior

Mainstream object-oriented languages have most of the original concepts from SIMULA: classes and objects, subclasses, virtual methods, polymorphism, etc. An exception is the SIMULA notion of an active object with its own action sequence (in addition to behavior in terms of methods). Strangely enough, many other programming languages had not adopted this.

In SIMULA, it was regarded as obvious that objects had their own behavior, in addition to behavior associated with methods. Objects were not just intended for the modeling of data entities, but also for processes with their own behaviors.

The AXE design had similarly function blocks with their own behavior (in terms of a state machine), the same with processes in SDL'76. OOSE did not carry this over from AXE and SDL.

UML1 did not have a separate object behavior, although it had the notion of active objects. Booch and OMT followed the trend in programming languages to do this by means of a special method.

UML2 introduced a so-called ClassifierBehavior, so that classes may have a behavior. This extension comes from SDL, ROOM, and UML-RT, and it links back to the original AXE and SDL idea of processes as the main kind of objects.

Although **state machines** were part of AXE, this was not the only source of inspiration for state machines in SDL'76. The use of state machines was common in telecom and process control systems. UML1 had state machines, but they were for some reason intended for the specification of the behavior of collaborations, and not for the objects taking part in the collaborations, at least according to the semantics specification of UML1. The state machines of UML1 comes from [36], where objects and state machines were combined. In 1987, Harel introduced structuring mechanisms for state machines, hierarchical statecharts, with composite states containing states and transitions.

SDL-2000 adopted the hierarchical state machines of statecharts, extending them with entry/exit points [37], a kind of interface for composite states, so that states could be entered/exited without having to know to the details of the composite states. Entry/exit points for states are supported by UML2.

5 Use Cases

The story behind Use Cases is easy to track down. The first paper on Use Cases is from OOPSLA'87 [38], and this paper in turn points to two sources of inspiration: the way functions (we would call it services today) were described as part of the AXE development within Ericsson, and the IFIP'86 paper [39], where the notions of task and roles in system development were introduced. Use Cases also rest on the shoulders of both the Norwegian and Swedish approaches to participatory design.

Use Cases are sometimes criticized for leading to functional decomposition of systems instead of the object-oriented decomposition based upon objects. However, use cases were conceived in a setting where the system design was made by means of blocks, so use cases were the means by which it was possible to describe properties of the system independently of how it should be structured in terms of blocks.

Use Cases have been a success independent of UML and they are not really dependent on being tightly integrated with the rest of the UML.

6 Collaborations

Collaborations have two Scandinavian contributions: role modeling and realizations of use cases.

The role modeling method, OORAM[8] [40] is a method that emphasizes objects and the collaborations of objects playing different roles. It is covered in the survey [41], and it has been demonstrated at OOPSLA conferences since 1986. Collaborations in UML1 were based upon presentations of the method to the OMG.

[8] Based upon a former role modeling method [48]. Trygve Reenskaug was member of the UML Core Team, and part of small group that made the first proposal for UML1.1.

Realization of use cases by means of collaboration was presented at the UML conference in 1999 [42]. Collaborations were not part of OOSE, however, one of the co-authors on the book on OOSE contributed to the realization of use case by means of collaborations.

Collaborations were revised (in light of composite structures) as part of UML2, in a way that supports OORAM better than the Collaborations of UML1.

7 Interactions (Sequence Diagrams)

When SDL was standardized for the first time in 1976, the telecom field (including AXE) also used proprietary variations of sequence diagrams (called Message Sequence Charts – MSC). The 1988 SDL User Guidelines had a short section on message sequence charts as an auxiliary diagram that could be used in combination with SDL.

A paper [43] triggered the first standardization of Message Sequence Charts (MSC) in 1992[9]. MSC-2000 [44] refined earlier structuring mechanisms such as MSC references and inline expressions.

UML1 selected a somewhat different dialect similar to, but not identical to MSC-92, as the foundation of sequence diagrams, and did not support reuse and hierarchy.

The approach in [45] formed the basis for Interactions in UML2[10], and the two dialects have reached more or less the same expressiveness, see [46].

8 Conclusion: Executable Models or Unified Modeling and Programming

It all started out with a combined modeling and programming language (SIMULA), with software structuring (AXE) tightly integrated with a programming language, and a modeling language with an execution semantics (SDL). Although one of the main strengths of object-orientation is that it provides a unified approach to modeling and programming, modeling languages developed independently of object-oriented programming languages, and drifted away from execution and became languages in support for analysis and design. Recently we have witnessed the need for executable models, not just for special domain specific languages, but for modeling in general, and this has called for initiatives like Executable UML [47]. One may say that this is just a return to how it was in the beginning, but the difference is that executable models may still just translate to implementations in existing programming languages, yielding both a model and a program artifact. If development environments for modeling languages for executable models will not be able to match development environments for programming languages, then users still have to cope with both a modeling language and a programming language, and based upon experience the code

[9] A key contributor at the time and subsequently (until his untimely death in 2002) was Ekkart Rudolph of Siemens, Munich.

[10] Øystein Haugen (at that time with Ericsson) was leading this work for UML2.

artifact will be the dominant. Really returning to the original approach would imply going for a combined modeling -and programming language, see [48] for an account of that. Class models and state machine models will obviously be part of such a combined language, with e.g. interaction models for the specification of execution semantics, while other parts of modeling languages are obviously just intended as support for analysis methods, as with Use Cases.

Acknowledgments. Thanks to Anders Rockström, Rick Reed, Pär Emanuelsson, and Anders Olsen for providing valuable input, and to Stein Krogdahl for careful reading and commenting.

9 References

1. OMG: UML: Unified Modeling Language 2.0. OMG. ptc/2004-10-02 (2004)
2. Booch, G.: Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, Redwood City (1991)
3. Rumbaugh, J., et al.: Object-Oriented Modeling and Design. Prentice Hall (1991)
4. Jacobson, I., et al.: Object-oriented software engineering: a use case driven approach. Addison-Wesley Professional (1992)
5. Dahl, O.-J., Myhrhaug, B., Nygaard, K.: SIMULA 67 Common Base Language (Editions 1968, 1970, 1972, 1984). Norwegian Computing Center, Oslo (1968)
6. Dahl, O.-J., Nygaard, K.: SIMULA—a Language for Programming and Description of Discrete Event Systems. Norwegian Computing Center, Oslo (1965)
7. Dahl, O.-J., Nygaard, K.: The Development of the SIMULA Languages. In: ACM SIGPLAN History of Programming Languages Conference (1978)
8. Krogdahl, S.: The Birth of Simula. In: Bubenko, Jr., J., Impagliazzo, J., Sølvberg, A. (eds.) History of Nordic Computing. IFIP WG9.7 First Working Conference on the History of Nordic Computing (HiNC1), June 16–18, 2003, Trondheim, Norway. Springer, New York (2003)
9. Holbæk-Hanssen, E., Håndlykken, P., Nygaard, K.: System Description and the DELTA Language. Norwegian Computing Center, Oslo (1973)
10. Coad, P., Yourdon, E.: Object-Oriented Analysis. Prentice-Hall, Englewood Cliffs, N.J (1991)
11. Kristensen, B.B., et al.: Abstraction Mechanisms in the BETA Programming Language. in Tenth ACM Symposium on Principles of Programming Languages. Austin, Texas (1983)
12. Kristensen, B.B., Madsen, O.L., Møller-Pedersen, B.: The When, Why and Why Not of the BETA Programming Language. In: HOPL III: The third ACM SIGPLAN Conference on History of Programming Languages. San Diego (2007)
13. Madsen, O.L., Møller-Pedersen, B.: What Object-Oriented Programming May Be—and What It Does Not Have to Be. In: ECOOP'88 – European Conference on Object-Oriented Programming, Oslo, Norway. Springer Verlag (1988)
14. Aalto, J.-M., Jaaksi, A.: Object-Oriented Development of Interactive Systems with OMT++. In: TOOLS 14 – Technology of Object-Oriented Languages and Systems. Prentice Hall (1994)
15. Kruusela, J.: Object oriented development of embedded systems with the octopus method. In: Lectures on Embedded Systems – European Educational Forum School on Embedded Systems Veldhoven, The Netherlands November 25–29. Springer (1996)
16. Awad, M., Kuusela, J., Ziegler, J.: Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion. Prentice Hall (1996)

17. Jacobson, I.: Language Support for Changeable Large Real Time Systems. in OOPSLA'86 – Object-Oriented Programming, Systems Languages and Applications, Portland, Oregon. In: ACM Special Issue of Sigplan Notices (1986)
18. Rockström, A., Saracco, R.: SDL–CCITT Specification and Description Language. IEEE Transactions on Communications, 30(6) (1982)
19. Hemdal, G.: AXE 10 – Software Structure and Features. Ericsson Review, 2 (1976)
20. Haff, P., Olsen, A.: Use of VDM within CCITT. In: VDM '87 VDM – A Formal Method at Work, pp. 324–330. Springer, Berlin & Heidelberg (1987)
21. Belina, F., Hogrefe, D.: The CCITT Specification and Description Language SDL. Computer Networks and ISDN Systems, 16 (1989)
22. Madsen, O.L., Møller-Pedersen, B., Nygaard, K.: Object-Oriented Programming in the BETA Programming Language. Addison Wesley (1993)
23. Møller-Pedersen, B., Belsnes, D., Dahle, H.P.: Rationale and Tutorial on OSDL: An Object-Oriented Extension of SDL. Computer Networks, 13(2) (1987)
24. Olsen, A., et al.: Systems Engineering Using SDL-92. North-Holland (1994)
25. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons Inc. (1994)
26. Övergaard, G., Palmkvist, K.: Interacting Subsystems in UML. In: <<UML>>2000 – The Unified Modeling Language – Third International Conference, York, UK. Springer (2000)
27. OMG: UML 2.0 RESPONSE ON REQUEST FOR INFORMATION from Ericsson, Motorola, and Alcatel, Version 1.0. (1999)
28. Fischer, J., Holz, E., Møller-Pedersen, B.: Structural and Behavioral Decomposition in Object Oriented Models. In: ISORC (International Symposium on Object-oriented Real-time Systems). New Beach, CA (2000)
29. Bræk, R., Møller-Pedersen, B.: Frameworks by Means of Virtual Types—Exemplified by SDL. In: IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII) (1998)
30. Garlan, D., et al.: Modeling of Architectures with UML. In: The Third International Conference on The Unified Modeling Language, 2–6 October, 2000. York (2000)
31. Selic, B.: Using UML for Modeling Complex Real-Time Systems. In: Mueller, F., Bestavros, A. (eds.) Languages, Compilers, and Tools for Embedded Systems. Springer-Verlag (1998)
32. Rumpe, B., et al.: UML+ROOM as a Standard ADL? In: 5th International Conference on Engineering of Complex Computer Systems (1999)
33. Haugen, Ø., Møller-Pedersen, B., Weigert, T.: Structural Modeling with UML 2.0. In: Lavagno, L., Martin, G., Selic, B. (eds.) UML for Real. Kluwer Academic Publishers (2003)
34. Selic, B.: UML 2: A Model-driven Development Tool. IBM Systems Journal (2006)
35. Møller-Pedersen, B.: From SDL to UML2 – from an ITU Specification Language to an OMG Modeling Language. Telektronikk, 1 (2009)
36. Harel, D., Gery, E.: Executable Object Modeling with Statecharts. IEEE Computer (1997)
37. Møller-Pedersen, B., Nogva, D.: Scalable and Object Oriented SDL State(chart)s. In: IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques (FORTE XII) FORTE'99. Beijing (1999)
38. Jacobson, I.: Object Oriented Development within an Industrial Environment. In: OOPSLA'87 – Object-Oriented Programming, Systems Languages and Applications, Orlando, Florida. ACM (1987)
39. Nygaard, K.: Program Development as a Social Activity. In: INFORMATION PROCESSING 86 – IFIP 10th World Computer Congress, Dublin, Ireland. Elsevier Science Publishers B.V. (1986)

40. Reenskaug, T., Wold, P., Lehne, O.A.: Working with Objects: The OOram Software Engineering Method. Manning/Prentice Hall (1996)
41. Wirfs-Brock, R.J., Johnson, R.E.: Surveying Current Research in Object-Oriented Design. Communications of the ACM, 33(9), 113–116 (1990)
42. Övergaard, G.: A Formal Approach to Collaborations in the Unified Modeling Language. In: «UML»'99: The Unified Modeling Language – Beyond the Standard, Second International Conference, Fort Collins, CO. Springer (1999)
43. Grabowski, J., Rudolph, E.: Putting Extended Sequence Charts to Practice. In: 4th SDL Forum. North-Holland, Lisbon (1989)
44. ITU: Message Sequence Charts (MSC), Recommendation Z.120. Geneva (1999)
45. Haugen, Ø.: From MSC-2000 to UML 2.0 – The Future of Sequence Diagrams. In: 10th International SDL Forum. Copenhagen (2001)
46. Haugen, Ø.: Comparing UML 2.0 Interactions and MSC-2000. In: SAM 2004: SDL and MSC Fourth International Workshop. Ottawa (2004)
47. OMG: Semantics of a Foundational Subset for Executable UML Models (2009)
48. Madsen, O.L., Møller-Pedersen, B.: A Unified Approach to Modeling and Programming. In: MODELS 2010. Springer, Oslo (2010)