Models, To Model, and Modelling
-
Towards a Theory of Models, especially Conceptual Models and Modelling
-
Fourth Collection of Recent Papers (2020–2021)

Bernhard Thalheim

Christian-Albrechts-University Kiel,
Christian-Albrechts-Platz 4, 24118 Kiel, Germany
bernhard.thalheim@email.uni-kiel.de

Version 26. November 2021

Definition 1 [Generic notion of model.][Thaa]
A model is a well-formed, adequate, and dependable instrument that represents 'something' (called origin as a source, archetype, starting point) and that functions in scenarios of use.

Its criteria of well-formedness, adequacy, and dependability must be commonly accepted by its *community of practice* within some *context* and correspond to the *functions* that a model fulfills in *utilisation scenarios*.

As an instrument or more specifically an artifact 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. The background is often implicit and hidden.

A well-formed instrument is *adequate* for a collection of origins if it is *analogous* to the origins to be represented according to some analogy criterion, it is more *focused* (e.g. simpler, truncated, more abstract or reduced) than the origins being modelled, and 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* characterisation for internal quality, external quality and quality in use or through quality characteristics such as correctness, generality, usefulness, comprehensibility, parsimony, robustness, novelty etc. Sufficiency is typically combined with some assurance 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.

"We notice that all properties are parametric and can be refined in dependence of their envisioned function in scenarios of use. Configuration is a typical refinement in modelling. It can be based on a characterisation of the nature of the model, on an embedment into the application landscape and their scenarios, on strategic plan for achieving modelling objectives, and on tactical supporters and enablers for attaining objectives. The nature of a model is given by its representation of origins and its functioning as an instrument. The landscape of a model embeds the model-being into the application scenario. A simple landscape characterisation is based on the mission and the brand of a model (see [ST12a]). Strategy is given by the form of usage (i.e. mould) and the model background (i.e. deep model). Tactics is based from one side on enablers such as languages and workshop tools and on the other side on supporting means from the application area, e.g. knowledge, concept spaces, grounding, and basis. The nature, landscape, strategy and tactics are combined into the configuration of the model." [Thaa]

Definition 2 [Configured notion of model.][Thaa]
Given a configuration for modelling. A disciplinary model is a model with a pre-configured adequacy and dependability that functions according to the configuration requirements.

Content of This Fourth Collection

See also the collections of videos

1. youtube Channel “Bernhard Thalheim”,
2. vk.com https://vk.com/id349869409 (English),
3. vk.com https://vk.com/id463894395 (German) or
4. vimeo.

Content of The Third Collection

Available through Research Gate and Academia.

Remark 1: See our previous work on the ***entity-relationship approach to modelling***, e.g. [ET11, Tha00, MST09b, ST15], on semantics [ST13, Tha11b], on programming, on the theory of databases and information systems and on the technology of information systems at
http://dblp.uni-trier.de/pers/hd/t/Thalheim:Bernhard
or https://www.researchgate.net/profile/Bernhard_Thalheim/publications or
https://scholar.google.com/citations?user=lkH3h9gAAAAJ or
http://independent.academia.edu/BernhardThalheim or ...

Remark 2: Website information is going to be transferred to
to http://bernhard-thalheim.de/ and on M2P/MaP to http://bernhard-thalheim.de/ModellingToProgram/index.html .
The new email address is going to be: bernhard.thalheim@email.uni-kiel.de .

Remark 3: The Podcast (in German) on models you will find at
https://perspektivenpodcast.podigee.io/10-bernhard-thalheim-uber-konzeptuelle-modellierung-und-den-kieler-modellbegriff
.

Content of The Second Collection

Available through Research Gate and Academia.

Remark: page numbers from the second collection in the sequel.

The First Collection

available through research gate and academia

Remark: page numbers from the first collection in the sequel.

Preview for Fifth Collection (submitted 2021)

1. Modell-Kunde: Kurz & Knapp (KunstModell)
2. Models – The Fourth Dimension of Computer Science (FoundationCS)
3. Models in Life, Sciences, and Engineering as Universal Instruments – The Specifics of Social Models (KunstModell)
4. Fragen der Modellkunde an die Ägyptologie (KunstModell)
5. Model-Based Reasoning for Investigating the Heart Capability (Herzmodell)

Already at Research Gate with password protection (project name).

backward

Literatur

[ADT12a] S. Amarakoon, A. Dahanayake, and B. Thalheim. Domain requirements modeling framework for cross-disciplinary service systems development. In *ICCNDT*, pages 152–166. Gulf University, Bahrain, 2012.

[ADT12b] S. Amarakoon, A. Dahanayake, and B. Thalheim. A framework for modelling medical diagnosis and decision support services. *International Journal of Digital Information and Wireless Communications (IJDIWC)*, 2(4):7–26, 2012.

[AFT09] Sabah S. Al-Fedaghi and Bernhard Thalheim. Personal information databases. *IJCSIS*, 5(1):11–20, 2009.

[AGK+19] V.P.J. Arponen, S. Grimm, L. Käppel, K. Ott, B. Thalheim, Y. Kropp, K. Kittig, J. Brinkmann, and A. Ribeiro. Between natural and human sciences: On the role and character of theory in socio-environmental archaeology. *The Holocene*, 29(10, Special Issue "Scales of Transformation: Human-Environmental Interaction in Prehistoric and Archaic Societies"):1671–1676, October 2019.

[BFA+16] M. Bichler, U. Frank, D. Avison, J. Malaurent, P. Fettke, D. Hovorka, J. Krämer, D. Schnurr, B. Müller, L. Suhl, and B. Thalheim. Theories in business and information systems engineering. *Business & Information Systems Engineering*, pages 1–29, 2016.

[BST09] E. Börger, O. Sörensen, and B. Thalheim. On defining the behavior of or-joins in business process models. *Journal of Universal Computer Science*, 15(1):3–32, 2009.

[BT12] P. Broman and B. Thalheim. Interactive data extraction from semi-structured text. In *Information Modelling and Knowledge Bases XXII*, volume 237 of *Frontiers in Artificial Intelligence and Applications*, pages 1–19. IOS Press, 2012.

[CKT] P. Chawakitchareon, Y. Kiyoki, and B. Thalheim. Water quality index analysis and prediction: A case study of channel in Bangkok Thailand.

[DT10a] A. Dahanayake and B. Thalheim. Co-evolution of (information) system models. In *EMMSAD 2010*, volume 50 of *LNBIP*, pages 314–326. Springer, 2010.

[DT10b] A. Dahanayake and B. Thalheim. Towards a framework for emergent modeling. In *ER Workshops*, volume 6413 of *Lecture Notes in Computer Science*, pages 128–137. Springer, 2010.

[DT11] A. Dahanayake and B. Thalheim. Enriching conceptual modelling practices through design science. In *BMMDS/EMMSAD*, volume 81 of *Lecture Notes in Business Information Processing*, pages 497–510. Springer, 2011.

[DT12] A. Dahanayake and B. Thalheim. A conceptual model for IT service systems. *Journal of Universal Computer Science*, 18(17):2452–2473, 2012.

[DT13a] A. Dahanayake and B. Thalheim. The conception of a service model. In *Proc. ICNVICT'2013*, pages 36–49, Amman, 2013. IEEE.

[DT13b] A. Dahanayake and B. Thalheim. Continuous database engineering. *Int. Journal Business Inf. Syst. (IJBIS)*, 11(4):26–58, 2013.

[DT15] A. Dahanayake and B. Thalheim. W*H: The conceptual model for services. In *Correct Software in Web Applications and Web Services*, Texts & Monographs in Symbolic Computation, pages 145–176, Wien, 2015. Springer.

[DT18] A. Dahanayake and B. Thalheim. Development of conceptual models and the knowledge background provided by the rigor cycle in design science. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 3–28. College Publications, 2018.

[ET11] D. Embley and B. Thalheim, editors. *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*. Springer, 2011.

[FMPPT21] J. Fischer, B. Møller-Pedersen, A. Prinz, and B. Thalheim. Models versus model descriptions. In *Modelling to Program: Second International Workshop, M2P 2020, Lappeenranta, Finland, March 10–12, 2020, Revised Selected Papers*, volume 1401 of *CCIS*, pages 67–89. Springer Nature, 2021.

[HFHT18] J. Hedtrich, E. Fabritz, C. Henning, and B. Thalheim. Digital playground for policy decision making. In *Proc. XX Int. conf., DAMDID-RCDL'18*, pages 248–254, 2018.

[JHMT17] H. Jaakkola, J. Henno, J. Mäkelä, and B. Thalheim. Today is the future of yesterday; what is the future of today? In *Proc. MIPRO 2017, Opatija, Croatia*, pages 635–643. IEEE, 2017.

[JHMT19] H. Jaakkola, J. Henno, J. Mäkelä, and B. Thalheim. Artificial intelligence forever - again and again. In *Submitted to MiPRO*, page tba. IEEE, 2019.

[JHTM15] H. Jaakkola, J. Henno, B. Thalheim, and J. Mäkelä. Collaboration, distribution and culture – challenges for communication. In *MiPRO*, pages 657–664. IEEE, 2015.

[JHWD+16] H. Jaakkola, J. Henno, T. Welzer-Družovec, J. Mäkelä, and B. Thalheim. Why information systems modelling is so difficult. In *SQAMIA'2016*, pages 29–39, Budapest, 2016. CEUR Workshop Proceedings.

[JP10] K. Jannaschk and T. Polomski. A data mining design framework - A preview. In *Advances in Databases and Information Systems - 14th East European Conference, ADBIS 2010, Novi Sad, Serbia, September 20-24, 2010. Proceedings*, volume 6295 of *Lecture Notes in Computer Science*, pages 571–574. Springer, 2010.

[JT10] H. Jaakkola and B. Thalheim. Framework for high-quality software design and development: a systematic approach. *IET Software*, 4(2):105–118, 2010.

[JT11] H. Jaakkola and B. Thalheim. Architecture-driven modelling methodologies. In *Information Modelling and Knowledge Bases*, volume XXII, pages 97–116. IOS Press, 2011.

[JT18] H. Jaakkola and B. Thalheim. Modelling cultures. In Endrjukaite T., Jaakkola H., Thalheim B., and Kiyoki Y., editors, *Proc. 28'th EJC*, pages 33–52, Riga, Latvia, 2018. Transport and Telecommunication Institute.

[JT19a] H. Jaakkola and B. Thalheim. Cultures in information systems development. In *Information Modelling and Knowledge Bases XXX*, pages 61–80. IOS Press, 2019.

[JT19b] H. Jaakkola and B. Thalheim. Models as programs: The envisioned and principal key to true fifth generation programming. In *Proc. 29'th EJC*, pages 170–189, Lappeenranta, Finland, 2019. LUT, Finland.

[JT20a] H. Jaakkola and B. Thalheim. Model-based fifth generation programming. In *Information Modelling and Knowledge Bases Vol. XXXI*, Frontiers in Artificial Intelligence and Applications, 312, pages 381–400. IOS Press, 2020.

[JT20b] H. Jaakkola and B. Thalheim. Sixty years - and more - of data modelling. In *Information Modelling and Knowledge Bases XXXII*, volume 333 of *Frontiers in Artificial Intelligence and Applications*, pages 56–75. IOS Press, 2020.

[JTH+18] H. Jaakkola, B. Thalheim, J. Henno, J. Mäkelä, and H. Keto. Role of the user in information systems development. In *MiPRO*, pages 701–708. IEEE, 2018.

[JTH21] H. Jaakkola, B. Thalheim, and J. Henno. Polymorphism of intelligence – a look at human and artificial intelligence. In *Proc. 31'th EJC*, pages 24–47, Hamburg Germany, 2021. Kiel Computer Science Series, 2021/6.

[KST09] M. Kirchberg, O. Sörensen, and B. Thalheim. A BPMN case study: Paper review and submission system. In *GI Jahrestagung*, volume 154 of *LNI*, pages 4067–4081. GI, 2009.

[KT17a] F. Kramer and B. Thalheim. Metadata as support for data provenance. In *Information Modelling and Knowledge Bases XXVIII*, Frontiers in Artificial Intelligence and Applications, 280, pages 195–214. IOS Press, 2017.

[KT17b] Y. Kropp and B. Thalheim. Data mining design and systematic modelling. In *Proc. DAMDID/RCDL'17*, pages 349–356, Moscov, 2017. FRC CSC RAS.

[KT18a] Y. Kropp and B. Thalheim. Viewpoint-oriented data management in collaborating research projects. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 146–174. College Publications, 2018.

[KT18b] Y. O. Kropp and B. Thalheim. Deep model guided data analysis. In *DAMDID/RCDL 2017, Revised Selected Papers*, volume 822 of *Communications in Computer and Information Science*, pages 3–18. Springer, 2018.

[KT19] Y. Kropp and B. Thalheim. Model-based interface generation. In *Proc. 29'th EJC*, pages 70–87, Lappeenranta, Finland, 2019. LUT, Finland.

[KT20a] Y. Kropp and B. Thalheim. Conceptual modelling and humanities. In *Proc. Modellierung*, volume P-302 of *LNI*, pages 193–195. Springer, 2020.

[KT20b] Y. O. Kropp and B. Thalheim. Data management of age determinations in (pre-/proto-)history. In *Information Modelling and Knowledge Bases XXXII*, volume 333 of *Frontiers in Artificial Intelligence and Applications*, pages 187–195. IOS Press, 2020.

[KTD+20] Y. Kiyoki, B. Thalheim, M. Duzi, H.Jaakkola, P. Chawakitchareon, and A. Heimbürger. Towards a great design of conceptual modelling. Frontiers in Artificial Intelligence and Applications, 312, pages 531–544. IOS Press, 2020.

[MST09a] Hui Ma, K.-D. Schewe, and B. Thalheim. Modelling and maintenance of very large database schemata using meta-structures. In *UNISCON*, volume 20 of *Lecture Notes in Business Information Processing*, pages 17–28. Springer, 2009.

[MST09b] Hui Ma, Klaus-Dieter Schewe, and Bernhard Thalheim. Geometrically enhanced conceptual modelling. In *ER*, volume 5829 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2009.

[MSTQ09] Hui Ma, Klaus-Dieter Schewe, Bernhard Thalheim, and Q.Wang. A service-oriented approach to web warehousing. In *iiWAS*. ACM, 2009.

[MSTW09a] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang. A theory of data-intensive software services. *Service Oriented Computing and Applications*, 3(4):263–283, 2009.

[MSTW09b] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang. A theory of data-intensive software services. *Service Oriented Computing and Applications*, 3(4):263–283, 2009.

[MSTW11] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang. Cloud warehousing. *Journal of Universal Computer Science*, 17(8):1183–1201, 2011.

[MSTW12] Hui Ma, K.-D. Schewe, B. Thalheim, and Q. Wang. Conceptual modelling of services. *Journal of Universal Computer Science*, 18(17):2361–2363, 2012.

[MT19] A. Molnar and B. Thalheim. Usage models mapped to programs. In *Proc. M2P – New Trends in Database and Information Systems*, Bled, 2019. Springer, CCIS 1064.

[MT21] H. C. Mayr and B. Thalheim. The triptych of conceptual modeling. *Softw. Syst. Model.*, 20(1):7–24, 2021.

[NKT] I. Nissen, F. Kramer, and B. Thalheim. Underwater cooperation and coordination of manned and unmanned platforms using S-BPM. In *Information Modelling and Knowledge Bases XXX,*, pages 137–146.

[NT14] R. Noack and B. Thalheim. Multi-dimensional schema composition for conceptual modelling in the large. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 25–44. IOS Press, 2014.

[NT15] I. Nissen and B. Thalheim. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*, chapter The Notion of a Model, pages 615–618. De Gryuter, Boston, 2015.

[ST10] M. Skusa and B. Thalheim. Kohärente Softwareentwicklung - Grundlagen, Arbeitsumgebungen, Vorgehensweisen. Preprint 1018, Department of Computer Science, Kiel University, December 2010.

[ST12a] K.-D. Schewe and B. Thalheim. *Design and Development of Web Information Systems*. Springer, 2012. Forthcoming book.

[ST12b] E. Sivogolovko and B. Thalheim. Semantic approach to cluster validity notion. In *ADBIS (2)*, volume 186 of *Advances in Intelligent Systems and Computing*, pages 229–239. Springer, 2012.

[ST13] O. Sörensen and B. Thalheim. Semantics and pragmatics of integrity constraints. In *SDKB'11, LNCS 7693*, pages 1–17. Springer, 2013.

[ST15] K.-D. Schewe and B. Thalheim. Co-design of web information systems. Texts & Monographs in Symbolic Computation, pages 293–332, Wien, 2015. Springer.

[ST17] V. Storey and B. Thalheim. Conceptual modeling: Enhancement through semiotics. In *Proc. ER'17*, LNCS, 10650, pages 182–190, Cham, 2017. Springer.

[ST19] K.-D. Schewe and B. Thalheim. *Design and development of web information systems*. Springer, Chur, 2019.

[TD15] B. Thalheim and A. Dahanayake. A conceptual model for services. In *Invited Keynote, CMS 2015, ER 2015 workshop*, LNCS 9382, pages 51–61, Berlin, 2015. Springer.

[TD16] B. Thalheim and A. Dahanayake. Comprehending a service by informative models. *T. Large-Scale Data- and Knowledge-Centered Systems*, 30:87–108, 2016.

[TDT21] I. Triapitcin, A. Dahanayake, and B. Thalheim. Semantic discovery from sensors and image data for real-time spatio-temporal emergency monitoring. In *Proc. ADBIS 2021 Short Papers*, volume 1450 of *Communications in Computer and Information Science*, pages 82–92. Springer, 2021.

[TELT14] S. Torge, W. Esswein, S. Lehrmann, and B. Thalheim. Categories for description of reference models. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 229–240. IOS Press, 2014.

[TFTL+14] M. Tropmann-Frick, B. Thalheim, D. Leber, G. Czech, and C. Liehr. Generic workflows - a utility to govern disastrous situations. In *Information Modelling and Knowledge Bases*, volume XXVI of *Frontiers in Artificial Intelligence and Applications, 272*, pages 417–428. IOS Press, 2014.

[Thaa] B. Thalheim. Models – the fourth dimension of computer scienc;towards studies of models and modelling.

[Thab] B. Thalheim. Models in life, sciences, and engineering as universal instruments – the specifics of social models. In *Proc. Egyptology Congress, Luxor Nov. 2019*, pages 120–134.

[Tha91] B. Thalheim. *Dependencies in relational databases*. Teubner, Leipzig, 1991.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.

[Tha09a] B. Thalheim. Abstraction. In *Encyclopedia of Database Systems*, pages 6–7. Springer US, 2009.

[Tha09b] B. Thalheim. Extended entity-relationship model. In *Encyclopedia of Database Systems*, pages 1083–1091. Springer US, 2009.

[Tha09c] B. Thalheim. Specialization and generalization. In *Encyclopedia of Database Systems*, pages 2746–2747. Springer US, 2009.

[Tha09d] B. Thalheim. Towards a theory of conceptual modelling. In *ER Workshops*, volume 5833 of *Lecture Notes in Computer Science*, pages 45–54. Springer, 2009.

[Tha10a] B. Thalheim. Model suites for multi-layered database modelling. In *Information Modelling and Knowledge Bases XXI*, volume 206 of *Frontiers in Artificial Intelligence and Applications*, pages 116–134. IOS Press, 2010.

[Tha10b] B. Thalheim. Towards a theory of conceptual modelling. *Journal of Universal Computer Science*, 16(20):3102–3137, 2010. `http://www.jucs.org/jucs_16_20/towards_a_theory_of`.

[Tha11a] B. Thalheim. *Anwendungsorientierte Organisationsgestaltung*, chapter The Culture and Art of Conceptual Modelling, pages 127–144. baar, Hamburg, 2011.

[Tha11b] B. Thalheim. Integrity constraints in (conceptual) database models. In *The Evolution of Conceptual Modeling*, volume 6520 of *Lecture Notes in Computer Science*, pages 42–67, Berlin, 2011. Springer.

[Tha11c] B. Thalheim. The science of conceptual modelling. In *Proc. DEXA 2011*, volume 6860 of *LNCS*, pages 12–26, Berlin, 2011. Springer.

[Tha11d] B. Thalheim. The theory of conceptual models, the theory of conceptual modelling and foundations of conceptual modelling. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 17, pages 547–580. Springer, Berlin, 2011.

[Tha12a] B. Thalheim. The art of conceptual modelling. In *Information Modelling and Knowledge Bases XXII*, volume 237 of *Frontiers in Artificial Intelligence and Applications*, pages 149–168. IOS Press, 2012.

[Tha12b] B. Thalheim. The notion of a conceptual model. In N. Seyff and A. Koziolek, editors, *Modelling and Quality in Requirements Engineering*, pages 21–30. Verlagshaus Monsenstein und Vannerdat, 2012. ISBN 978-3-86991-740-5.

[Tha12c] B. Thalheim. The science and art of conceptual modelling. In A. Hameurlain et al., editor, *TLDKS VI*, LNCS 7600, pages 76–105. Springer, Heidelberg, 2012.

[Tha12d] B. Thalheim. Syntax, semantics and pragmatics of conceptual modelling. In *NLDB*, volume 7337 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2012.

[Tha13a] B. Thalheim. The definition of the (conceptual) model. In *Proc. EJC 2013*, pages 256–269, Nara, Japan, 2013.

[Tha13b] B. Thalheim. Open problems of information systems research and technology. In *Invited Keynote, BIR'2013. LNBIB 158*, pages 10–18. Springer, 2013.

[Tha14] B. Thalheim. The conceptual model $\equiv$ an adequate and dependable artifact enhanced by concepts. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 241–254. IOS Press, 2014.

[Tha15] B. Thalheim. Das Modell des Modelles. *Erwägen-Wissen-Ethik*, EWE-Heft, Heft 3, 2015, 26. Jg.:407–409, 2015.

[Tha17a] B. Thalheim. General and specific model notions. In *Proc. ADBIS'17*, LNCS 10509, pages 13–27, Cham, 2017. Springer.

[Tha17b] B. Thalheim. Model-based engineering for database system development. In *Conceptual Modeling Perspectives*, pages 137–153, Cham, 2017. Springer.

[Tha18a] B. Thalheim. Conceptual model notions - a matter of controversy; conceptual modelling and its lacunas. *EMISA International Journal on Conceptual Modeling*, February:9–27, 2018.

[Tha18b] B. Thalheim. Model adequacy. In *Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018, Braunschweig, Germany, February 21, 2018.*, volume 2060 of *CEUR Workshop Proceedings*, pages 11–18. CEUR-WS.org, 2018.

[Tha18c] B. Thalheim. Normal models and their modelling matrix. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 44–72. College Publications, 2018.

[Tha18d] B. Thalheim. Qualitative and quantitative models. In *Proc. XX Int. conf., DAMDID-RCDL'18*, pages 112–119, 2018.

[Tha19a] B. Thalheim. Models, To Model, and Modelling — Towards a Theory of Models, especially Conceptual Models and Modelling. *Collection of Papers*. https://www.researchgate.net/publication/338685564_Models_To_Model_and_Modelling_-_Towards_a_Theory-_of_Models_especially_Conceptual_Models_and_Modelling_-_Third_Collection_of_Recent_Papers_2018-2019 , https://www.researchgate.net/publication/320688612_Models_To_Model_and_Modelling_-_Towards_a_Theory-_of_Models_especially_Conceptual_Models_and_Modelling_Second_Collection_of_Recent_Papers_2015-2017 , and https://www.researchgate.net/publication/344510996_Models_To_Model_and_Modelling_-_Towards_a_Theory-_of_Models_especially_Conceptual_Models_and_Modelling_-_First_Collection_of_Recent_Papers_2009-2014 , 2009–2019.

[Tha19b] B. Thalheim. Conceptual modeling foundations: The notion of a model in conceptual modeling. In *Encyclopedia of Database Systems*. Springer US, 2019.

[Tha19c] B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, pages 123–139. Springer, 2019.

[Tha19d] B. Thalheim. Models and their foundational framework. *Studia Metodologiczne*, (39):11–48, 2019.

[Tha19e] B. Thalheim. Models for communication, understanding, search, and analysis. In *Proc. XXI (DAMDID/RCDL 2019), CEUR Workshop Proceedings, vol. 2523, Kazan, Russia, October 15-18, 2019.*, pages 19–34, 2019.

[Tha20a] B. Thalheim. Models as programs: A tutorial. In *Modellierung 2020, 19.-21. Februar 2020, Wien, Österreich*, volume P-302 of *LNI*, pages 193–195. Gesellschaft für Informatik e.V., 2020.

[Tha20b] B. Thalheim. Schema optimisation instead of (local) normalisation. In *Proc. FoIKS*, LNCS 12012, pages 1–20. Springer, 2020.

[Tha20c] B. Thalheim. Surrounded with models in computer science. In *The art and craft of scientific publishing*, LNAH 2020, pages 141–150. Springer, 2020.

[Tha21a] B. Thalheim. From models for programming to modelling to program and towards models as programs. In *Modelling to Program: Second International Workshop, M2P 2020, Lappeenranta, Finland, March 10–12, 2020, Revised Selected Papers*, volume 1401 of *CCIS*, pages 3–44. Springer Nature, 2021.

[Tha21b] B. Thalheim. Model-based reasoning. In *Proc. 31'th EJC*, pages 4–23, Hamburg Germany, 2021. Kiel Computer Science Series, 2021/6.

[Tha21c] B. Thalheim. Models and modelling in computer science. In *Logic, Computation and Rigorous Methods*, LNCS 12750, pages 301–325, Cham, 2021. Springer.

[TJ19] B. Thalheim and H. Jaakkola. Models and their functions. In *Proc. 29'th EJC*, pages 150–169, Lappeenranta, Finland, 2019. LUT, Finland.

[TJ20] B. Thalheim and H. Jaakkola. Functions of models and their maturity. In *Information Modelling and Knowledge Bases Vol. XXXI*, Frontiers in Artificial Intelligence and Applications, 312, pages 265–284. IOS Press, 2020.

[TN15] B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.

[TSF19] B. Thalheim, A. Sotnikov, and I. Fiodorov. Models: The main tool of true fifth generation programming. In *Proc. EEKM 2019 – Enterprise Engineering and Knowledge Management*, pages 161–170, Moscov, 2019. CEUR workshop poceedings, Vol. 2413.

[TT13] M. Tropmann and B. Thalheim. Mini story composition for generic workflows in support of disaster management. In *DEXA 2013*, pages 36–40. IEEE Computer Society, 2013.

[TT16] B. Thalheim and M. Tropmann-Frick. Model capsules for research and engineering networks. In *New Trends in Databases and Information Systems*, volume 637 of *CCIS*, pages 202–214. Springer, 2016.

[TTF15] B. Thalheim and M. Tropmann-Frick. The conception of the conceptual database model. In *ER 2015*, LNCS 9381, pages 603–611, Berlin, 2015. Springer.

[TTF16a] B. Thalheim and M. Tropmann-Frick. Models and their capability. In C. Beierle, G. Brewka, and M. Thimm, editors, *Computational Models of Rationality*, volume 29 of *College Publications Series*, pages 34–56. College Publications, 2016.

[TTF16b] B. Thalheim and M. Tropmann-Frick. Wherefore models are used and accepted? The model functions as a quality instrument in utilisation scenarios. In I. Comyn-Wattiau, C. du Mouza, and N. Prat, editors, *Ingénierie Management des Systèmes d'Information*, pages 131–143. Cépaduès, 2016.

[TTFZ14] B. Thalheim, M. Tropmann-Frick, and T. Ziebermayr. Application of generic workflows for disaster management. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 64–81. IOS Press, 2014.

Model-Based Reasoning

Bernhard THALHEIM [1]
Christian-Albrechts-University Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, Germany

Abstract. Models are used everywhere, in daily life, sciences, engineering, and thoughts. They represent, support, and enable our thinking, acting, reflecting, communication, and understanding. They are universal instruments. Model-based reasoning is, however, different from those that we use in 'exact' sciences and is far less understood. The notion of model is becoming nowadays well-accepted. Model-based reasoning is far less understood and a long lacuna. This keynote aims at closing this gap.

Keywords. model-based reasoning, plausible reasoning, approximative reasoning, abduction, induction, explanation, hypotheses, empiric

1. Introduction

Humans intensively use models everywhere, at any time, for any reason, by everybody, for everybody, and at any sphere of human activity. They transform complex, abstract, or partial ideas, systems, and theories into more easily to understand and simpler to use things, i.e. humanise them in dependence on human abilities. Pupils learn natural sciences through models. They are already used to deploy models with their first thoughts. The very first intellectual instrument we use is a model. It is not surprising that babies quickly develop their own models or at least concepts of the 'mother' and 'father'. They cannot yet use a natural language but they know already models of their surroundings [12]. Later they realise that their models are completely different from those of their contemporaries.

We are using the word 'model' widely in our daily life as well as in sciences and engineering. Models are also widely used in the social sphere, in religion, in communication, in interaction, and collaboration. They must not be correct but should be useful as an instrument ('*model_for*'). Models can be understood as a collection of competing interpretations, perception, prehension, ideas, comprehension, imaginations, or conceptions about the world a human observes and understands, each with a utility core, which nevertheless must prove to be progressive over time. This wide usage of models direct us to consider models as the fourth sphere of our life beside sensing and reflecting the *world of the being*, acting and mastering the *world we create*, and the *intelligible world* of science, knowledge, and concept(ion)s. Models can be considered as our 'third reflection

[1] bernhard.thalheim@email.uni-kiel.de

eye' ('*model_of*') we use for comprehension, acceptance, understanding, finding our way around, socialising, communication, planing, and actuation.

The theory and practice of models and modelling is already fairly rich (see, for instance, [33]) and resulted in a large body of knowledge for almost all disciplines of science and engineering. Let us start with five observations:

Models have not to be called 'model' since we use them anyway as some kind of explicit means or instruments. They are far older than the explicit naming of things as models. The oldest explicit model we definitely know as model is older than 4.000 years. Models often appear as conceptions, notions, visions, images, view(point)s, concepts, pictures, basic orientations, performances, comprehensions, representations etc.

Mental models are used by everybody in an explicit but often implicit form. They can be less rigid than notions we use in science and engineering but they are useful. As such we consider ideas, imaginations, perceptions, prospects as a mental picture, beliefs, conceivability, visions, and imaginations as models.

Models are also used in manufacturing for instance as as a template, pattern, reference, presentation, prototype, origin for production, master copy, and sample.

Sciences and technology might claim that the are model-free but are typically 'downright contaminated' with models. Look, for instance, to claims made in Computer Science. There is neither a systematic study of models not a modelling sub-discipline. However, almost anything is done by use of models.

Models can but don't have to be explicitly designed for usage. Engineering as the approach to create the artificial [27] is based on models as a starting point for construction, as a documentation of the construction, as a means for developing variations, as a thought and imagination instrument, and as an artifact within the creation process. There are, however, objects that became models at a far later stage of their existence.

These few observations allow use to expect that models are one of our main instruments similar to the broad usage of languages. Moreover, modelling is an activity that comes before we learn a language and accompanies us our entire life. So, we should ask ourselves first: What is a model? Next, we have to ask: what is model-based reasoning? The first question got already hundreds of answers. The third one almost none. Therefore, the paper aims at answering the third question after gaining an understanding of the first and second answer.

2. Modellkunde – Towards a Study of Models and Modelling

2.1. The Notion of Model

The notion we use since [29] generalises almost all notions or pre-notions used and known so far in general model theory [13,20,21,28,33][2]:

[2]Its advantage is that all notion we have seen so far can be understood as a parametric specialisation. More specific notions can be declined by parameter refinement and hardening from this notion.

"A **model** *is a well-formed, adequate, and dependable instrument that represents origins*[3] *and that functions in utilisation scenarios.*
Its criteria of well-formedness[4], adequacy[5], and dependability[6] must be commonly accepted by its community of practice (CoP) within some context and correspond to the functions that a model fulfills in utilisation scenarios." [30]

This notion also allows consideration of the *model-being of any instrument*[7]. Anything – any thought and any thing – can be a model as long as it is used as such. The model-being is, therefore, an assignment for an instrument that is used in scenarios.

2.2. Functions of Models in Scenarios

Models function in application scenarios, i.e. they have in those scenarios a function[8,9]. Typical functions in science and engineering scenarios are reflection, illustration, visualisation, being a theory surrogate, guiding thoughts and activities, aiding for theory construction, mediating, and substituting theories.

Models are used instruments. The instrument-being is, thus, a pre-requisite for the model-being. The means that models have to be optimised on the function that the model has in the given application scenario. Instead of considering holistic models, model suites with a sophisticated and explicit association schema among models in the model suite are far better accommodated to model-based reasoning and deployment in scenarios. A scenario consists of a task space and an envisioned delivery space. Instruments may functions in a variety of ways. Therefore, a model may serve in several functions. Also, a scenario may consist of a collection of scenarios. The upper part in Figure 1 depicts this 'landscape' of the model-being.

---

[3]The 'origin' is different from 'original'. 'Origin' means the source of something's existence or from which it derives or is derived. It points to the place, event, the point of origination, the initial stage of a developmental process, etc. where something begins, where it springs into being.

[4]*Well-formedness* is often considered as a specific modelling language requirement.

[5]The criteria for *adequacy* are analogy (as a generalisation of the mapping property that forms a rather tight kind of analogy), being focused (as a generalisation of truncation or abstraction), and satisfying the purpose (as a generalisation of classical pragmatics properties).

[6]The model has another constituents that are often taken for granted. The model is based on a background, represents origins, is accepted by a community of practice, and follows the accepted context. The model, thus, becomes *dependable*, i.e. it is justified or viable and has a sufficient quality. Most notions assume dependability either as a-priori given or neglect it completely.

[7]We note that the instrument-being is based on the function that a model plays in some scenario.

[8]The word 'function' has seven word fields for the noun and three for the verb. We use here the meaning of a function that is associated with purpose, role, use, utility, usefulness, i.e. what something is used for.

[9]The word 'function' is often considered a synonym of 'goal' or 'purpose'. We distinguish the three word and use a layered approach: *Goal* is definable as a ternary relation between initial state, desired states and community of practice who may assess the states and follow their beliefs, desires, and intentions. *Purposes* extend goals by means, e.g. methods, techniques, and operations. *Functions* embed the model into practices in applications and, thus, relate the purpose to the application, i.e. as a role and play of the model in an application scenario.

2.3. The Model-Being of Things and Thoughts

The model-being is determined by the function of an instrument in an application scenario. Nothing is a-priori a model. Things and thought have not to be models forever. Models have their journey in the model-being. They can be used in one function, remain to be useful or pass away as model. They can be used in a different function at a later stage. Criteria for the model-being seem to be necessary for some demarcation, i.e. a discrimination between things and thoughts as different and distinct on the basis of their characteristics or attributes. The demarcation can be derived from the model-being of an instrument and from the instrument-being of something:

1. A model functions in scenarios. It may functioning well, optimally, flawlessly, properly, satisfactorily, or primarily. Or barely and poorly.
2. A model may have several functions. The function might change during model's existence.
3. Functions can be characterised. This characterisation is an essential element of the mission, determination, meaning and identity of something[10].
4. Functioning may be matured. The maturity level depends on the model objectives.
5. Functioning can only be defined for specified scenarios. There is no universal function of a model.
6. Model functions determine the adequacy and dependability.

Typical engineering functions are blueprint for realisation, starting point, prescription, mould, guide, companion, modernisation, integration, replacement, deploy, informative, recording, and assess. These functions, the usefulness, the utility, and the quality in and of use determines whether an instrument is a model. The instrument-being is based on the actual, practised, skilled, ideal, and desired play of a role in an application scenario. There are two main roles: the reflection of those origins the model have to be represented and the achievable result through use of the instrument. The instrument-being depends on the temporal, spatial, and disciplinary context of the community of practice.

The model-being is based on three viewpoints that determine the model utility as a mediator (see Figure 1): (1) the model-being as a 'model_of'[11] (2) the ends for the model-being as a 'model_for' [12] (3) the model-being based on the mediator function of the instrument[13]. Mediation includes transfer of main properties of origins that are essential in the given scenario to the result by means of the model, i.e. the model 'transports' those properties to the results of model application as *invariants*. The explication of the mediation can be directly given as

[10] B. Mahr [19,20] introduced the notion of *cargo* as a carrier of main properties and objectives of origins to important issues for the result. It describes the instrument, the main functions, the forbidden usages, the specific values of the instrument, and the context for the usage model.

[11] Representing a collection of origins that are really of interest and relevant, i.e. the whereof as the source for the models existence or from which it derives or is derived.

[12] Essentially, a plan that is intended to achieve and that (when achieved) terminates behavior intended to achieve it, i.e. the for what as the cause or intention underlying model usage.

[13] Determined by a function in a given context and scenario, i.e. whereby as the helper that offers benefits and supporting means for achieving a result.

an informative model of the model (as an instrument). Informative models [32] are, thus, essentially the 'product insert' of the model. This model of the model is used as some kind of a leaflet or model suite insert that represents the essentials of the model suite. That means, we use already a model suite consisting of at least two models.

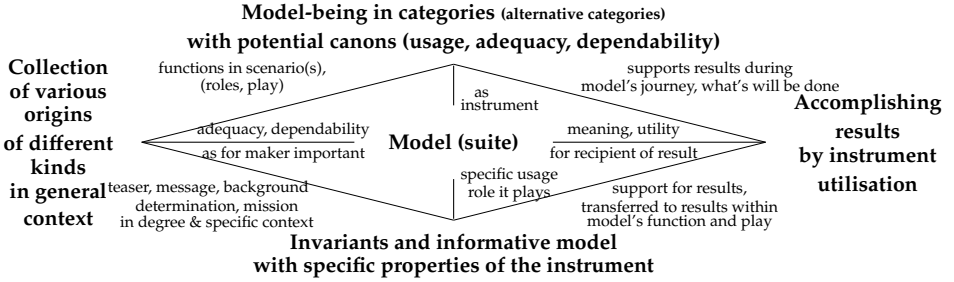


Figure 1. Characterising the model-being of instruments according to reflected origins and results to be accomplished by means of the model

The large variety of models in life, science, engineering, and thought seems to prevent development of a general 'Modellkunde' as a systematic study of models and modelling. There are (a) perception or mental, (b) representation or reflection, (c) communication or collaboration, (d) actuation and activity, (e) guidance or steering, (f) thought or reasoning, (g) substitution or sense-making, (h) socialisation or interaction, and (i) orientation models. These nine kinds of model follow however their specific adequacy and dependability canon and provide means for their usage. We, thus, realise that we have essentially nine different model categories of models.

As any instrument, a model has its own additional properties that are neither of importance for origins nor for results, its own authority, its obstinacy[14], its profile (goals, purposes, functions) and anti-profile, its modus agendi and mode action, and its materiality. A model may, thus, also be misleading, disorienting, and of lower quality than other models.

3. Model-Based Reasoning Beyond Classical Logics

Sciences are oriented on true statements and consistency of theories, at least to certain extent. This explains the omnipresence of deductive systems as the main reasoning mechanism. Models must not be true. They can be contradictious or even paraconsistent. Models should be useful at least in some application scenario, for some time, for some community of practice, in some context, for some origins, for some results, within some background, on the basis of some supporting and enabling mechanisms, and within human restrictions. Model-

[14]M.W. Wartofsky already states in [36]: "*There is an additional trivial truth, which may strike some people as shocking: anything can be a model! ... And although it is the case that anything can be a model of anything else, it is taken as a model which makes an actual out of a potential model; and every case of being taken as a model involves a restriction with respect to relevant properties.*"

based reasoning does not have to be entirely based on classical logics. A similar observation can be made for engineering[15]. The study of models and model-based reasoning has also to be based on other kinds of reasoning.

The study of models is also concerned with obstacles, mismatches, limitations, and restrictions of model-based reasoning. Despite the common belief in most books and research on a theory of models (e.g. [28]), model-based reasoning is, however, rather seldom based on deduction and deductive-nomological reasoning.

3.1. The Obstinacy of Classical Logics

Classical mathematical logics mainly considers deductive systems and various mechanisms of deduction. Already C.S. Peirce [22] distinguished three reasoning mechanisms: deduction, induction, and abduction. Their difference is illustrated in Table 1 for a set of premises, supporting means, and results: These three reason-

	Deduction	Induction	Abduction
Reasoning style	Rule-Data-Result	Data-Result-Rule	Rule-Result-Data
First	general rules	observed primary phenomena	general rules
Second	specific observations	dependent secondary phenomena	dependent secondary phenomena
Finally	conclusion for observations and new rules	rule supposition and questions	potential (causal) explanations

Table 1. Subduction: deduction, induction, abduction

ing styles are well-known. Deduction is considered to be the main mechanism. It is the basis for Mathematical Logics. Deductive reasoning is based on three postulates that are too restrictive: (A) completeness of the specification, (B) agreement on the background and the matrix, and (C) context-independence. The Peirce triangle is more general, however: From its *abductive* suggestion, *deduction* can draw a prediction which can be tested by *induction*. Abduction is well-known. Induction is far less accepted since rule suppositions are only hypothetical results and have to be revised whenever primary and secondary phenomena are not matching anymore. Many researchers, e.g. K.R. Popper[16], however, strictly deny usefulness and utility of induction and avoid usage of induction.

[15] See, for instance, [31] on the problematic side of first-order predicate logics for database engineering.

[16] [23]: *"Induction simply does not exist, and the opposite view is a straight-forward mistake. ... I hold that neither animals nor men use any procedure like induction, or any argument based on repetition of instances. The belief that we use induction is simply a mistake."*

3.2. *Inductive Model-Based Reasoning*

Induction is the most prominent and important reasoning mechanism in daily life and for model building based on evidences or observations. Inductive conclusions are uncertain due to the incompleteness of observations. Worlds that are potentially infinite are and will be, however, never completely observable. The inherent incompleteness of the world of phenomena shows that induction is the best logical mechanism for human reasoning.

We distinguish between:

- Induction in broad sense as explanatory inferences, as well as analogical and 'more-of-the-same" inferences in the style: *'All observed Xs have property P'* to *'The next X observed will have property P'*.
  It includes explanatory inferences, as well as analogical and 'more-of-the-same" inferences.
- Induction in narrow sense is based on a random sample (with test/validation set) and results in simple enumerative induction (or the straight rule).

Induction degrees are either strong inductive argument based on authority,on evidence, or stronger inductive argument based on better evidence.

The inductive reasoning schema is based on given knowledge $\mathcal{K}$, beliefs $\mathcal{B}$, models $\mathcal{M}$ known so far, and data $\mathcal{D}$ observed. Within the setting of some reasoning systems Γ, we assume $\mathcal{K} \cup \mathcal{M} \not\models_\Gamma \mathcal{D}$.
The induction task aims at discovery of a formula α (not uniquely defined) such that

- it is coherent with $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{D}$ and
- that allows to explain $\mathcal{D}$, i.e.
  - $\mathcal{K} \cup \mathcal{M} \cup \mathcal{D} \not\models_\Gamma \neg\alpha$ and
  - $\mathcal{K} \cup \mathcal{M} \cup \{\alpha\} \models_\Gamma \mathcal{D}$

Inductive reasoning schemata can be extended to Solomonoff induction bound by Kolmogorov complexity. In this case, sophisticated inductive reasoning generates most relevant, most simple, and preferred generalisations from facts and/or observations. The conclusions can be revised whenever the fact or observation set is extended. Inductive reasoning inherits the obstinacy of the representation language of facts and observations.

Induction is a kind of compilation-so-far reasoning with uncertainty, preferences for conclusions, and complexity reduction for result presentation.

Induction is transfer of likely truth from a number of observations to a general principle. It is based on conjecture spaces and specific approaches, experience, (tacit) knowledge, parsimony, economy, clever sampling, and wise experimentation. Induction is a very strong modelling principle. We appreciate statistical, probabilistic, possibilistic, eliminative, and mathematical induction. Induction can be treated as 'blind' search (depth-first, breath-first). It can also be clever search for suppositions as humans like to do.

3.3. *Abductive Model-Based Reasoning*

Abductive reasoning (e.g. [2,18,24]) is a kind of concise reasoning that infers particular cases from general observations and rules. It is a weak kind of inference because we cannot say that the explanation is true, but that it can be true.

Premises are given in the form:

- D is a collection of data, facts, observations
- M explains D within a given reasoning mechanism.
- No other model can explain D as well as M does.

Conclusion: Therefore, the model M is probably acceptable.

Abductive model-based reasoning is a process that tries to form plausible models for some situations. It covers also abnormal situations. The inference result is a model, which is somehow acceptable within the given reasoning mechanism and, thus, could explain the occurrence of the given facts. This approach can be used for detection of good explanations and especially good causal explanations.

A typical abductive hypothetical reasoning schema is the following:

1. Searching somehow anomalous, surprising, or disturbing phenomena and observations.
2. Observing details, little clues, and tones.
3. Continuous search for hypotheses and noting their hypothetical status.
4. Aiming at finding what kind or type of explanations or hypotheses might be viable to constraint the search in a preliminary way.
5. Aiming at finding explanations (or ideas) which themselves can be explained (or be shown to be possible).
6. Searching for 'patterns" or connections that fit together to make a reasonable unity.
7. Paying attention to the process of discovery and its different elements and phases.

Abductive reasoning also allows to consider negative information by modus tollens

$$\frac{H \to I\,,\ \neg I}{\neg H}$$

.

The Mathematical Model of Meaning [15] is a third kind of abductive reasoning schema that is used for categorisation of observations:

- Empirical observations can be represented by data representing the importance of some feature for the observation.
- Importance data should be normalised, e.g. 0, .., 10.
- Data can be represented as a table (or matrix) with some features/indicators as attributes.
- Attributes (in the universal world approach) can be related to categories. The database is then a universal relation with tuples where those values that are $\neq 0$ show belongness to a category.
- Multiplication of tuples from the observations with the feature-category matrix results in a tuple that characterises the belongness of an observation to a category.

Feature $F_j\,(1 \leq j \leq f)$ are relevant $d_{i,j}\,(1 \leq i \leq m,\, 1 \leq j \leq f)$ for observations. $o_i\,(1 \leq i \leq m)$. These features belong to categories $C_k\,(1 \leq k \leq R)$ by a knowledge or abduction matrix $c_{j,k}\,(1 \leq j \leq f,\, 1 \leq k \leq R)$.

$$\begin{pmatrix} & F_1 & ... & F_f \\ o_1 & d_{1,1} & ... & d_{1,f} \\ ... & ... & ... & ... \\ o_m & d_{m,1} & ... & d_{m,f} \end{pmatrix} \times \begin{pmatrix} & C_1 & ... & C_R \\ F_1 & c_{1,1} & ... & c_{1,R} \\ ... & ... & ... & ... \\ F_f & c_{f,1} & ... & c_{f,R} \end{pmatrix} = \begin{pmatrix} & C_1 & ... & C_R \\ o_1 & r_{1,1} & ... & r_{1,R} \\ ... & ... & ... & ... \\ o_m & r_{m,1} & ... & r_{m,R} \end{pmatrix}$$

The result of multiplication of the observation matrix with the abduction matrix is a matrix of relevance of a category $C_1, ..., C_R$ for an observation o_i.

3.4. Principles and Assumptions of Model-Based Reasoning

Models are instruments that properly function in utilisation scenarios. The utility is given by the quality of appropriateness in use. Therefore, we have to understand which objects, artifacts, and thoughts can really be used as models whenever we base reasoning on models. Appropriateness of models is a specific variant of the design principle 'form-follows-function'.

Goals, purposes, and function must be well-defined, well-thought and achievable: In most cases, utilisation scenarios are neither an ad-hoc, nor chaotic, or nor trial-and-error flows of work. They must not be fully defined. We have to understand to a greater or lesser extent what should be done and, especially, which instruments might be useful in which way on which grounds. The profile of an instrument is given by the goal we follow, by the means we could use for our goal (i.e. purposes of the instrument), and by the way how the instrument is going to be used according to the purpose (i.e. function of the instrument). Instruments shall be effective. From the other side, appropriateness of an instrument is also determined whether the goal is accomplishable.

Models are mental compilations of observed worlds: Models are a product of our thoughts. As a referent, we observe some situation in our world. Following the consideration by [16] on the three analogies by Platon (analogy of the cave, of the sun, and of the divided line), the referent recognises shadows in the observable world, builds some comprehension based on the thoughts and his/her intellect, and uses some language (not necessarily natural one; potentially some visual one) for reflection by terms, e.g. signs and images.

The background of models strikes through or is limiting reasoning: As already noted, models are often only given as the normal model while the deep model is implicit and the matrix of model application is commonsense in a discipline. As long as the deep model and the matrix are unconditionally acceptable and have not to be changed, the results of model functioning are reliable. There are, however, reasons to reconsider this background and these application frames. The potential and capacity of a model is restricted to these assumptions. Models are, however, not really context-free. They have their anti-profile also due to restrictions and their focus.

Evidential reasoning as initial point for model development: Evidential reasoning starts with evidences or glues and compiles guesses or conclusions, thus, providing hints about possible or likely conclusions with an explicit representation of uncertainty. Evidences, thus, support or refute hypotheses about the current status of the existing and observable situation. Unobservable propositions can be then determined on the basis of observable evidence, e.g. the observable data are used to reason on the almost unobservable real and micro-data. Evidential reasoning should be distinguished from causal reasoning which orients on explaining observable evidences by a hypothesised cause. It has its limits which should be integrated into this reasoning style.

Living in a world without necessity for a universal world formula — Almost plausibility and inherent incompleteness: Models have to be incomplete whenever they are based on the principles of reduction, decontextualisation, vagueness, and ignorance. Models used in some domain have not to be consistent. A property that is acceptable in this case is coherence what means that sub-models of models which express the same set of properties are compatible and partially homogeneous to certain degree (so-called non-adjunctive model suites) [14]. Inconsistency is handled in a controlled way by many-facetted coherence without integration. A classical example is Bohr's theory of atom and the system of Maxwell's equations. Paraconsistency treats a collection of models as consistently as possible without requiring full consistency. Model suites represent then some kind of 'knowledge islands' with partial bridge axioms.

It is surprising that neither form-follows-function, form-restricts-function, function-and-form-determine-techniques, nor inherent incompleteness and almost plausibility have been explicitly discussed in model theory and practice. A model property that is commonly accepted is well-formedness (some times called 'beauty', stronger well-defined) of models. They allow proper application of methods that support functioning of models.

4. Model-Based Reasoning Mechanisms

Model-based reasoning is completely different from classical logical reasoning techniques. It might use deduction. Models don't have to be true, consistent, fully integrateable with other models, based on a homogen understanding, at the most recent state-of-art, or acceptable by everybody. They can be certain to some limited extent, somehow coherent or even paraconsistent, heterogenous, representing islands without homogeneity, combine various generations of knowledge, or personal opinions. The two lists are not complete but demonstrate the difficulty to develop a sophisticated theory of model-based reasoning. Instead, let us consider some of the most essential reasoning procedures for models.

4.1. Plausible Reasoning

Models must not be complete and are considered within the given but changeable context. Models don't have to be true. They have to be useful and functioning

as instruments in the given scenario. Models focus and scope on certain parts while neglecting others, i.e. they are using approaches of ignorance. Therefore, they are typically incomplete. The premise set does not strictly allow for every conclusion with certainty. Model development is often based on inductive and evidential reasoning which is another source for incompleteness. Model-based reasoning has to additionally use more appropriate reasoning mechanisms. We especially use approaches based on plausibility and approximation.

Plausible reasoning stand for reasoning with uncertain conclusions for both certain or uncertain premisses. Typical forms are abductive, analogical, autoepistemic, counterfactual, default, defeasible, endorsement-backed, presumptive, and non-monotonic reasoning techniques.

The classical approach to plausible reasoning is given by the following schema:
The lack of soundness makes the conclusion plausible with a certainty below 1.0 based on evidence $CertF(\alpha|e)$ and reasonable (called believable) $ReasonF(\alpha|e)$ with reasonability below or equally certainty.
Certainty factors and reasonability factors may follow empirical rules to aggregate pieces of evidence, e.g.

$CertF(\alpha|e_1 \wedge e_2) = min(CertF(\alpha|e_1), CertF(\alpha|e_2))$ and
$CertF(\alpha|e_1 \vee e_2) = max(CertF(\alpha|e_1), CertF(\alpha|e_2))$

in the Dempster/Shafer reasoning style or in the possibility theory style.

These rules shall be applied in dependence on the context since they may lead to unpredictable, problematic, and counterintuitive results. Negation can be handled as negation-by-failure or in a multi-valued or paraconsistent form [25].

4.2. Approximative Reasoning

Models must not be precise although precision is necessary whenever models are used for automatic generation of solution from a given model [4,7,10]. Instead, model can be reduced, abstracted, truncated, imprecise, and raw. We follow the principle of parsimony and economy. Models must support efficient and effective thinking and actuation. The final and optimal solution might not exist at all or is infeasible both in time for its generation and in space for its presentation. Although, tools might not exist.

Approximation supports aggregative, generative, imprecise and robust reasoning. Approximate reasoning based on models is a common form to avoid complexity throwback due to over-detailing. Typical kinds are reasoning systems supporting aggregation and cumulation, generalisation and categorisation, imprecision, heuristics, robust thinking, and shallow consideration. These reasoning styles are used in daily life and especially for models, e.g. best characterised by the Austrian saying 'paßt schon' (fits somehow, fits already, close enough, suits, somehow convenient). In Computer Science, approximate algorithms provide a reasonable solution to problems at polynomial time instead of optimal solutions computable at (hyper-)exponential time in dependence on the problem complexity measured by Kolmogorov complexity [17]. The principle of Occam's razor orients on models as 'simple" as possible. We, thus, do not miss simple models. We may also use approximative rules with preference and simplicity in the Solomonoff style [8], e.g. for model-based explanation.

4.3. Hypothetical Model-Based Reasoning

Hypothetical model-based reasoning is based on the following schema:

1. Given a hypothesis model M that implies a statement E which describes observable phenomena.
2. The statement E has been observed as true.
3. The conclusion is that M is true.

The method of the hypothesis is not deductively valid because wrong hypotheses can also have real consequences.

Different assumptions are considered in order to see what follows from them, i.e. reasoning about alternative possible models, regardless of their resemblance to the actual world. Potential assumptions with their possible world conclusions assertions are supported by a number of hypotheses (allowing to derive them). Inductive model-based reasoning can be combined with abductive reasoning.

Hypothetical model-based reasoning restricts inductive reasoning by specific forms of inductive conclusions:

1. *statistical inductive generalizations*, in which the premise that x percent of observed A's have also been B's, so that the conclusion is, x percent of all A's are B's;
2. *predictive conclusions*, in which the premises are that x percent of the observed A's have also been B's, and a is an A and where the conclusion is that a is a B;
3. *direct conclusions*, in to which the premises are that x percent of all A's are also B's, and that a is an A and where the conclusion is that a is a B, and
4. *conclusion by analogy* in which the premises are that certain individual objects have the properties $F_1, ..., F_n$ and a have the properties $F_1, ..., F_{n-1}$, and where the conclusion is that a also has the property F_n.

4.4. Model-Based Explanation

The reasoning schema used for inductive reasoning can be extended to model-based explanation that describes, explains, illustrates, clarifies and characterises in a guiding way in a mediation scenario essential, central and in the given scenario important elements of complex origins in a comprehensible, concrete and coherent form for the recipient.

The reasoning schema for model-based explanation, elaboration, and comprehension can be defined as follows:
Given

some theoretical background $\mathcal{T}$ based on the context, background, knowledge, concepts, etc.,

a model class $\mathfrak{M}$ with orders for preference $\leqq$ and simplicity $\precsim$,

a deducability operator $\Vdash$ as an advanced operator for deductive, abductive, inductive, non-monotonic, approximative, and plausible derivation of conclusions, and

data under consideration $\mathcal{O}$ (observations) from the data space derived from input model suite data and prepared for analysis while being $\mathcal{T} \nVdash \mathcal{O}$ (non-trivial for $\mathcal{T}$) and $\mathcal{T} \nVdash \neg\mathcal{O}$ (not in conflict with $\mathcal{T}$).

The model $\mathcal{M}_E$ from $\mathfrak{M}$ is an **explanation model** for $\mathcal{O}$ within $\mathcal{T}$ if

it explains $\mathcal{O}$ within $\mathcal{T}$, i.e. $\mathcal{T} \sqcup \mathcal{M}_E \Vdash \mathcal{O}$ while being
non-trivial (or parsimonious) for $\mathcal{O}$, i.e. $\mathcal{M}_E \nVdash \mathcal{O}$ and
coherent with $\mathcal{O}$ and $\mathcal{T}$, i.e. $\mathcal{M}_E \nVdash \neg\mathcal{O}$ and $\mathcal{T} \sqcup \mathcal{M}_E \nVdash \bot$.

Based on the orders in $\mathfrak{M}$ we may be interested in the weakest (best) explanation model that is additionally parsimonious, i.e. $\mathcal{M}_E \nVdash \mathcal{O}$.

Model-based explanation is not consolidative modelling that uses the model as a surrogate for the system, for instance, by consolidating known facts about the system for purposes of analysis whether the model adequately represents the system. Model-based explanation explores how the world would behave if various models were correct. Many details and mechanisms of a system are uncertain. The model has not to be a reliable image of the world. Relevant 'ground truth' data for evaluating model may not be obtainable. We, thus, identify an ensemble of plausible models and modelling assumptions, identify the range of outputs predicted by plausible models under plausible assumptions, and identify the relationship between modelling assumptions and model outputs. A trick is to find assumptions that have a large impact on model outputs. Another trick is to identify predictions that are robust across different sets of modelling assumptions.

There is no methodological approach for derivation of a good explanation model. It seems that the embracement method is the strongest one. This method considers at the same time the generation of models from one side and the generation of partially explaining models from the other side. These partially explaining models can be seen as hypotheses which would form a good explanation model together with the generated model. A simple embracement method are Mill's methods of agreement, difference, joint methods of agreement and difference, residues, and concomitant variations.

4.5. The Model as Mediator in Empiric Model-Based Reasoning

Empiric investigation and reasoning is based on data spaces. Data are structured according to properties of parameters. It is often not possible to observe all parameters. We may distinguish outer parameters that can be observed and inner parameters that cannot be accessed or are not yet observed or are not yet observable. This separation is similar to genotype and phenotype observations. The problematic accessibility for inner parameters has already been discussed by Platon in his analogy of the cave (see, for instance, [16]). Development of an understanding based on outer parameters is a real challenge that is difficult to overcome. The data space should give an insight into potential quantitative observation concepts or conceptions. We need an insight into the data space for the inner parameters in order to reason on the reality situation.

Empirical reasoning starts with an investigation of data sources for the outer parameters and develops some quantitative observation concepts that might be embedded into a theory offer. A *theory offer* is a scientific, explicit and systematic discussion of foundations and methods, with critical reflection, and a system of

assured conceptions providing a holistic understanding. A theory offer is understood as the underpinning of technology and science similar to architecture theory [26] and approaches by Vitruvius [35] and L.B. Alberti [1]. Theory offers do not constitute a theory on their own, rather are some kind of collection consisting of pieces from different and partially incompatible theories, e.g. sociology theories such as the reference group theory, network theories, economic theories such as the agent theories, Darwinian evolution theories, subjective rationality theories, and ideology theories.

The main target is however, to form a theory that is based on the data, that is based on concept or conceptions, and that allows to draw conclusions on this theory. Concepts or conceptions to be developed should be qualitative and theory-forming. If qualitative concepts cannot be drawn then we need quantitative concepts that allow reasoning. Figure 2 displays this challenge. The challenge is solved if a number of functions exist.

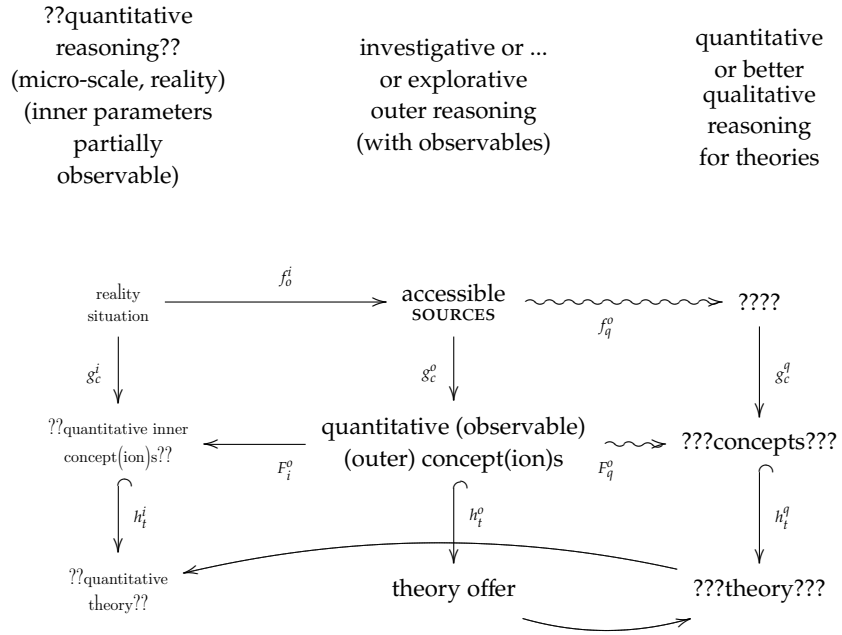


Figure 2. The challenge of empirical reasoning targetting on qualitative concepts and theory development with only partially known data for the inner parameters

The best solution for this challenge would be if we can map the inner parameters to the outer ones by a function f_o^i and use some kind of abstraction function g_c^o for association of data to concept(ion)s. In this case, we might succeed in constructing functions g_c^i resp. F_i^o from the reality situation resp. outer concept(ion)s to quantitative inner concept(ion)s. Then we could use the theory embedding of outer quantitative concept(ion)s h_t^o for construction of such inner functions h_t^i. This would also result in a coherence condition and a commuting diagram $F_i^o(g_c^o(f_o^i(situation))) = g_c^i(situation)$. We will be able to use the em-

bedding function h_t^c for construction of a corresponding supposition h_t^i for inner parameter theories.

The next step is a constrcution of a reasoning system. We use some aggregation function f_q^o for compiling sources in support of concepts by a function g_c^q and for embedding these concepts into a theory by h_t^q. If we succeed then we can use the theory offer for the construction of a theory for reasoning and as the next step for mapping this theory back the to inner quantitative theory.

We arrive therefore with the big challenge of empiric research: *How we can close the gap between quantitative theory offers and qualitative theories?*

This program for empirical reasoning is not really feasible. The construction of the functions is a higher-order challenge. Instead we can use model-based reasoning as displayed in Figure 3. The model is then used as a mediating means between qualitative and quantitative reasoning. The model is at the same time (1) a means, (2) a mediator, and (3) a facilitator [3], i.e. (1) an instrumentality for accomplishing some end, (2) a negotiator who acts as a link between quantitative and qualitative issues, and (3) an instrument that makes reasoning easier. Since models are more focused, we do not have to have fully-fledged functions. Instead, we can concentrate on the main issues. At the same time, we properly support qualitative reasoning based on our data spaces. This would also allow to formulate proper hypotheses from the model world to the quantitative world. The validity power of the model would then support qualitative reasoning.

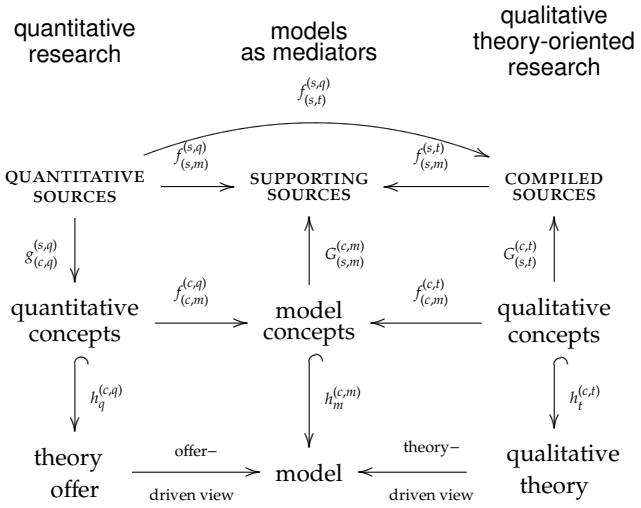


Figure 3. Models as integrating and mediating instrument in empiric research

In this case, we may succeed in constructing insights that go far beyond data-backed reasoning, e.g. in data science. We could then also construct massives of supporting sources for our models. The model accommodates the quantitative theory, the theory offers, and the qualitative theory.

Theories can be built on the basis of theoretical concepts which are supported by sources. Quantitative concepts should be associated with qualitative concepts. The association can only be developed in the case when the association among the data has been clarified. So far, the explanations that can be generated are mainly developed for explaining the observations made on the basis of outer data.

4.6. Meta-Model Reasoning Used for Model-Based Steering

Thinking in models should be supported by a systematic methodology. Model-based steering use meta-models (i.e. *steering models*) as a guiding or motivating model that directs the direction of model-based reasoning. This kind of meta-reasoning enables us to explore potential opportunities in the opportunity and possibility spaces. We arrive at some proposal as a result of model-based reasoning, i.e. putting forward or stating something for consideration by making or offering a formal plan or suggestion for acceptance, adoption, or performance. The *proposal model* is used in a second step as an additional origin. The trick we use is then based on second-order modelling. We know already *governing models*. Such meta-models make and administer the selection of opportunities based on possibilities, regulate, and control model-based reasoning while keeping exploration under control. They exercise a deciding or determining influence on selection and thoughts.

As a result of steering and governing, we obtain an *advice model* as a recommendation regarding a decision or course of conduct. It could be considered to state an opinion about what could or should be done about a situation or problem, what is going to be recommended offered as worthy to be followed. Advice models are used as a counsel and denote an opinion as to a decision or course of action.

We may use meta-techniques such as specific question-answer forms (or, more specifically, query-answer or input-output forms [9]). These question-answer forms have their inner meta-structure and inner meta-flow that could be used in investigative research, e.g. what-if analysis, what-would-be-if, 5-why-drill-down, rolling-up distancing, context-enhancement, assumption-slicing with attention restriction, why-it-must-be, why-this-question-and-not-other, why-not-rephrasing, observation-in-context dependencies, immersion-into-context, why-finish, how-we-can-know, why-this-question, question-reformulation by opening or closing the parameter space, and parameter-space-reduction by dicing with tolerance of errors, e.g. by principal component analysis. Essentially, these meta-techniques are steering models.

We use the steering model for driving into a problem space and detecting opportunities and possibilities. Sciences, engineering and daily life are full of such 'wisdom' techniques.

This approach can be generalised to meta-models for research, i.e. moulds[17]. Methodologies are simple moulds. They provide a guidance for a flow of work. Frameworks are complex moulds that can be adapted to the given situation[18]. Civil engineering uses moulds as frame on which something can be constructed.

Steering models are used to control or to direct or to guide the course of actuation. They set, follow, pursue, and hold to a course of action and reasoning and especially a hint as to procedure. Steering models are used as a piece of advice or information concerning the development of a situation. They allow to control a situation so that it goes in the direction that you want. They enable to take a particular line of action. Such meta-models are models of the models, of the modelling activities, and of the model association within a model suite. Their goal is to improve the quality of model outcomes by spending some effort to decide what and how much reasoning to do as opposed to what activities to do. It balances resources between the data-level actions and the reasoning actions. A typical case is design of activities in data mining or analysis [11] where agents are preparation agents, exploration agents, descriptive agents, and predictive agents. Meta-models for a model suite contain decisions points that require macro-model control according to performance and resource considerations. This understanding supports introspective monitoring about performance for the data mining process, coordinated control of the entire mining process, and coordinated refinement of the models. Meta-level control is already necessary due to the problem space, the limitations of resources, and the amount of uncertainty in knowledge, concepts, data, and the environment.

Steering models extend the origin' collection (see Figure 1) by meta-reasoning origins. These origins enable us to use second-order cybernetics [34], i.e. to continuously reason on insight we got in previous steps and to change our mind whenever new insight has been obtained. The methodology follows a mould of continuous changing method application.

This approach is based on control by meta-reasoning [6] as displayed in Figure 4. We distinguish the activity layel that is based on methods ground level, the meta-modelling level that contains the modelling methods level and rules the selection of actions, and the meta-meta-modelling level that contains the abstraction to meta-modelling methods and controls the middle level section. This approach is similar to government and binding [5] where the utterance playout is based on a second layer playout selection for the utterance that is again ruled by controller for settling the kind of utterance and its general form.

[17] A *mould* is a distinctive form in which a model is made, constructed, shaped, and designed for a specific function a model has in a scenario. It is similar to mechanical engineering where a mould is a container into which liquid is poured to create a given shape when it hardens. Im Mathematics, it is the general and well-defined, experienced framework how a problem is going to be solved and faithfully mapped back to the problem area.

[18] For instance, analytical solution of differential equations use a set of solution methods formulated as ansatz. Database development can be guided by specific frameworks. Artists and investigative researchers are guided by moulds, i.e. by steering or governing models. Software engineering is overfull of meta-models for design, development, and quality management.

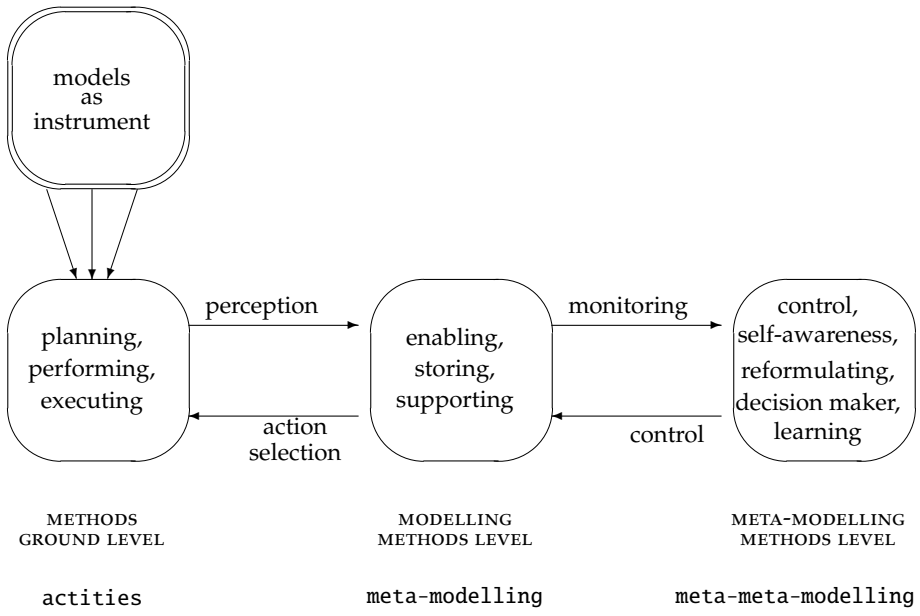


Figure 4. The meta-control mould for meta-reasoning

5. Conclusions for 'Modellkunde' – The Study of Models and Modelling

Models must not be true or consistent. They should be useful as instruments in application scenarios. Usefulness presupposes the existence of techniques for model utilisation. Reasoning is one kind of technique. Reasoning is a daily life practice that is rarely based on deductive systems. Instead, induction and abduction are used for model-based reasoning. These reasoning techniques are especially useful for models since we do not require consistency and truth maintenance.

We demonstrated the power of such techniques by the generalisation to plausible, approximative, hypothetical, and explanatory reasoning. These mechanisms are really sophisticated. For instance, pattern recognition and Modellkunde for pattern can be based on explanatory model-based reasoning in combination with mediator approaches. One of the best achievements is mediator-based reasoning that allows to overcome pitfalls of middle-range theories and badly associated theory offers in empiric research. Instead we use models as a mediating device between empiric and qualitative reasoning. The utilisation of models may be governed by other models. Steering meta-models guide application and usage of models.

This paper can be extended by application of other model-based reasoning techniques such as separation and concentration of concern, playing with ignorance and de-contextualisation, qualitative techniques used for data analysis, and probabilistic calculi.

This paper has been centred around reasoning techniques and model-based reasoning. There are many other techniques beside reasoning that can be applied

to models. Typical techniques are enhancements similar to conceptualisation, model inheritance from generic or reference models, parameter hardening used for inverse modelling in physics, model-based checking and control for systems, and simulation of behaviour for some of the parameters. Cognitive modelling is another technique that has been left out for this paper. Shallow and deep reasoning techniques are another lacuna for the study of models.

The study of models has to considers also other techniques for model utilisation. Models form a landscape. Some models are partially isolated. These isolated models should be supported by bridging techniques. Models are focused and have, thus, their abstraction level. Model-based problem solving use, therefore, also techniques for generalisation and governed specialisation.

Models can also be composed in vertical or horizontal layering. Models can be also origins for other models. The composition should supported by techniques similar to nested data warehousing, i.e. roll-up, drill-down, dice, slice, rotate, algebraic construction, peaceful renovation and updating, unnesting, and nesting. Models may consists of a well-associated collection of models, i.e. of a model suite. Association techniques allow also management of coherence of models in a model suite. Some models in a model suite may play the role of a master (or order) model while others are slaved. The model suite should, thus, be enhanced by control models for adaptation of the master models to a given situation, e.g. in inverse modelling.

References

[1] L.B. Alberti. *On the art of building in ten books*. MIT Press, Cambridge. Promulgated in 1475, published in 1485, 1988.

[2] A. Aliseda-Llera. *Seeking explanations: Abduction in logic, philosophy of science and artificial intelligence*. PhD thesis, Universiteit van Amsterdam, 1997.

[3] C. Blättler. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*, chapter Das Modell als Medium. Wissenschaftsphilosophische Überlegungen, pages 107–137. De Gruyter, Boston, 2015.

[4] E. Börger and A. Raschke. *Modeling Companion for Software Practitioners*. Springer, 2018.

[5] N. Chomsky. *Some concepts and consequences of the theory of government and binding*. MIT Press, 1982.

[6] M.T. Cox and A. Raja, editors. *Metareasoning - Thinking about Thinking*. MIT Press, Cambridge, 2011.

[7] A. Dahanayake, O. Pastor, and B. Thalheim. *Modelling to Program: Second International Workshop, M2P 2020, Lappeenranta, Finland, March 10–12, 2020, Revised Selected Papers*, volume 1401 of *CCIS*. Springer Nature, 2021.

[8] David L. Dowe, editor. *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, volume 7070 of *LNCS*. Springer, 2013.

[9] H. Jaakkola and B. Thalheim. Supporting culture-aware information search. In *Information Modelling and Knowledge Bases XXVIII*, Frontiers in Artificial Intelligence and Applications, 280, pages 161–181. IOS Press, 2017.

[10] H. Jaakkola and B. Thalheim. Models as programs: The envisioned and principal key to true fifth generation programming. In *Proc. 29'th EJC*, pages 170–189, Lappeenranta, Finland, 2019. LUT, Finland.

[11] K. Jannaschk. *Infrastruktur für ein Data Mining Design Framework*. PhD thesis, Christian-Albrechts University, Kiel, 2017.

[12] M. Kangassalo and E. Tuominen. Inquiry based learning environment for children. In *Information Modelling and Knowledge Bases XIX*, volume 166 of *Frontiers in Artificial Intelligence and Applications*, pages 237–256. IOS Press, 2007.
[13] R. Kaschek. *Konzeptionelle Modellierung*. PhD thesis, University Klagenfurt, 2003. Habilitationsschrift.
[14] R. Kauppi. Einführung in die Theorie der Begriffssysteme. *Acta Universitatis Tamperensis, Ser. A, Vol. 15, Tampereen yliopisto, Tampere*, 1967.
[15] Y. Kiyoki, T. Kitagawa, and T. Hayama. A metadatabase system for semantic image search by a mathematical model of meaning. *SIGMOD Rec.*, 23(4):34–41, 1994.
[16] C. Lattmann. *Vom Dreieck zu Pyramiden - Mathematische Modellierung bei Platon zwischen Thales und Euklid*. Habilitation thesis, Kiel University, Kiel, 2017.
[17] M. Li and P.M.B. Vitanyi. *An introduction to Kolmorov complexity*. Springer, New York, 2008.
[18] L. Magnani, W. Carnielli, and C. Pizzi, editors. *Model-Based Reasoning in Science and Technology: Abduction, Logic, and Computational Discovery*. Springer, 2010.
[19] B. Mahr. *Visuelle Modelle*, chapter Cargo. Zum Verhältnis von Bild und Modell, pages 17–40. Wilhelm Fink Verlag, München, 2008.
[20] B. Mahr. Information science and the logic of models. *Software and System Modeling*, 8(3):365–383, 2009.
[21] B. Mahr. Modelle und ihre Befragbarkeit - Grundlagen einer allgemeinen Modelltheorie. *Erwägen-Wissen-Ethik (EWE)*, Vol. 26, Issue 3:329–342, 2015.
[22] C.S. Peirce. *The Collected Papers of Charles S. Peirce,* (C. Hartshorne, P. Weiss, and A. W. Burks (eds.)). Cambridge: Harvard University Press, 1931-1966.
[23] K. R. Popper. *Logik der Forschung*. J.C.B. Mohr (Paul Siebeck), Tübingen, 10th. edition, 1994.
[24] M. V. Rodrigues and C. Emmeche. Abduction and styles of scientific thinking. *Synth.*, 198(2):1397–1425, 2021.
[25] K.-D. Schewe and B. Thalheim. NULL value algebras and logics. In *Information Modelling and Knowledge Bases*, volume XXII, pages 354–367. IOS Press, 2011.
[26] G. Semper. *Die vier Elemente der Baukunst*. Braunschweig, 1851.
[27] H. Simon. *The Sciences of the Artificial*. MIT Press, 1981.
[28] H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.
[29] B. Thalheim. The conceptual model ≡ an adequate and dependable artifact enhanced by concepts. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 241–254. IOS Press, 2014.
[30] B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, pages 123–139. Springer, 2019.
[31] B. Thalheim. Semiotics in databases, keynote paper. In *Proc. MEDI2019, LNCS 11815*, pages 3–19. Springer, 2019.
[32] B. Thalheim and A. Dahanayake. Comprehending a service by informative models. *T. Large-Scale Data- and Knowledge-Centered Systems*, 30:87–108, 2016.
[33] B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.
[34] S. A. Umpleby. Second-order cybernetics as a fundamental revolution in science. *Constructivist Foundations*, 11(3):455–465, 2016.
[35] Vitruvius. *The ten books on architecture (De re aedificatoria)*. Oxford University Press, London, 1914.
[36] M.W. Wartofsky. *Conceptual foundations of scientific thought*. MacMillian, New York, 1968.

Remark: More detailed information on our research papers can be found on research gate in collections at https://www.researchgate.net/profile/Bernhard_Thalheim .

See also the youtube channel "Bernhard Thalheim", videos at https://vk.com/id349869409 or https://vk.com/id463894395 (in German).

About the Essence of Intelligence - Will Artificial Intelligence (ever) Cover Human Intelligence?

Hannu JAAKKOLA[a 1], Bernhard THALHEIM[b] and Jaak HENNO[c]
[a] Tampere University, Finland
[b]Christian Albrecths University at Kiel, Germany
[c]Tallinn University of Technology, Estonia

Abstract: Computers were originally developed for executing complex calculations fast and effectively. The intelligence of computer was based on arithmetic capabilities. This has been the mainstream in the development of computers until now. In the middle of 1950s a new application area, Artificial Intelligence (AI), was introduced by researchers. They had interest to use computers to solve problems in the way intelligent beings do. The architecture, which supported calculations, were conquered to perform tasks associated with intelligence beings, to execute inference operations and to simulate human sense. Artificial intelligence has had several reincarnation cycles; it has reappeared in different manifestations since this research area became interesting for the researchers. All the time a lot of discussion about intelligence of these systems has been going on – are the AI based systems and robots intelligent, what is the difference of human and machine intelligence, etc. Abilities related to intelligence cover ability to acquire and apply knowledge and skills, as well as ability to learn. AI provides different manifestations to the term "intelligence": the human intelligence is a wide variety of different types of intelligence, as well as the meaning of artificial intelligence has varied over time. In our paper we will look to this term, especially to provide means for comparing human and artificial intelligence and have a look to the learning capability related to it.

1. Introduction

Artificial intelligence (AI) is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings[2]. *Intelligence* (synthesis of several sources) is defined to be the ability to acquire and apply knowledge and skills; the ability to learn, understand and think in a logical way about things; the skilled use of reason; the ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria.[3] Key aspects related to intelligence are ability to apply knowledge, *reasoning, learning capability, ability for abstract thinking and the aim to use intelligence to affect to something*. Artificial intelligence instead *"simulates" human intelligence*. What are the differences and similarities of this to kinds of intelligence? This is the starting point of our paper.

Human Intelligence (HI) is manifold, so is Artificial Intelligence (AI). AI has certain importance in ICT because it is one of the current emerging technologies. It is also an

[1] Corresponding Author: Hannu Jaakkola, Hannu.jaakkola@iki.fi.
[2] Britannica. https://www.britannica.com/technology/artificial-intelligence
[3] Oxford: https://www.lexico.com/definition/intelligence; https://www.oxfordlearnersdictionaries.com/; Merriam-Websterhttps://www.merriam-webster.com/dictionary/intelligence

example of recurring technologies, which has time by time reappeared in waves. The characteristics of intelligence have been changed over the waves, as well as the driving force and the opportunities provided by it. The definition in [35] compares AI and HI: "Artificial intelligence (AI) is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals, involves *consciousness and emotionality*"; this definition points out something that AI is not able to handle (yet).

The term "Artificial Intelligence" was coined by John McCarthy in 1955. Late in 1950s he introduced the programming language Lisp, which provided means for developing computer programs having ability for self-modifying code dynamically in run time. Computers were developed to conduct complex calculations; this architecture had to be conquered to support tasks related to intelligent systems by the software level support; Lisp was the first effort in this area. The dynamic modification of the code implemented a primitive *learning capability* in the time (1950s) when processing power of the computers was low and availability of data to process was limited and access to it was slow.

AI systems create knowledge from a variety of data elements, based on its built-in "intelligence". The DIKW (Data, Information, Knowledge, Wisdom) pyramid in Figure 1 illustrates the cultivation process from data to wisdom (modified from [36]):

- *Data* is conceived of as symbols or signs. To simplify, data is representation of something having itself no exact meaning or interpretation.
- Information affiliates semantics or meaning to the data.
- Knowledge is processed or organized from information based on the rules or algorithms and make information utilizable in the use context.
- Wisdom is applied knowledge in its use context.

The role of *intelligence* is connected to the cultivation process introduced above. We need intelligence to create meaning for data (-> information) and further to handle the information to create useful knowledge about it. Wisdom provides guidelines for utilizing the knowledge and transfers it to actions and behavioral patterns

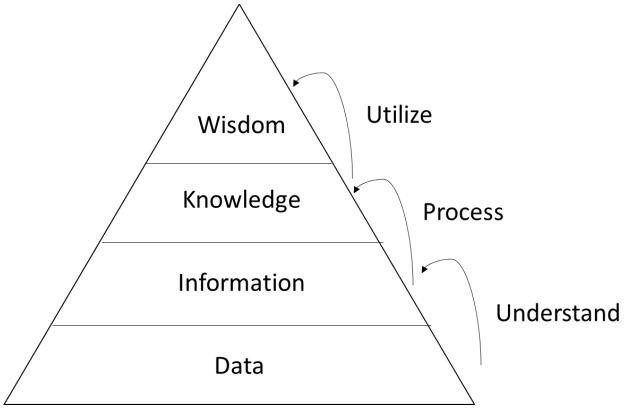


Figure 1. DIKW Pyramid

DIKW pyramid can be connected to the human and artificial intelligence. In both cases we need *data* to handle and the (*intelligent*) *process* to refine it to the applicable form in the use context. Human way and machine way are different. How different, this will be discussed in the following Sections. Intelligence, in addition to data processing capability also needs capability to *learn* – ability to change the future behavior based on the experiences or external knowledge available.

The aim of this paper is to find (at least partial) answer to the questions:

- What is Intelligence?
- What are the differences between HI and AI?
- What is the intelligence of AI?
- What are the key elements of (Artificial) Intelligence?

The paper is structured to find answer to the questions in the following way . *Section 2* handles the characteristics of HI, compared to AI. *Section 3* has focus it the intelligence of AI – what are the functionalities to implement intelligence in these systems. Section 4 handles two key aspects of intelligence, communication, and learning. Section 5 concludes the paper.

2. Human Intelligence

2.1. The Potential of AI and HI

Human intelligence is far broader than the artificial one. According to [14], we may distinguish three kinds of human reasoning systems: brain-based central nervous system with reasoning, the partially autonomous vegetative nerve system with body system control, and the governing survival nerve system for reproduction. AI research is centered around the first system and only concentrates on one type of intelligence: creative or problem-solving intelligence. There are, however, four other kinds of intelligence that we cannot (yet) support: emotional (or social) intelligence, self-reflection (or spiritual or existential) intelligence, body intelligence as second human reasoning system, and survival intelligence. These four types are supporting and partially governing by the central nerve systems while interacting with it.

We may distinguish several specific types of creative intelligence that are so far partially covered and not yet well-supported by AI research:

- linguistic, narrative, or verbal intelligence including metaphorical intelligence,
- musical intelligence,
- abstract intelligence including analytical intelligence, logical-mathematical intelligence, and numerical intelligence
- visual intelligence
- practical intelligence including application intelligence, and practical wisdom,
- imaginative intelligence,
- physical-kinesthetic intelligence, and

- spatial intelligence.

Creative intelligence also includes intuitive intelligence including crystallized intelligence. This specific type cannot be so far supported at all.

Thesis 1a: *HI can only be properly supported by AI if the specific kind and type of intelligence is well-understood.*

Intelligent human behaviour requires above all a great deal of knowledge about details. Knowledge is to be distinguished from intelligence. Intelligence operators (such as the very large variety of inferential reasoning) allow us to derive new knowledge or at least insight from knowledge, experience, observations, models, and intuition. We often hypothesise in 'unknown territory' and develop hypotheses, models or even theories. Detailed knowledge is helpful to 'falsify' hypotheses, i.e. to eliminate them as certainly false, if they contradict facts or experiences.

Consciousness, learning, creativity, freedom, communication behaviour are not (yet) understood in an algorithmic fashion, e.g. decision making under restrictions, competing simultaneous objectives and uncertainties about the future.

Currently, the AI hype aims at development of deep learning mechanisms. This is however only the first step towards emulation of HI. Consciousness, feeling and other topics, such as creativity or will, must be understood before becoming supported by AI.

Thesis 1b: *AI and IT may far better handle the regular and typical case. Anything else is beyond the horizon.*

We discover that Turing-based computation is mainly based on an algorithmic treatment by deductive systems. It is an incarnation of the digital. Turing-based computation is limited by the second Rice's theorem [25] that has extended by many non-computability and undecidability results. It shows that with current technology it is, thus, impossible to build safe and secure software systems. Advanced reasoning mechanisms such as induction and abduction are not yet well-supported. We need a more sophisticated support for reasoning. For instance, logics research showed that there are true properties in Arithmetics that cannot be proven by deduction. The Turing machine model is not at all the only kind of computation. Analogical, plausible, and approximative computation is badly covered by Turing computation. Neural networks are so far very simplistic networks. The web of neurons in living systems is far more sophisticated and provides advanced computations that cannot yet emulated by our machines. There exist many kinds of computations (deterministic, non-deterministic, randomized, quantum) each of which is characterized by a class of computationally equivalent mechanisms. This is also the case of cognitive systems which are but specialized non-uniform evolutionary computational systems supplied by information delivered, thanks to their own sensors and effectors, from their environment. There is evidence that the computational power is human intelligence might be bounded by the $\sum_2$ level of the arithmetic hierarchy [34] what is far beyond computability.

Thesis 1c: *AI needs an IT that goes far beyond Turing-based computation and logical reasoning by deduction and is not only based on digitalisation.*

2.2. Some Limitations of Artificial Systems

Modern IT systems are thriftless in their energy requirements. One may wonder why a complete operating system must be loaded before using the computer as a typewriter. Human systems are energy efficient. AI systems are oriented on luxurious featuritis with many tools that cannot be handled by a singleton human. Additionally, human reasoning is energy-minimalised while AI computation as well as any kind of current computation is energy extravagant and recklessly wasteful.

Humans use also other reasoning systems. These systems are parsimonious and energy-optimised what is not the case for current AI systems. They form some kind of super-organism that are 'living' structures whose ability to survive depends on appropriate coordination of the interaction of individual systems, which are themselves viable, just as a human being is made up of billions of living cells. Self-organisation and self-optimisation usually goes through an (evolutionary) process and in the process must be 'attractive' enough for the individual elements involved to join together to form a whole.

Thesis 2a: *HI follows the utility value paradigm in a parsimonious manner whereas AI is mainly based on the marketable value paradigm in a lavish manner.*

We are used nowadays to the goodies of modern AI solutions such as smartphones. At the same time we do not realise that we sacrifice cultural achievements necessary for HI. A good example is the lost education depth due to dominance of technical solutions such as Wikipedia or other web services which reduce knowledge to what the monopoly player considers as opportune. Anything else vanishes.

Thesis 2b: *The monopoly game played by the big internet players is oriented on the brainsickness of simple direct consumers. AI monopoly is not interested in intelligent consumers. HI is an evolutionary advantage that is based on co-evolution within all kinds of intelligence.*

Living beings and human systems are concentrating on the main function. Think, for instance, the albatross can fly thousands of kilometres several days without interruption and intermission. Evolution of the fittest led to such abilities. This development is some kind of meta-evolution rule that uses principles and paradigms we do not know for technical systems.

Thesis 2c: *Nature and evolution are oriented on the selection of the fittest and best accommodating system. AI system evolution is – at its best - based on meta-models of evolving systems what must be developed in advance.*

2.3. The Opportunities Offered by Artificial Systems

Technical system are often far better than any human. Compare, for instance, a highly skilled chess player with a computation system that considers all potential moves and that adapts the position evaluation function after assessing the position quality and comparing it to different positions.

Thesis 3a: *AI and HI are different methods of solving certain difficult tasks. AI here does not include the four other kinds of human intelligence. In terms of intelligence in the sense of problem-solving quality in non-trivial subject areas, technology is already further ahead than humans in some areas today.*

AI systems can easily learn regular and smooth functions with a limited number of discontinuity points. Such functions can be considered as routine and approximative solutions. Beside this 'normality' we approach the unlearnable. Consider, for instance, a Cauchy function that is neuronally unlearnable. In general, non-learnability applies to all functions that require quite different results or actions for the smallest variations of parameters. So, a neural network of any depth cannot learn 'ugly' functions, but it can learn smooth ones [29].

Thesis 3b: *AI systems are better in support of regular and good application tasks but fail in complex domains which are badly understood, irregular, or insufficiently modelled.*

IT systems as well as AI are instruments that might ease human life whenever this is beneficial. As instruments, they can be used with proper intent, with care, with a belief in reliability and robustness. Modern life is impossible without all these instruments. However, we do not need instruments on any place, at any time, for any task, for everybody, and in any environment. Often we cannot survive without our instruments.

Thesis 3c: *AI systems are instruments that are great in hostile to life environments, for support of activities far beyond human skills, more efficient and effective than humans, and provide services that ease life.*

2.4. The Risks and Threats of the Artificial

IT and AI have a high innovation rate that is far from being easily understandable. The society is unable to capture the impact by rules, regulations, or laws. Think, for instance, threats imposed by micro-trading or AI weapons. The insight into such systems vanishes. Who can repair a car? Garages are also relying on support instruments. Systems are built with 'AI inside'. IT systems operate in a form that CS people have no chance to fully understand their operating. The 'race between education and technology' seems to have the artificial system as a winner. Look on the smartphone behaviour that substantially decreases human reasoning, understanding and reasoning, blindness for environment observation, and human social warmth. With the advent of internet and modern TV technology we detected media competence as an essential skill for everybody (e.g. see [23]). Nowadays we have to develop literacy and competence for AI systems.

Thesis 4a: *AI education is still in a fledgling stage. Similar to the impact of techniques and technologies to education and disciplines we have to develop a proper technology and disciplinary education for AI.*

Modern applications and environments are becoming more complex, advanced, and sophisticated. This complexity is often far beyond comprehension skills of humans. The reaction speed of technical systems beets human abilities. Programmers program whatever is requested without being restricted by ethical restrictions. Technology

regulates human behaviour. [8] observes how AI is currently intentionally misused for destroying any democratic society and to nudge any human behaviour up to daily life issues. In near future, systems combining a group of humans and thousands of 'intelligent' computers are going to transform humanity to hybrid human-technics super-systems that will not be properly controllable or limitable.

Thesis 4b: *Artificial systems create a threat to human existence and must become changeable whenever systems start to command humanity.*

Modern systems make humans lazy. Who is nowadays not relying on a navigation system and prepares in advance a highway trip with maps? Human intelligence regresses without being continuously challenged since biological systems optimise themselves and thus reduce reasoning if it is not requested. Evaluation algorithms and computational systems do not follow ethical principles. Computers and programmers become ethics-free. Political systems are far too slow and too sloppy and cannot handle such challenges. Information overflow and pollution by senseless services make the human being a plaything for the big players (see [27]). The human 'laziness', loose of tacit background knowledge, and resulting lack of education is resulting in AI dependence and debility similar to 'illiteracy'. The software crisis is a crisis of proper program and software development culture. Nowadays, we have a data crisis, a (large and embedded) system crisis, an infrastructure crisis, and an energy crisis. The next crisis we can expect is the AI crisis. Sophisticated systems such as AI systems operate without feelings, without heart, without compassion, without conscience, and without ethics.

Thesis 4c: *The forthcoming AI crisis can only be tackled if we consider the end from the beginning and if we develop a proper culture of coevolving and collaborating symbiotic intelligent systems.*

2.5. The Qualia Question

HI and AI are two kinds of 'intelligence' that have to coexist. The first one is not really well-understood, the second one is human-made for improvement of life. AI cannot mirror HI. AI is currently mainly based on programs and meta-programs made by humans within the human understanding of that moment when the programs have been developed. Programs can be based on meta-programs that change the code according to change scenarios foreseen in advance. Therefore, HI and AI capabilities are different and will be different in future.

Thesis 5a: *HI and AI coevolution and symbiosis are encouraging and are a resource for prosperity that should be used wisely. They will give us wings to better life if properly designed, managed, and handled with care and proper wisdom.*

We discovered that HI and AI are two very different kinds of 'intelligence'. Artificial 'knowledge' systems such as Google, Twitter and Wikipedia are also intentionally used for misguidance. They form their own ecosystem that goes beyond human understanding. From the other side, HI is also based on model-based reasoning. Mental models are something like the 'third eye' in our human, emotional, experience-backed, intuition-guided, and hormone-driven digestion of the observed environment. Models

are also used for context-backed and culture-based human interaction. Of course, they are also a means for language-based development of the artificial or of IT solutions. HI is properly supported by models at any abstraction level. The mechanisms of model-based reasoning are not understood but we may consider models to be the fourth sphere of our understanding beside our understanding and handling of our natural environment, science, and technology. The mismatch between model-based reasoning and AI model handling remembers the 'lost in translation' problem,

Thesis 5b: *The mismatch between HI and AI is also caused by human model-based reasoning abilities that go far beyond what can be formally handled and managed.*

Many researchers claim that machine intelligence and neural networks are going to cover human capabilities and might replace human reasoning. There are limits and boundaries of current computational approaches that are essentially state transformations and based on Turing style computing. Human reasoning is far more advanced. Behaviouristic detection of brain activities uses rather naive models and assumptions how the brain works. Furthermore, AI reasoning systems are bound by our current logics approaches. Logical deduction calculi already cover revisable and non-monotonic derivation. Human inductive, abductive, approximative, plausible, and model-based reasoning is far more advanced. They must not be language-based. Humans use instruments beyond languages.

Thesis 5c: *Neural networks used in AI are based on the neuron models developed in the 1950ies. The next generation of neuro machines needs decades of advanced research in order to reach maturity of brain-based central nervous system with reasoning. The other human reasoning systems might be understood in the next century.*

Additionally, HI is one kind of natural intelligence. There is no reason why HI should be the only form of intelligence of living beings. Furthermore, the human body consists of many synergy-stimulated systems where human cells are the most essential part of this system. The human cell system cannot survive without the other systems. The other systems are currently really badly understood. The interaction in such systems is a 'black hole' in medicine.

3. Artificial Intelligence

3.1. Evolution of the AI and the role of enabling technologies

AI is one of the technologies having *recurring appearance* in the role of emerging technologies. Emerging technologies are "*technologies that are perceived as capable of changing the status quo*" [10]. Emerging technologies have a radical novelty and potential for fast growth and impact, but under uncertainty; the progress may sometimes be different than expected (hype phenomenon).

The evolution of AI has been highly dependent on the progress of *enabling technologies* (Jaakkola et al. 2017):

- *VLSI Technology –Processing Capacity* doubles every 18 months and *Memory capacity* of computers every 15 months.

- *Mass memory capacity* (magnetic devices) increases by a factor of ten every decade, i.e. it is doubling every 18 months.
- *Data transmission* capacity speed doubles every 20 months. This dimension is a bit complicated because of a heterogeneity of transmission channels and their role in the whole. Data transmission capacity is the key issue in the adoption of distributed solution in information processing.

We agree that the forecasts above are not scientifically exact but provide rough trend about the progress in the key technologies related to AI. We have extrapolated the progress backwards from late 1960s (invention of microprocessors and LSI) to the era of early computers (1950s). The progress is summarized in Table 1.

Table 1: Progress of AI related enabling technologies.

Double capacity in months (m)	1955	1975	1990	2020	2030
Computing 18m	1	2^{13}	2^{23}	2^{42}	2^{50}
		1 →	$(2^{10}) = 1$ →	$(2^{19}) = 1$ →	(157)
Memory 15m	1	2^{15}	2^{28}	2^{51}	2^{60}
		1 →	$(2^{12}) = 1$ →	$(2^{23}) = 1$ →	(445)
Mass memory 18m	1	2^{13}	2^{23}	2^{42}	2^{50}
		1 →	$(2^{10}) = 1$ →	$(2^{19}) = 1$ →	(157)
Transmission 20m	1	2^{12}	2^{21}	2^{38}	2^{49}
		1 →	$(2^{9}) = 1$ →	$(2^{17}) = 1$ →	(97)

The years selected in the table represent the different eras of AI. The next ten years the progress is continuing and provides new means to the future of AI: 157-fold computing power, 445-fold memory capacity, 157-fold mass memory capacity and 97-fold higher data transmission capacity compared to the situation today.

Artificial Intelligence has been in the continuous interest of people since its existence. That is why new approaches are born cyclically in a kind of "*reincarnation cycles*". These cycles can be explained (at least) by the following three factors:

- *Demand pull*: there is continuous (hidden) demand of new (more) intelligent applications. People expect more and more intelligent applications to help their daily life and to improve the productivity of their work.
- *Technology gap*: the performance of the existing technology limits the opportunities to implement applications that the users would like to have.
- *Technology push*: when technology allows, the demand pull "activates" the new type of applications – new cycle starts.

Additional aspects having importance in the progress of AI applications come from general trends observable in ICT infrastructure: transfer to mobile and wireless, distributed processing and data management, transfer towards more complex user

interface technologies, growing interoperability of applications, embedding of (AI) solutions, growing role of "robotics" (IoT), etc.

3.2. Reincarnation Cycles of Artificial Intelligence

Figure 2 provides a general overview to the four reincarnation cycles of AI. To be exact, there should also be additional one – the era of *Ancient AI*. AI has roots in antiquity in the form of myths, stories and rumors of artificial beings endowed with intelligence or consciousness by master craftsmen [37]. However, the *ancient AI* (Cycle 0) left the ideas at a theoretical and story level. As discussed in the beginning of this paper, the real AI is based on the ability to cultivate (currently masses of) data to the user's wisdom and help him to fill the goal set to the (intelligent) data processing. This has been enabled by the computers. However, a lot of ancient philosophical foundations (theories about the human mind and human way of thinking) are useful as a theoretical foundation in the current AI research.

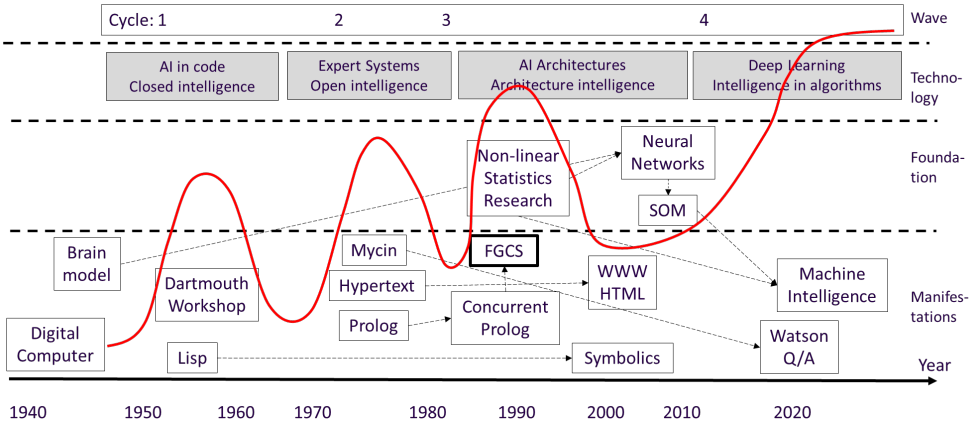


Figure 2. The four cycles of AI

The four waves (cycles, eras) provide view to the spread of AI in new kind of application logic and ability to apply new kind of intelligence in systems developed. Typical to the wave-based approach is that every wave as an emergent technology has slowly growing *embryonic phase* in the beginning, phase of the fast growth until reaching *the peak* (highest importance) and turning slowly to the *decline phase*, which runs it to be a "part of normal" without meaningful innovative power. Typically, it is the beginning a new wave based on the new technology replacing the old one and taking its role as an emergent technology; this leads to the sequence of waves as in Figure 2. This aspect of technology analysis is handled in several papers of the authors, see e.g. [11; 12].

The First Wave – AI in program code, from 1950s to 1970s

The term "Artificial intelligence" was introduced by Professor John McCarthy[4] in 1955. He was a key person in organizing the Darthmouth workshop[5]. It was a summer school, which provided forum for brainstrorming for a dozen of scientists about the novel technology a research topic of "thinking machines". The workshop proposal [16] proposed to proceed the study "*of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves* ".

Later in 1950s McCarthy introduced *Lisp language*[6] (acronym of LISt Processor), which became the first tool to develop "real" AI applications. Lisp is based on Lambda calculus and allows code to modify itself in runtime. This creates a simple learning capability to the applications. Lisp has, since its birth, many dialect implementations, which have followed the general trends of improvements in programming languages.

Another remarkable finding in AI programming is the *logic programming language* Prolog[7] developed by Alain Colmerauer and Philippe Roussel. Prolog is based on first-order logic, a formal logic, in which the program is declared in the form of relations "if-this-then-this" (declarative programming). The execution of relations can create new relations, which on its part creates facilities for *learning* in the applications. The program is executed by applying relations (*reasoning*) in parallel, instead of (typical to the era) sequential, manner. Similarly, to Lisp, Prolog has many different manifestations; one worth of mentioning here is *Concurrent Prolog* used in the Japanese Fifth Generation Computer System Project (third wave in this paper) as a basis for the computer architecture.

The intelligence related to the first wave is that knowledge to solve the problem is "hard-coded" to the program and known by the programmer only (closed intelligence). For the user of the application, it is reasonable difficult to see or understand the logics of the solution.

The Second Wave – Expert Systems, from 1970s to 1980s

Expert system (ES) is a computer application that has "built-in" intelligence – knowledge in the form of a *rule base*. By definition, expert system is a computer system emulating the *decision-making* ability of a human expert. Instead of programming language the end-user is defining his problems to the system by using the structures of the *problem specific* user interface. The problem solving is built-in to the implementation of the ES, which may be partially documented and understood by the end-user. This makes the built-in intelligence at least partially visible and understable to the user (open intelligence).

[4] John Mc Garthy: https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist).
[5] Darthmouth workshop: https://en.wikipedia.org/wiki/Dartmouth_workshop.
[6] Lisp: https://en.wikipedia.org/wiki/Lisp_(programming_language).
[7] Prolog: https://en.wikipedia.org/wiki/Prolog.

The rise of ESs started in 1970s. Two mainstream implementation technologies were rule-based and frame-based systems. In the former the knowledge was presented to the system in the form of rules; the latter one was based on the structured approach, in which the solution was found by matching the problem to the frames in the system's frame base.

The first expert system was introduced by *Edward Feigenbaum*[8] - *"*father of expert system*s"* - from Stanford University. The *Mycin*[9] system was developed for diagnostics of infectious diseases to recommend medical treatment. The system was written in Lisp and had knowledge base of 600 rules. Another early-stage ES developed at Stanford was Dendral[10] developed for hypothesis formation and discovery in science; it was first used to help organic chemists in identifying unknown organic molecules.

The currently best-known expert system is *IBM Watson*[11]. It is a question-answering system capable of answering questions posed in *natural language* and used in a variety of application areas. Its knowledge resources are available via APIs to third parties to develop their own applications. Watson is an example of the rebirth of the idea connected to the ES outside their main era. The development effort can be timed to start in 2000s and is continuing, of course applying a variety of technologies available today (compared to the situation in 1970s).

In Figure 2 we have included Hypertext (Hypermedia) and WWW as technologies closely related to AI. These have high importance to the basement of the current computing and information / knowledge management in the form of linked content structures; in a way this represents built-in structural intelligence in documents and document structures.

The Third Wave – AI in Architectures, from 1980s to 1990s

The traditional computers were designed to execute algorithmic programs in sequential manner. Some trials about implementing parallel processing and parallelization of software were made already in 1970s; supercomputers of time were based on multi-processor architecture, in which mainly arithmetic operations could be executed parallelly. This allowed complex scientific calculations but was not useful for tasks executed by AI systems.

Knowledge engineering and AI systems are based on *reasoning and inference processing*, instead of algorithmic data processing. Rule-based knowledge management – like in Prolog – has typically no execution order of the rules and because of that is possible to parallelize. From the late 1960s (V)LSI technology had fast progress. Already from 1970s technology to develop *Application Specific Integrated Circuits* (ASIC; even processors) had provided means for developing *specific computer architecture* to allow effective application-specific computing. This opened space for the new era in AI –

[8] Feigenbaum: https://en.wikipedia.org/wiki/Edward_Feigenbaum.
[9] Mycin: https://en.wikipedia.org/wiki/Mycin.
[10] Dendral: https://en.wikipedia.org/wiki/Dendral.
[11] Watson: https://en.wikipedia.org/wiki/Watson_(computer).

implementing inference and reasoning support *directly to the computer architecture* to get processing in such tasks more effective.

The most famous activity in this area was the Japanese nationwide project called "*New (Fifth) Generation Computer System*" (FGCS) [12] coordinated by the *Institute for New Generation Computer Technology (ICOT)*. This ten-year project was open for international collaboration and had focus on computer architectures, software, and (intelligent) applications like speech processing, natural language processing, language translation. The computer architecture was based on the *Concurrent Prolog* developed by Ehud Shapiro[13]. In architectural side both computers for personal use (PSI – Personal Sequential Inference Machine) and massive processing (PIM – Parallel Inference Machine) were developed. The latter one implemented massively parallel processing having thousands of processors.

The direct commercial success of the project remained finally insignificant – inference machines did not remain a part of mainstream computing. Despite, in two areas Japan made a giant step: in *software engineering* and in *intelligent applications* (image processing, speech recognition, natural language processing, online language translation) developed in the project. The deep *knowledge of computer architectures* Japanese already had before the project and strengthening of it was obvious, too. However, this was evidence on the opportunity to build *intelligence in architecture (architecture intelligence)* as a new era in the history of AI.

In the same time of Japanese effort MIT had activity to develop Lisp-based computer architecture. The commercial work was transferred to MIT Spin-off company Symbolics Inc., which produced Symbolics[14] computers a while in the 1980s. Neither these became a commercial success but had evidence about opportunity to implement intelligent architectures providing effective processing capacity for knowledge engineering tasks.

Why did this kind of specialized architectures finally not stay to the market? We refer to Table 1 providing a view to the progress of key factors. In 1980s slow processing of data in AI systems was a bottleneck. Fast progress in the enabling technologies has changed the situation: instead of maintaining and further developing the specialized "niche architectures" growth of computing power has finally made the use of software-based solutions as effective as specialized implementations. In addition, AI systems are quite often not independent but a part of complex interacting systems of systems, implemented by mainstream tools.

The Fourth Wave – Learning-based AI, from 2000s and continuing

Intelligent systems are based on system's ability to adapt (change the behavior, react in feedback) and to learn about the situation, in which it is used. Learning might be first taught and then self-learning during the use of the system. Traditional approach (in the

[12] FGCS: https://en.wikipedia.org/wiki/Fifth_generation_computer
[13] Ehud Shapiro: https://en.wikipedia.org/wiki/Ehud_Shapiro
[14] Symbolics: https://en.wikipedia.org/wiki/Symbolics.

three waves discussed) is to build the learning capability (intelligence) to the code, rule base or architecture.

The current wave of AI is based on the effective use of learning algorithms. In Figure 2 we have listed some concepts related to this: Neural Networks, Self-Organizing Maps, Deep Learning. *Neural network*[15] builds a model that resembles the structure processing of a human brain. It uses "what-if" based rules and it is taught (*supervised learning*) by examples. The network learns the non-linear dependencies between variables. An improved version of neural network is the Self-Organizing Map (SOM)[16] that is based on *unsupervised learning.* A multidimensional input (learning) data set is organized into layered relationships, which are represented as a low-dimensional map. This can be used as an abstraction of the real data space. *Deep learning*[17] theory. It is based on the *independent* learning of masses of data. The learning algorithms are based on the use of nonlinear statistics.

In this case, *intelligence is built in algorithms*, which itself are application independent and implement the learning capability of the system. Powerful learning algorithms and masses of data replace complex application specific intelligent algorithms, e.g. Google reports about its Translate application that the earlier translation algorithm of 500.000 LOC was replaced by a learning algorithm of 500 LOC (and data). Additional benefit is the learning algorithms' flexibility in learning new facts during the use.

3.3. Intelligence of the Artificial Intelligence – Analysis of the cycles

We have introduced four different approaches in Artificial Intelligence in the context of their birth:

- Intelligence in software code. Closed intelligence, in which the details were known only by the programmer.
- Intelligence in rules and the "knowledge engine" logics. The operational logics of the system is open to the user.
- Intelligence in the architecture. Intelligence transferred to the computer architecture. Direct support for the efficiency of the applications.
- Intelligence in the (learning) algorithms (and data). Human kind learning based systems. Algorithms are not known by the end-users. Key aspect is the *quality of the data*.

Figure 2 describes the cycles as a sequence. The idea is to have a look to the importance of each AI cycle in the time it was born and its role as an emergent technology – its innovation power. In reality, all of these technologies are still valid and in active use in a wide variety of applications (Figure 3).

[15] Neural networks: https://en.wikipedia.org/wiki/Artificial_neural_network.
[16] SOM: https://en.wikipedia.org/wiki/Self-organizing_map
[17] Deep learning: https://en.wikipedia.org/wiki/Deep_learning.

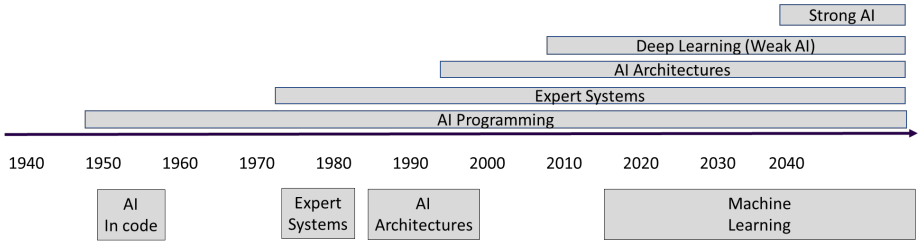


Figure 3. AI application space – the growth of AI applications

The current AI wave is called w*eak (narrow) AI*. The applications are task-oriented, in which the knowledge is not transferrable to a new application context. Learning algorithms themselves are general. Weak AI does not have its "own sense" related to the data it handles, nor its own will about how it should be handled. We are transferring towards *strong AI.* It can handle the facts and their relationships and has features of human beings, like *common sense*, but it does not have its own will either, rather a kind of understanding of its surroundings.

4. Learning and Intelligence – Computers like a Humans?

4.1. Creating new human-like texts

Currently we see a growing number of new AI applications based on use of natural, human language in various industries including banking, recruitment, health-care, agriculture, transit, etc. Advances of AI in creating human-like communication and replicating natural language patterns used by humans are based on large language corpora – a collection of human-produced texts in various encodings, first of all written text, but also, spoken, signed, etc.

Large corpora with billions of words are used to create text models, i.e. algorithms, which can parse input text and 'understand' it, i.e. answer some simple questions concerning the input. Abilities of corpora are often demonstrated with text generation – the text model continues given seed (start of a story) and produces believable output, i.e. new text which looks like created by human.

The main problem of language understanding is prediction of the next word (or character). Text/corpus model is a collection of conditional probabilities of the next word in text.

Suppose we already have a sequence of words:

$$w_1, w_2, \ldots, w_{i-2}, w_{i-1}$$

The next word w_i could not be arbitrary, it depends on preceding words. The next word could be guessed maximizing the relative probability (the Bayesian inference [13]):

$$\max(w_i \in V) P(w_i \mid w_1,...,w_{i-1})$$

Here $P(w_i \mid w_1,...,w_{i-1})$ is the conditional probability that after words $w_1,...,w_{i-1}$ follows the word w_i. In practice probabilities are estimated from real-word frequencies, i.e. the relative probability of word '*students*' after the previous words '*all our*' could be calculated from the frequencies of use of these words in a (large) corpus of text where these words were already used:

$$P(w_i \mid w_1,...,w_{i-1}) \approx \frac{Fr(w_1,...,w_{i-1},w_i)}{Fr(w_1,...,w_i)}$$

The probability of the whole phrase is the product of probabilities (the naïve Bayes rule), i.e. probability of the beginning phrase $P(w_1,...,w_{i-1})$ and the conditional probability that it is followed by word w_i :

$$P(w_1,...,w_i) = P(w_1,...,w_{i-1}) P(w_i \mid w_1,...,w_{i-1})$$

Human language is often considered as a process with limited memory (the Markov process) – assuming that the meaning of the next word depends only on a limited number of preceding words. This is generally not true, we expect often that the reader/listener already knows the meaning of many words which have been use. But applying the Markov process assumption '*probability of word depends only on few numbers of previous words*' simplifies programs and this is used in NLP everywhere. Thus, for prediction of the next word is used only a sequence of fixed length *k* (the naïve Bayes assumption, i.e. *k* is the length of the sliding cutout of the last *k* words) and the search goal is

$$\arg\max(w_i \in V) P(w_i \mid w_{i-k},...,w_{i-1}) \qquad (*)$$

To simplify notations, shift $i-k \to 1$, thus we are looking for

$$P(w_1,...,w_i) \approx P(w_1,...,w_{i-1}) P(w_i \mid w_1,...,w_{i-1})$$

Probability of the first phrase could be expressed the same way or those words are given as a seed.

In practice (to speed up calculations) the last formula is simplified even more. Using the naive Bayes conditional independence assumption that the probabilities $P(w_j \mid w_{j-1})$

are independent are in language models often used only binary probabilities (a very rough assumption), thus

$$P(w_1,...,w_n) = P(w_1)\prod_{i=2}^{n} P(w_{i-1} \mid w_i)$$

The presented above argmax formula (*) can be used to create new texts based on probabilities occurring in the text corpus – give some words $w_1,...,w_{i-1}$ as a seed and find a word w_i which maximizes probability $P(w_i \mid w_1,...,w_{i-1})$, then shift the 'action window' one step to right and repeat the process starting with sequence $w_2,...,w_{i-1}w_i$.

To get influence from farther earlier words are in NLP used contexts with lengths > 50 (words or characters – text processing has often done also on the level of characters), but this 'far influence' does not come from n-grams.

Longer contexts allow to predict next words, e.g. in the above text, if we already see words '*are using several programming*' then the next word could/should be '*languages*', if we see '*are popular social'*, then the next word should be '*networks*'. The longer the context the better it predicts, but the prediction always the next word would be some word which already followed in the corpus – there would never be pairs of words which had not occurred in this order already in corpus. A perfect parrot.

Modern digital methods can substantially increase influence the previous context in next word, but for this everything is converted to digital and instead of n-grams (exact fragments of input text) are used functions, which calculate inferences from 'farer' contexts.

For calculations words are first replaced by their numeric code in vocabulary and then are used two functions (inverse to each other):

- for a list of words (the bag-of-words) find word which probably can occur together with these words;
- for a word find its contexts, i.e. words which with high probability occur near this word (skip-gram).

In both cases for every occurrence of a word is calculated vector of probabilities of near-by words (the vord2vect) – words are represented by their contexts.

Thus, words are considered as elements of vectors of probabilities of their context words. Mapping from words to vectors is called 'Word Embedding'. The dimensions of these vectors can be rather large, e.g. the Stanford collection of pre-trained word embeddings [22] dimensions vary from 50...600.

The word vectors are not unique – they depend on the text corpus and even with the same text corpus different NLP packages (different methods for creating text model) produce (somewhat) different results.

Representing words as vectors with real-valued coordinates allows to calculate from the cosines product of their vectors distance (i.e. similarity) between words Vocabulary of the whole corpus becomes a 'cloud' of dots in multidimensional space. Methods to create word vectors and to use them for new text creation belong to Machine Learning (ML) research area.

4.2. What is ML ?

Learning is a process to improve, change learner's behavior in order that learner can better respond to its environment, better achieve its tasks. Computers are deterministic devices whose behavior does never change – is it does, then the computer is severely broken. When the same text corpus is re-used (with the same model structure) computer creates the same model and if it is used for text creation (with the same seed) appears the same text. From here it follows, that the acronym 'Machine Learning' is a misuse of the word 'learning'.

In order to understand each other, we should have some common understanding of terms what we use, but there is lot of dissension in use of terms 'information', 'knowledge', 'learning'. Would you say that Newton learned the Law of Gravity or Einstein learned the Theory of Relativity? They did not 'learn' those laws, they discovered them setting up totally new frames of thought, performing experiments, what nobody had thought of before. They first created new mental approach, new framework, then observed, collected data in this framework and then generalized their observations data as a new Laws of Nature.

When composing new text, for prediction of the next word is used only a sequence of fixed length *k* (the naïve Bayes assumption, i.e. *k* is the length of the sliding cutout of the last *k* words) of already created words and the search goal is

$$\arg\max(w_i \in V)P(w_i \mid w_{i-k},...,w_{i-1})$$

When humans speak/write, the next word also depends on all the already produced words, i.e. they use a procedure similar to rule (*) what computers use, but the process begins in their consciousness (denoted by $"$):

$$\arg\max(w_i \in V)P(w_i \mid ",w_1,...,w_{i-1}) \qquad (**)$$

The rule what computers use is only an approximation of the tail of the human's procedure. The premise $w_{i-k},...,w_{i-1}$ of the used in rule conditional probability is only a small tail of the premise $",w_1,...,w_{i-1}$ used by humans, thus the consequence w_i is less exact (its probability is smaller) and thus also the entropy (information content) of the whole produced phrase is smaller.

Word vectors (however long) can't express the meanings of words the way as we know them – we change them constantly. Depending on our mood, previous events, time of year/day etc. we can use the same words with quite opposite meanings: "*John, You did well!*" may mean ("*Good, we expected you to fail*") or ("*You failed, we expected you to win!*"). The current NLP research is trying to analyze sentiments (positive or negative) and some researchers even try to analyze more feelings [15; 2; 32]. But this (and many problems connected with memory) are difficult forms of verbal expression and difficult to re-produce – computers (yet) do not have feelings and do not know, what to remember - is the word *Hamburg* a name of a student, bird, virus or programming language and should it be stored in memory?

And here lays the main, most important difference between Machine Learning (ML) and Human learning (HL). Machine Learning in NLP is an approximation of the tail (visible) part of human communication.

4.3. Disentangling Hype from Reality

When speaking about neural algorithms, 'deep' learning, data science etc. it is often mentioned, that none of used here methods are mathematically proved. For many practical problems – how many 'hidden' layers, how many nodes in each layer, what kind of activation function to use etc. exist only some suggestions [7], the design decisions are stated, not explained [9]:

- Input layer will have 784 nodes
- Hidden layer 1: we have decided to reduce the number of nodes from 784 in the input layer to 128 nodes;
- Hidden layer 2: we have decided to go with 64 nodes;
- Output layer: we are reducing the 64 nodes to a total of 10 nodes

Many approaches which have become nearly standard do not have any reasonable explanation. For instance, use of the sigmoid function as activation functions:

$$\phi(z) = \frac{1}{1+e^{-z}}$$

This function is computationally expensive – uses power and division and can produce values close to zero, but its use is explained with "*The main reason why we use sigmoid function is because it exists between (0 to 1)*"[28] – any function can be normalized to have values between any two constants. ML is overabundance with 'ad hoc' methods and nearly mysterious ways in producing 'deep' inference models - you start a Tensorflow model and then follow on screen, how the main parameter – loss – first decreases, but then increases, i.e. the model is overfitting and should be re-organized::

```
58/987 [>.....................] - ETA: 2:01 - loss: 3.9472
59/987 [>.....................] - ETA: 2:00 - loss: 3.9366
60/987 [>.....................] - ETA: 2:00 - loss: 3.9261
...
808/987 [=======>......] - ETA: 23s - loss: 0.9714
```

809/987 [=======>.....] - ETA: 23s - loss: 0.9715
…
984/987 [==========>.] - ETA: 0s - loss: 0.973
985/987 [==========>.] - ETA: 0s - loss: 0.9738
986/987 [==========>.] - ETA: 0s - loss: 0.9738

Humans do not like such an unexplained 'black magic' and thus establishing trust in NLP, ML and AI technologies may be one of the most important skills of Data Scientists. This has created a new research direction: Explainable AI (XAI) – developing tools and frameworks to help you understand and interpret predictions made by your machine learning models [3; 31]. But XAI is trying to explain, not to prove anything.

To 'prove' ML or NLP is in principle not possible. To prove something (in mathematics) is possible only if we have a formal system in which can be formalized all our statements. Machine Learning is extracting information what an input random variable $X \in \mathsf{X}$ contains about an output random variable $Y \in \mathsf{Y}$, if we have their joint distribution $p(X,Y)$ and precise (i.e. mathematical) definition of input-output structures X,Y - we know, what are the properties and all possible values of probabilistic variables X,Y.

Among many (mathematical) results about neural nets the central are the Universal Approximation Theorems [38], which state, that neural net can approximate (i.e. calculate with whatever precision) any continuous (the graph is smooth continuous line) function $X \to Y$ (for simple explanation see e.g. [4; 19]).

But these theorems rely on precise mathematical properties of inputs-outputs. For NLP this means, that we should have a formal description of human language. Formal description for any human language is impossible in principle.

It is impossible to check, that there is a common for all speakers understanding of even our own mother language, that we all always understand all out utterances the same way – if there were, most of our social system, courts, laws, advocates etc. could be cancelled and replaced with computers (and humans were obsolete and officious), also the whole progress would vanish – progress happens, if somebody interprets established facts, common beliefs in a different way.

Human languages change constantly just the same ways as the whole mankind – the next generations constantly renew our language. For instance, to the Oxford English Dictionary were in Mach 2021 added more than 1400 new words [20], another source reveals, that in every 98 minutes is to the English language added a new word [5] and all other (living) natural language behave the same way.

All neural nets are inference algorithms, which can find consequences from given facts, but can't create new facts which do not follow from given data. Mathematicians have long ago devised a precise definition for 'provable'. All ML algorithms are inferences on a given set of facts (called database or text corpora). An inference algorithm $\models$ (e.g.

Tensorflow) on a database or text corpora KB is provable if for every sentence α inferred from KB, i.e.

$$KB \models \alpha$$

all interpretations in which sentences in the KB are true is α also true (see any textbook on formal logic, e.g. [6]).

Interpretators of text are humans. We all know from our everyday experience, that the database KB need not to be very big in order different meanings appear and the bigger the database KB, the more there will be different interpretations, i.e for many people KB does not contain only true statements and truth of inferred sentences α is (generally) very rare event – nearly the same rare as production of Shakespeare's opuses using the Engine.

Thus to 'prove' NLP text models or ML inference algorithms is impossible in principle. The NLP text models can make everything looking like truth. One of (currently) biggest models, the GPT2 accepted the following fable [21]:

"In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English."

The GPT-2 system continued the fable to look like a true story from some news agency:

"The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.…"

If a program can fluently explain *four-horned silver-white English-speaking unicorns* then it certainly can also prove that *Earth is flat*, *vaccines and 5G are evil* etc. - a perfect creator of 'fake news', but these news are 'fake' (or in modern terms. 'alternative truth') just because they are not provable. It can also explain contradicting statements, e.g. that those unicorns had five horns, could fly and where speaking Putonghua, the Mandarin dialect of Beijing, but some were also fluent in Russian.

The situation with 'proving' ML is rather similar with the claim "Computers today are Turing-complete, ie. can represent any computable algorithm" [17, p.4]. This seemingly very forceful statement is non-provable – the concept 'computable algorithm' (or in more mathematical terms 'effectively calculable function') cannot be formalized, thus cannot be used in mathematical formalized proof – any formalization will be subjective, will cover only these forms of algorithm that the author considered meaningful but nobody can know, whether there exist some other forms of algorithms.

The natural language may just be an example of algorithm, which can't be formalized. The NLP assumes, that in a (long enough) sequence of words the next word is predictable. Thus if the sequence $w_1,\ldots,w_{i-1},w_i$ can occur in a natural language (native

language users accept it), then there exists a function/algorithm for producing the next word:

$$f(w_1,...,w_{i-1}) = w_i$$

However, the numerous efforts and continuing research have not yet managed to produce such a function/algorithm – they may produce explanation to *silver-white four-horned English-speaking unicorns* or the Chomsky-s famous phrase *Colorless green ideas sleep furiously* but cannot prove truth of these statements.

Human messages are created by or consciousness, our feelings. Trying to make the source of meaningful messages these messages themselves (text corpuses, however how big) is exactly what did Baron Munchausen - pulled himself out from a swamp by his own hair. Shouting: "NLP Cracked Transfer Learning" is adding the horse to the load (Baron did this – he was riding).

Neural net is a multi-variable function from input space to output space. All proven statements about neural nets assume precise formal description of input-output spaces – otherwise we could not prove anything. The Universal Approximation Theorems (for popular introduction see (e.g. [19]) establishes that neural nets can approximate whatever continuous functions between Euclidean spaces; there are also variations for non-Euclidean spaces, algorithmically generated function spaces etc. In practice are some aspects of these theorems often overlooked.

First, theorems do not say, how to organize approximating neural net – how many layers, how many units in every layer, what kind of activation function to use – the best valued have to be find with practical experiments.

Second, many problems are not continuous functions, e.g. all classification problems, image recognition problems etc. For non-continuous problems a neural net may not converge at all and researcher has to start experimenting.

There are no mathematical results of type:

ML: *Input_text* $\rightarrow$ *Output_text*

The *Input_text*, *Output_text* are not mathematical structures, every natural language model (e.g. neural net) creates (makes a mathematical approximation) his own way. Ambiguity and misunderstanding has created lot of frustration among data scientists [1; 33; 18]. This has both deeper causes and also deeper consequences.

According to research [30]: "Sixty-six percent of data scientists describe themselves as self-taught.", thus most probably they have not learned the (elementary) facts about proofs discussed in the previous paragraph. As a consequence, they are uncertain meaning and value of their activities (see e.g. [26]) and are every week "spending 1-2 hours a week looking for a new job" [30].

5. Conclusion

AI as well as any kind of computation has its own merits. HI is oriented on the needs and challenges a human face. It also supports human societies. There are many tasks that cannot be handled by humans and living beings. [24] compares the abilities to fly. As already mentioned, think about an albatross who can stay in the air for days and covers thousands of kilometers. If we need to carry hundreds of tons from Europe to Japan then artificial devices such as planes can manage this task much better. Routine, heavy, or complex tasks can better handle by artificial devices.

Our artificial systems do not really produce anything new in reality. They bring, however, a great purely practical improvement in life. They increase speed, effectivity, and performance for everybody who has access to them. They enable a comfortable life for many people. Whether we call them 'intelligent' is a matter of definition for intelligence.

Looking to the future, we need an approach to limit the abuse and misdevelopment of technical systems. Already the scare with the atomic bomb has brought us the insight that any kind of weapon - be it also a chemical or biological one - must be wisely limited with a worldwide moratorium, lest humanity be brought to the brink of existence by power-obsessed elites. This is just as true for AI systems today. We also need containment against such abuses of AI.

The history of AI proves that human-like computer applications attract humans. The superiority of AI is based on its ability to handle big amounts of data mined from a variety of distributed sources. Finally, this superiority is based on "brute force" controlled by algorithms. In a way, even HI is based on algorithms. Researchers have tried to model adopt these algorithms to be applied in AI. In the current wave of AI learning is the key element. This is good start towards HI, but still a lot is missing: human sense, human kind of criticism, emotions and human ethics are examples of the missing elements. Current AI (weak, narrow) is still context dependent and not transferrable to new application areas. Next step leads towards human-like strong (general) AI. An interesting topic to think is computer-brain integration. Gartner[18] defines it "a type of user interface, whereby the user voluntarily generates distinct brain patterns that are interpreted by the computer as commands to control an application or device". In the Gartner's hypeslope of Emerging Technologies 2020[19] it locates in first segment of five (Innovation trigger) but expects its appearance during the next decade. The same segment in the curve includes a lot of promises in the AI area: self-supervised learning, adaptive machine learning, composite AI (variety of AI techniques combined), generative AI (ability to create new content). To conclude – still a lot new to wait, but human intelligence is unreachable.

[18] https://www.gartner.com/en/information-technology/glossary/computer-brain-interface.
[19] https://www.gartner.com.au/en/articles/5-trends-drive-the-gartner-hype-cycle-for-emerging-technologies-2020.

References

[1] Brooks-Bartlett J. (2021). Here's why so many data scientists are leaving their jobs. Retrieved May 3rd 2021 from https://towardsdatascience.com/why-so-many-data-scientists-are-leaving-their-jobs-a1f0329d7ea4.

[2] Zhang D., Wu L., Sun C., Li S., Zhu Q. and Zhou G (2019). Modeling both Context- and Speaker-Sensitive Dependence for Emotion Detection in Multi-speaker Conversations. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), pp. 5415-5422.

[3] Explained AI (2021). Understand AI output and build trust. Retrieved May 3rd 2021 from https://cloud.google.com/explainable-ai.

[4] Fortuner B. (2021). Can neural networks solve any problem? Retrieved May 3rd 2021 from https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6.

[5] Global Language Monitor (2021). Number of Words in the English Language. Retrieved May 3rd 2021 from https://languagemonitor.com/number-of-words-in-english/no-of-words/.

[6] Hauskrecht M. (2021). Propositional logic. Inferences. Retrieved May 3rd 2021 from https://people.cs.pitt.edu/~milos/courses/cs1571-Fall2013/Lectures/Class12.pdf.

[7] Heaton Research (2017). The Number of Hidden Layers. Retrieved May 3rd 2021 from https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[8] Hesse, Wolfgang (2020). Das Zerstörungspotential von Big Data und Künstlicher Intelligenz für die Demokratie (The destructive potential of Big Data and Artificial Intelligence for democracy). Informatik Spektrum 43(3), 339-346.

[9] IBM Developer (2021). Neural Networks from Scratch. Rerieved May 3rd 3021 from https://developer.ibm.com/technologies/artificial-intelligence/articles/neural-networks-from-scratch/.

[10] Jaakkola, H., Henno, J., Thalheim, B. and Mäkelä, J. (2017). The educators' telescope to the future of technology. In P. Biljanović (Ed.), MIPRO 2017 - Proceedings of the 40th Jubilee International Convention. May 22-26, 2017, Opatija, Croatia. (pp. 766-771). Opatija, Croatia: Mipro and IEEE.

[11] Jaakkola, H., Brumen, B., Henno, J., and Mäkelä, J. (2013). Are We Trendy? In P. Biljanović (Ed.), 36th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2013 - Proceedings (Vol. ISBN 978-953-233-074-8; ISBN 978-953-233-075-5, pp. 649-657). Opatija: Mipro and IEEE.

[12] Jaakkola, H., Henno, J., & Mäkelä, J. (2017). Technology and the Reincarnation Cycles of Software. In Z. Budimac (Ed.), SQAMIA 2017 - Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications. Belgrade, Serbia, September 11-13, 2017. (Vol. Vol-1938, pp. 5:1-10). Belgrade, Serbia: CEUR Workshop Proceedings.

[13] Jurafsky D., Martin J.H. (2020). Speech and Language Processing. Retrieved May 3rd 2021 from https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf.

[14] Kandel, Eric R. (2007). In search of memory: The emergence of a new science of mind. WW Norton & Company.

[15] Kerkeni L., Serrestou Y., Mbarki M., Raoof K., Mahjoub M.A. and Cleder C. (2019). Automatic Speech Emotion Recognition Using Machine Learning. DOI: 10.5772/intechopen.84856

[16] McCarthy J., Minsky M.L., Rochester N., Shannon C.E. (1955). A Proposal for the Darthmouth Summer Research Project on Artificial Intelligence. Retrieved April 25th, 2021 from http://raysolomonoff.com/dartmouth/boxa/dart564props.pdf.

[17] Mikolov T. (2021) . Statistical Language Model Based on Neural Networks. Retrieved May 3rd 2021 from https://www.fit.vut.cz/study/phd-thesis/283/.en

[18] Moutard C. (2021). Data Science and AI: From Promises to Frustrations. Retrieved May 3rd 2021 from https://medium.com/data-by-saegus/data-science-and-ai-from-promises-to-frustration-cd0cffc60933

[19] Nielsen M. (2021). Neural Networks and Deep Learning, A visual proof that neural nets can compute any function. Retrieved May 3rd 2021 from http://neuralnetworksanddeeplearning.com/chap4.html

[20] OED (2021). Updates to OED. Retrieved May 3rd 2021 from https://public.oed.com/updates/.

[21] OpenAI (2021). Better Language Models and Their Implications. Retrieved May 3rd 2021 from 18 https://openai.com/blog/better-language-models/.

[22] Pennington J., Socher R., D. Manning C.D., GloVe (2021). Global Vectors for Word Representation. Retrieved May 3rd 2021 from https://nlp.stanford.edu/projects/glove/

[23] Postman, N. (1985). Amusing Ourselves to Death: *Public Discourse in the Age of Show Business. Penguin.*

[24] Radermacher, F. J. (2017). Die Zukunft der digitalen Maschine. Was kommt auf uns zu? (Die future of the digital machine. What lies ahead?). Denkströme. Journal der Sächsischen Akademie der Wissenschaften, 2017, No. 17. Retrieved March 20th 2021 from http://www.denkstroeme.de/heft-17/s_134-159_radermacher

[25] Rogers, H. Jr. (1987). Theory of Recursive Functions and Effective Computability. Cambridge, MA: MIT Press.

[26] Saroufim M. (2021), Machine Learning: The Great Stagnation. Retrieved May 3rd 2021 from https://towardsdatascience.com/machine-learning-the-great-stagnation-3a0f044e17e0

[27] Schirrmacher, F. (2015). EGO – The Game of Life. *Polity*, Munich.

[28] Sharma S. (2021). Activation Functions in Neural Networks. Retrieved May 3rd 2021 from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[29] Siegelmann, H. T. and Sontag, E. D. (1995). On the Computational Power of Neural Nets. Journal of Computer and System Sciences *50 (1995), 132–150.*

[30] Smith Hanley Associates (2020). Why Are Data Scientists Frustrated? Retrieved May 3rd 2021 from https://www.smithhanley.com/2020/03/31/data-scientists-frustrated/.

[31] Turek M. (2021). Explainable Artificial Intelligence (XAI). DARPA. Retrieved May 3rd 2021 from https://www.darpa.mil/program/explainable-artificial-intelligence.

[32] Wadhwa M., Gupta A. and Pandey P.K. (2020). Speech Emotion Recognition (SER) through Machine Learning. Retrieved May 3rd 2021 from https://www.analyticsinsight.net/speech-emotion-recognition-ser-through-machine-learning/.

[33] Waters R. (2021). How machine learning creates new professions — and problems. Financial Times. Retrieved May 3rd 2021 from https://www.ft.com/content/49e81ebe-cbc3-11e7-8536-d321d0d897a3.

[34] Wiedermann, Jan (2012). A Computability Argument Against Superintelligence. Cognitive Computation 4(3), 236-245.

[35] Wikipedia (2021). Artificial Intelligence. Retrieved April 25th 2021 from https://en.wikipedia.org/wiki/Artificial_intelligence.

[36] Wikipedia (2021). DIKW Pyramid. Retrieved April 24th, 2021 from https://en.wikipedia.org/wiki/DIKW_pyramid.

[37] Wikipedia 2021b, History of artificial intelligence, Retrieved April 25th, 2019 from https://en.wikipedia.org/wiki/History_of_artificial_intelligence.

[38] Wikipedia (2021). Universal Approximation Theorem. Retrieved May 3rd 2021 from https://en.wikipedia.org/wiki/Universal_approximation_theorem

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

Models and Modelling in Computer Science

Bernhard Thalheim[0000−0002−7909−7786]

Christian-Albrechts University at Kiel, Dept. of Computer Science, D-24098 Kiel, Germany bernhard.thalheim@email.uni-kiel.de
http://www.is.informatik.uni-kiel.de/∼thalheim

Dedicated to Egon Börger

Abstract. Models are a universal instrument of mankind. They surround us our whole lifespan and support all activities even in case we are not aware of the omnipresence. They are so omnipresent that we don't realise their importance. Computer Science is also heavily using models as companion in most activities. Meanwhile, models became one of the main instruments. The nature and anatomy of models is not yet properly understood.
Computer Science research has not yet been properly investigating its principles, postulates, and paradigms. The well-accepted three dimensions are states, transformation, and collaboration. An element of the fourth dimension is abstraction. The fourth dimension is modelling. We review here the fourth dimension.

Keywords: models, art of modelling, modelling theory, computer science pitfalls, model theory

1 Models are the Very First Human Instrument

Wavering forms, you come again;
once long ago passed before my clouded sight.
Should I now attempt to hold you fast? ...
You bear the images of happy days,
and friendly shadows rise to mind,
With them, as almost muted tale,
come youthful love and friendship.
Goethe, *Faust I*

Models, Models, Models – We are Surrounded with Models

Humans use many instruments and especially intellectual instruments. The very first intellectual instrument we use is a model. Babys quickly develop their own models of the 'mother' and 'father'. They cannot yet use a natural language but they know already models of their surroundings [22]. Later children realise that their models of 'father' or 'mother' are different from those used by other

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

children although they use the same terms in their native language for referencing to mothers and fathers.

Later, we develop mental concepts and we learn to strengthen them to codified concepts. Concepts can be considered as the main and basic elements of mental models. Codified concepts are essential elements of conceptual models. Education is model-backed. Almost all scientific and engineering disciplines widely use modelling. Daily life and social interaction is also model-backed.

The wide use of models is caused by: models are often far simpler; models reduce complexity of systems; models support reasoning; models support interaction, communication, and collaboration; models are more focused, abstract, and truncated; models support perception; models must not be correct - they should however be coherent; models may be preliminary; models ease understanding; models can be understood on the fly; etc.

In general, models are and must be useful. Mental models are used by everybody in an explicit but often implicit form. They have an added value in life, society, science, and engineering, e.g. models in Ancient Egypt. Models have not to be named 'model'[1]. Their importance is far higher than often assumed.

Our discipline is not an exception for the misunderstanding of modelling. Moreover, modelling is a central activity due to the complexity of our systems and especially our software. There is a simple reason for the importance of models: systems are far more complex than a human can entirely understand. Models, however, allow to concentrate, scope and focus on certain aspects while neglecting others or coping with other through other models.

Models, Everywhere, Anytime, for and by Everybody

Computer Science[2,3] (CS) and especially Applied Computer Science (ACS)[4] are unthinkable without modelling. Modelling has been already used when the

[1] The oldest mention we acknowledge is the usage in Ancient Egypt with the use of models as moulds, models as representations, and models of the right order ('maat'). The first explicit notion of model is 'metron' in Ancient Greece and 'modulus' in Roman time, i.e. at least 40BC. The wide use of this word came with engineering in the 16th century and with sciences in the 19th century.

[2] A description of Computer Science has been given in [12]:
"Computer Science and engineering is the systematic study of algorithmic processes their theory, analysis, design, efficiency, implementation and application that describe and transform information. The fundamental question underlying all of computing is, *What can be (efficiently) automated.*".

[3] Computer Science can be divided into kernel CS and applied CS. The first subdiscipline spans theoretical, practical and technical CS.

[4] We do not know a commonly accepted description of this subdiscipline.
Essentially, applied CS has two branches: specific application-oriented CS and engineering of solutions in applications. The first branch has led to a good number of so-called 'hyphen' CS such as business informatics, biomedical informatics, and geoinformatics. The second branch is engineering of work systems [2], i.e. systems "in which humans or machines perform processes and activities using resources to produce specific products or services for customers". It spans topics such as information

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

history of CS started. We use very different names for artifacts that are models, e.g. specification, declaration, or description. We also know more that 60 different notions of conceptual model. As far we know, the variety of notions of model goes far beyond 450.

Analysing the state of art in CS&ACS we conclude that each branch widely uses models. Depending on the construction, description, prescription, prognosis, investigation, and application scenarios where models are used, models function as blueprint for system and software construction, as development companion, as starting or inspiration point, as means for systems analysis and prognosis, as means for learning and reflection, as reasoning instrument, as reflection and representation of some observed reality or system, as thought guide, as means for explanation and elaboration, as helper for integration and modernisation, etc. There is no branch in CS&ACS that does not use models.

Models are specific in dependence on the role they play as instruments in scenarios. If the model is dependable and especially of sufficient quality then it properly functions in those scenarios. Models are also used for sense-making in a more foundational setting, for delivering essential information about the system, and for representation of aspects that are currently of interest. Models are a good instrument for communication or more general interaction and collaboration. And they are used by everybody at the perception or conceptual or intelligible levels.

Inheritance of heritage knowledge is considered to be a really bad habit in CS. "Never cite a paper that is older than 5 years" and "never read and learn from old wise papers" are common approaches in modern research. Such revolutionary approach has never led to a real good outcome in history and disciplines. The body of knowledge of a science consists however of compiled achievements of research. Models and modelling is one of those. Modelling has its huge body of knowledge that waits its digestion, systematisation, and compilation.

2 The Fourth Dimension of Computer Science: Modelling

What you inherit from your father,
Earn it anew before you call it yours.
What does not serve you is a heavy burden,
What issues from the moment is alone of use.
Goethe, *Faust I*

Towards an Understanding of our Disciplines

The great success of redevelopment of Mathematics by the Nicolas Bourbaki group [6] led to a structure-oriented redesign of Mathematics. This group found

systems, information science, information theory, information engineering, information technology, information processing, or other application fields, e.g. studying the representation, processing, and communication of information in natural and engineered systems. It also has computational, cognitive and social aspects.

that Mathematics has three guiding dimensions: algebra, order, and topology. Following [47], we can distinguish three dimensions of computer science: states, transformation, and collaboration[5]. Computation is considered so far as state transformation. Systems are never completely monolithic. They consist of interacting components. The disciplinary orientation understands interaction as some kind of collaboration. So far we straiten human behaviour into this schema and squeeze together human behaviour and machine operating[6]. The fourth dimension is often underestimated but not less important: models. Figure 1 depicts the interrelation of the four dimensions and adds to the CS&ACS intext the outer context.

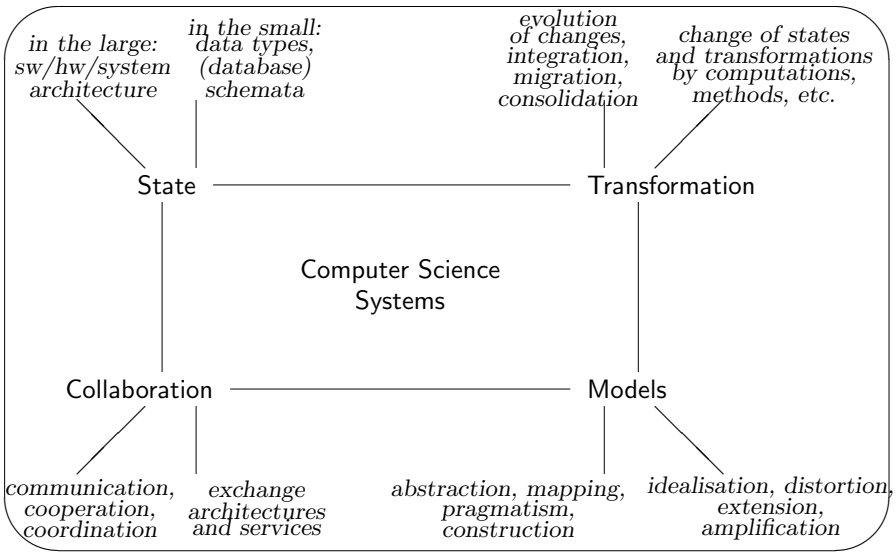


Fig. 1. The four dimensions of computer science: (1) the state dimension; (2) the transformation dimension; (3) the collaboration dimension; (4) the model dimension

The three dimensions are interwoven. For instance, central concerns in database management are states and state transformation. The fourth dimension is also interwoven with the other three. Models scope and focus to the essentials and allow us to restrict attention and to construct systems in a thoughtful way.

[5] A dimension that has not found its proper entry into our discipline is approximation. Approximation is the fifth dimension orthogonal to states, transformation, and collaboration.

[6] We expect that modern applications such as internet technology have to use humanisation as the sixth dimension in order to cope with modern interdisciplinary tasks.

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

Models allow us to avoid complexity, to concentrate first on the most important issues (pragmatism), to 'dream' on system extensions not yet observable or developed (amplification), and to abstract from the current state by an ideal state of affairs (idealisation). Humans are restricted in their mental abilities and cannot properly reason on complex systems. Instead, we order, categorise, generalise, abstract, and stereotype in a methodological mould in such a way that our models, macro-models, meso-models, and micro-models can be mapped to operating mechanisms.

CS&ACS have not yet developed a 'Bourbaki' collection of paradigms and principles. At present, Computer Science can be defined as the systematic study of computer and computation systems, "including their design (architecture) and their uses for computations, data processing, and systems control." (slightly changed from [32]). Following [12], we can assume that kernel CS is based on three main competencies: theory, abstraction, and construction (or composition; originally 'design'). We may consider modelling as a fourth competency. Modelling did not find a proper foundation in CS. It is however considered to be one – if not the main – of the core competencies in ACS.

What Means Modelling?

Models are used as instruments in our activities. Modelling thus includes mastering and manufacturing the instruments. Models should be useful. Modelling thus also includes deployment in preparation for work, utilisation with skilled acts of using, and well-organised application stories for models. We should not target for ideal or perfect instruments. The 3U macro-model considers usefulness, usability, and usage as a holistic quality for models. Modelling as mastering meets thus two challenges: appropriate[7] models with sufficient quality in use. Models are oriented on its use in application 'games'[46] by users within their context and usage culture.

CS Theoreticians, Researchers, and Developers are Modelling

A common misbelief is that theoreticians, researchers, and developers do not need models. They can develop their ideas and thoughts without modelling. Code is then something like a plan and presentation of these ideas. What are then ideas and presentations? These are two mental states, the first one as perception or idea (in German 'Auffassung') and the second one as imagination (in German 'Vorstellung') [7, 28, 43]. These are then nothing else than perception models and their presentations.

A second claim is often made: the code is the model. We ask ourselves 'of what'? Of the operating of a machine from one side and of our understanding how the machine would work from the other side. So again, it is a model. Whether it is unconscious, preconscious, or subconscious does not matter. It is already a model mediating between expectations and operating.

[7] See below: adequate and justified, sufficient internal and external quality.

A third claim sometimes asserted strongly is the belief that Theoretical Computer Science may sidestep models. What about their models of computation that became THE unshakable paradigm of computation? Nothing else than these models can be accepted. We may ask ourselves whether there are other useful models of operating.

CS Modelling is so far an Art or Handicraft

Modelling is so far treated as craft or handicraft, i.e. the skilled practice of a practical occupation. Modellers perform a particular kind of skilled work. Modelling is a skill acquired through experience in developing and utilising a model. At present it becomes an art. Art aims at creation of significant (here: appropriate) things (here: models)[8]. Artistry is a conscious use of skill and creative imagination that one can learn by study and practice and observation. usually acquired proficiency,capability, competence, handiness, finesse, and expertise in modelling

We envision that modelling is becoming a sub-discipline in CS due to its importance as the fourth dimension. It will be a branch of knowledge based on a controllable and manageable system of rules of conduct or methods of CS practice. So far, a number of methodologies has been developed. Some of them became a mould[9] for modelling. Modelling is not yet a science [10] or culture[11].

Models in Applied Computer Science

Models are also used in manufacturing for instance as as a template, pattern, reference, presentation, prototype, origin for production, master copy, and sample. We should note that models and modelling is different for engineering and

[8] Donald Knuth followed this meaning by calling his four volumes: 'The art of programming' [24].

[9] A mould is a distinctive form in which a model is made, constructed, shaped, and designed for a specific function a model has in a scenario.
It is similar to mechanical engineering where a mould is a container into which liquid is poured to create a given shape when it hardens. In Mathematics, it is the general and well-defined, experienced framework how a problem is going to be solved and faithfully mapped back to the problem area.

[10] A model and modelling science consists of a system of knowledge that is concerned with models, modelling and their phenomena. It entails unbiased observations and systematic experimentation. It involves a pursuit of model and modelling knowledge covering general truths or the operations of fundamental laws.

[11] Culture combines approaches, attitudes, behaviour, conventional conducts, codes, traditions, beliefs, values, customs, thought styles, habits, the system comprising of the accepted norms and values, goals, practices, and manners that are favored by the community of practice. It is a characteristic of this community and includes all the knowledge and values shared by the community of practice.
Culture of modelling is a highly developed state of perfection that has a flawless or impeccable quality.

also for ACS. It is far less well understood in CS. This observation is also true for software engineering[14] that makes heavy use of models but neither coherent, systematically, nor considering the engineering specifics. Software architecture [13] is not an exception.

Models can but don't have to be explicitly designed for usage. Engineering as the approach to create the artificial [36] is based on models as a starting point for construction, as a documentation of the construction, as a means for developing variations, as a thought and imagination instrument, and as an artifact within the creation process. There are, however, objects that became models at a far later stage of their existence[12].

3 Myths about Modelling

I've studied now Philosophy and
Jurisprudence, Medicine – and even, alas!
Theology – from end to end with labor keen;
and here, poor fool with all my lore I stand,
no wiser than before.
Goethe, *Faust I*

Is modelling important? Should modelling become a professional skill? Or a profession? It seems that this is not (yet) and must not be the case. It is a side technique used by everybody. Modelling does not have a prominent role in kernel CS education. A separate module named 'Modelling' is a curiosity in curricula. Let us briefly consider some arguments against the existence of such modules in curricula. From the other side, all branches of CS including ACS make heavy use of models. So far we know, the earliest publication which uses the notion of model is 2.500 years old[13]. May be, the notion is even older since it seems that Ancient Egyptians already deliberately used models. CS and ACS have developed many novel approaches to modelling and are far away from considering their way of modelling as the sole right of representation[14]. We claim that modelling is one of the main activities in our field and one of our main competencies. The analysis [41] underpins this exceptional role of models in CS compared to other disciplines.

[12] Objects can be developed for usage. At a later stage, they were exhibited and become models used for explanation, e.g. about culture. See for instance, Ancient Egyptian objects in modern museums. We also may observe the opposite for the model-being of object, e.g. see [8].

[13] The earliest source of systematic model consideration we know is Heraclitus (see [26] for a systematic and commented re-representation of Heraclitus fragments) with his concept of $\lambda\acute{o}\gamma o\varsigma$ (logos).

[14] Mathematician often claim that they are the only ones who know what is a model and what is modelling. We notice, however, that modelling is typically performed outside Mathematics.

(1) There is no commonly acceptable notion of model

CS uses the term 'model' in a manifold of cases. Models have very different functions. It seems not to be fruitful to use a general singleton notion of model. Instead we may use a parameterised notion with parameters that can be refined and adapted according the specific function that a model plays in an application scenario, in a given context, for some community of practice, for a collection of origins which must be represented by the model[15].

(2) Modelling languages must be as expressive as only possible

After having their success stories, modelling languages become dinosaurs. Anything what is thinkable in the application domain of such languages is integrated. For instance, the entity-relationship modelling language started with 4 constructs (entity type, relationship type, attribute, cardinality constraints) and was quickly extended by more than 150 additional constructs [38]. Boiling down these extensions and reducing to essential elements we use nowadays around a dozen of constructs.

(3) Modelling languages should be primarily syntactic – Syntax first, semantics later, avoid pragmatics, disregard pragmatism

CS follows the mathematical or logical treatment of semiotics. A concept is introduced in a syntactic form and gets it meaning in a second semantification step. Pragmatics is often neglected as well as pragmatism. The two extremes are then full fledged syntactical constructions or rather simplistic and simple syntactical constructions that are then combined with heavy semantics. Natural languages use words and construction in a holistic semiotics form. For instance, the relational data structure language uses a very small syntactic language. Since the syntax for structures is insufficiently expressive for applications, this syntactical part is extended by more than threescore classes of integrity constraints.

(4) Each hype must have its modelling language

CS is an area of extensive and eager hype development. Since hypes come and vanish and since hypes are typically born due to some challenges to existing

[15] The notion
"A model is a simplified reproduction of a planned or real existing system with its processes on the basis of a notational and concrete concept space. According to the represented purpose-governed relevant properties, it deviates from its origin only due to the tolerance frame for the purpose." [44]
is a typical example of this parametrisation. The origin is the system and the inherited concept space. Analogy is essentially a mapping. Focus is simplification. Purpose is reproduction or documentation. The justification is inherited from the system and its processes. Sufficiency is based on tight mapping with some tolerated deviation.
In a similar form we use parameters for the definition in [1]:
"A model is a mathematical description of a business problem."

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

approaches, they use new vocabulary and thus new models. For instance, big data applications are considered to be schema-less and thus model-free. In reality they use models. Realising this we use a 'novel' language for their description. Essentially, big data modelling is based on the sixth normal form from relational database approaches.

(5) Implementable modelling languages should be minimalistic

The relational database modelling language is based on two syntactical constructors (relational type and attribute). Anything what is to be given in more detail can be semantically added. This treatment requires skills of a master or journeyman. It often results in introduction of artificial elements. For instance, RDF use only labelled binary relationships among labelled nodes. From predicate logic we know that any predicate can be expressed by associated binary predicates if we introduce a concept of identifier which is essentially a surrogate value and makes modelling a nightmare whenever the application is a bit more complex than the toy examples used for explaining RDF.

(6) A model reflects main aspects of applications and allows to derive all others

A model should holistically cover all aspects of a given application. This restricts development of a model or results in infeasibility or impossibility of model development. Database modelling is a typical example. A database schema is introduced in the global-as-design approach. All specific viewpoints of business users can be represented by views that are from the global schema. This philosophy results in assumptions such as unique-name, unique-granularity, and unique identifiability. New database applications needs then work-arounds or other languages.

(7) Models can be discarded after being used

A common programming approach is to use some kind of model for inspiration or for documentation of the negotiated agreement in a community of practice. These models are later thrown away since they seem not to be of any use anymore. For instance, UML models are used as inspiration models for programmers. They can be discarded after they have been used for coding. At the best, they are stored in the documentation. Revising, modernising, or migrating the code does not havde an impact on the model.

(8) At its best, models are used for programming and coding

Models often carry far more knowledge about an application domain than the program that has been derived from the model. For instance, language terms used in models have their own linguistic meaning and carry important semantics. The main function of models is the support for program construction. Programs might also be models to certain extend. But the model is used in this case mainly

as the initial point for programming.
From the other side, modelling languages are often so well-developed that models written in these languages can be fully (or at least in form of templates) translated or compiled to code. The modelling-to-program initiative[16] matures the model-driven approaches to executable models, i.e. modelling is then professional programming.

(9) One model is sufficient for holistic representation

Systems have typically many sub-systems, are embedded, and are used with a good variety of features. Modelling often assumes that a singleton model allows to represent all these facets and aspects. The database schema approach mentioned above is a typical example of this belief. A weaker assumption is the existence of a model ensemble that uses separation of concern for development of models for each aspect. We better use model suites that consist of tightly and coherently associated models [10]. Models in a model suite reflect specific aspects and facets but are associated to each other in a way that allows to control and to manage coherence.

(10) Modelling can be practised as handicraft or an art

Modelling uses cookbook approaches which can be used in a handicraft at the level of a journeyman who starts as apprentice with learning how to model on the basis of toy examples. Modelling is thus – at its best – an art and does not need a theory. We envision, however, that modelling must be performed at the level of a professional master craftsman. It will then develop its own modelling culture.

(11) Modelling languages should have a flexible semantics and fixed syntax

The UML community uses an approach with refinable semantics. Each member of the community of practice and tools may use their own interpretation. The result is a 'lost-in-translation' situation resembling a nightmare. The syntax allows then a variety of translations depending on a hidden intention of the language user. Instead, languages must support precise descriptions both in syntax and semantics. Moreover, the UML approach uses model suites with a number of somehow associated models each of it will have their variety of interpretations. Harmonisation becomes a nightmare. Integration, modernisation, and migration of systems cannot be properly supported. Instead, a model suite can be based on institutions for signatures. Each model in the model suite is expressed in a properly defined language. Models are associated by a fixed association schema. We can then uniquely translate one meaning to another one.

[16] See http://bernhard-thalheim.de/ModellingToProgram/ .

(12) Programs and algorithms first, structures later

Programmers are used to concentrate on an algorithmic problem solution. The lessons of first computer programming with bit and byte tricky realisation seem to be forgotten. At least with data mining we discovered that efficient algorithms have to incorporate supporting data structures. We should break with the bad habit to consider design of proper variable collection as nonessential for programming. CS solution development has to be based on a sophisticated integration of structures, functionality, interaction, and distribution. So, models have to be based on co-design approaches that guarantee this integration.

(13) There are no hidden deep models behind

Computer Science models often keep quiet about postulates, paradigms, and more general its basis or grounding. For instance, the basis of Turing machines includes a number of implicit principles such as compositionality, functional or relational state transformations, step-wise computation, and context-freeness. One of the guiding implicit postulates is the Von-Neumann-machine and its sequentiality and separation into computation, control, and data. An implicit and concealed deep model circumvents deliberate model utilisation. In order to overcome this pitfall, the abstract state machine approach explicitly uses the three guiding postulates for sequential computation (postulates of sequential time, of abstract state, of bounded exploration of the state space)[3].

(14) Models represent machines as state-transformers

Digital computers are based on a notion of state. Computation is controlled state transformation. Many technical devices like analogous computers are not built on this paradigm. Neural networks might be coarsened understood as state transformers. The first Leibniz multiplication machine integrated analog and digital computation. CS concentrated on state transformation. Models seem to follow the same paradigm although this is not necessary.

(15) All models are conceptual models

There is no common agreement which artifact should (not) be considered to be a (conceptual) model although the term 'conceptual model' is used for more than for five decades in computer science and for more than one century in science and engineering. It is often claimed that any model is a conceptual one. Conceptual models are, however, models with specific properties. Better we claim:
"A conceptual model is a function-oriented and consolidated model of origins that is enhanced by concept(ion) spaces of their origins, is formulated in a language that allows well-structured formulations, is based on mental/perception/domain-situation models with their embedded concept(ion)s and notions, and is oriented on a matrix that is commonly accepted." [20]

(16) Models are monolithic at the same level of abstraction

It seems that programming has to be based on the same level of abstraction, e.g. programs based on formal grammars with word generation at the same level of abstraction. Second-order grammars do not change this picture. Models representing UML model collections, OSI communication layering, and information systems are typically model suites (see myth (9)). Model suites use several levels of abstraction. The abstraction layer meta-model [38] for development of an information system is an example of integrated and controlled use of models in a model suite at several abstraction layers.

(17) CS is entirely based on programming

Computer application engineering is mainly treated as software production. CS is also concerned with hardware, with embedded systems, and proper system composition. Engineering has some common concerns with science. It is however a completely different activitiy[17]. Most of the practical and applied Computer Science is in reality engineering. We note that engineering and ACS are extensively using of models.

(18) The inner structure of models must be static

Tools we use in daily life are constantly adapted. The structure and the composition of these tools change. So far, we assume that the structure of a system is static. For instance, ASM uses a static set of functions. The functions themselves may change. The category of the functions may change among being static, dynamic, private, and public. However, the set of functions is fixed. There is no reason beside convenience why the set of functions should not be dynamic. Practical systems may have functions that are incorporated depending on the state of a running system and especially on the context.

(19) Model-based reasoning is based on deductive systems

The reasoning approach is still often based on some kind of deductive system that allows to derive conclusions from rules, facts, premisses, presuppositions, etc. Model detection already uses inductive and evidential reasoning. Model application also uses plausible and approximative reasoning. For instance, abduction is used as a technique for detection of the best explanation in a given context. Although classical and non-classical mathematical logic has overwhelming success stories we should integrate reasoning systems that are completely different

[17] Engineering is the art of building with completely different success criteria (see [33]: "Scientists look at things that are and ask 'why'; engineers dream of things that never were and ask 'why not'." (Theodore von Karman)).
"Engineers use materials, whose properties they do not properly understand, to form them into shapes, whose geometries they cannot properly analyse, to resist forces they cannot properly assess, in such a way that the public at large has no reason to suspect the extent of their ignorance." (John Ure 1998, cited in [33])

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

from deductive systems. Therefore, model-based reasoning is also based on a combination of various kinds of deductive, inductive, evidential, abductive or more general plausible, and approximative reasoning. Most of these reasoning mechanisms are based on coherence instead of on consistency. Moreover, deduction has to be handled beyond first-order predicate logic despite undecidability, incompleteness, and worse-case complexity of higher-order logics.

(20) AI models may cover human intelligence

Although AI tools are nudging, directing, and misleading us, they do not cover all sides of human intelligence. Neither emotional nor self-reflection or other kinds of human intelligence can be programmed with current approaches. As a matter of fact, human creativity in problem solution is far from being programmable. Mental models are essential for daily problem solution as well as for technical realisations. Therefore, modelling has to properly reflect this kind of intelligence.

4 Models - Towards a General Theory

It's written here: 'In the Beginning was the Word!'
Here I stick already! Who can help me? It's absurd,
Impossible, for me to rate the word so highly.
I must try to say it differently.
If I'm truly inspired by the Spirit. I find
I've written here: 'In the Beginning was the Mind'.
Let me consider that first sentence,
So my pen won't run on in advance!
Is it Mind that works and creates what's ours?
It should say: 'In the beginning was the Power!'
Yet even while I write the words down,
I'm warned: I'm no closer with these I've found.
The Spirit helps me! I have it now, intact.
And firmly write: 'In the Beginning was the Act!'
Goethe, *Faust I*

We will not present approaches that allow to overcome all the myths. The development of a general theory of models and modelling is on the agenda for the next threescore years. Parts and pieces for this theory are already developed.

This agenda includes answers to a number of questions, e.g.:

- *What are the essential properties of models? Can modelling be systematised?*
- *Which artifacts and thoughts are models? Is there any demarcation for the model-being? How can the model-being can be characterised?*
- *What is the main orientation of a model? Can we separate aspects of model use?*
- *Are we using some model or are we using an ensemble of models?*
- *What are the properties of models that are used for interaction in social settings?*

– *What about myths on models that seems to be valid?*

Models are thoughts or artifacts. They are used as instruments at a certain point of time. At other points of time, they might not be models. I.e. thoughts and artifacts have their own journey in the model being. What is then the notion of a model?

The Notion of Model

Let us first introduce a general notion of model. This notion can be specialised to more specific in disciplines. The notion generalises almost all notions or prenotions used and known so far in general model theory [23, 27, 29, 37]. More specific notions can be declined by parameter refinement and hardening from this notion.

"A **model** *is a well-formed, adequate, and dependable instrument that represents origins and that functions in utilisation scenarios.*
Its criteria of well-formedness[18], adequacy[19], and dependability[20] must be commonly accepted by its community of practice (CoP) within some context and correspond to the functions that a model fulfills in utilisation scenarios." [40]

This notion also allows consideration of the model-being of any instrument. We note that the instrument-being is based on the function that a model plays in some scenario. The generality of this notion and its parametric form provides a means for specialisation. For instance, models used in Computer Science are both based on 'normal' models that are adequate from one side, 'deep' models that determine justification, and canonical sufficiency frames for model quality from the other side. For instance, assuming the deep model, quality sufficiency, and the background as definitely given and as being unchangeable, we can restrict the determination of the model-being of an instrument to adequacy. Such assumptions are often made while considering models. Reasoning by models and deployment of models of models becomes problematic if the deep model, quality sufficiency, and the background are implicit.

The Logos, Archē, and Nous of Modelling

The quote we use for this Section is based on reasoning about four of the seven meanings of λόγος (logos): word, concept, judgement, mind, power, deed,

[18] *Well-formedness* is often considered as a specific modelling language requirement.

[19] The criteria for *adequacy* are analogy (as a generalisation of the mapping property that forms a rather tight kind of analogy), being focused (as a generalisation of truncation or abstraction), and satisfying the purpose (as a generalisation of classical pragmatics properties).

[20] The model has another constituents that are often taken for granted. The model is based on a background, represents origins, is accepted by a community of practice, and follows the accepted context. The model thus becomes *dependable*, i.e. it is justified or viable and has a sufficient quality.
Most notions assume dependability either as a-priori given or neglect it completely.

and reason. The notion of logos characterises the model-being. Models must be well-formed (word), based on world-views (concepts), acceptable (judgement), understandable (mind), applicable (power), support their successful use (intended application), and allow to assess their utilisation (reason). We distinguish *perception* and idea (in German 'Auffassung'[21]; including thought, conception, opinion, view, understanding, apprehension, thinking,claim to truth) from imagination and *reflection* (in German 'Vorstellung'; including vision, image, (re)presentation, conceivability, belief, mental image or picture, speech, explanation, justification, accountability, justification, meaningful and -founding spoken/written/thought speech, verifiability, reasonableness, correctness). Essentially, the two kinds of mental models are *perception model* and the *imagination* or reflection *communication model*.

Logos is according to Heraclitus [26] the dynamic, creative, and ordering principle that is injected to perception and reflection. It is subordinated to Nous as 'intellectus', ordering spirit, reason, and rational soul. Archē is the third element in this triptych. It is the general principle of reality and the grounding one in the sense of a fundamental laws. According to the Platon's three analogies (cave, divided line, sun) we cannot fully access it.

Computer Science modelling uses archē as its suite of *domain-situation models* (see the encyclopedic side in the model triptych [30]). The model triptych has its linguistic or representation dimension. This dimension enables model expression. It also hinders it due to the obstinacy of each language. These languages heavily influence modelling due to their hidden grounding and basis. For instance, state-oriented languages with doubtful paradigm of computation-is-state-transformation have a hard time to represent events or continuously changing systems (see also [45] for Sapir-Whorf principle of linguistic relativity).

ACS needs Approaches Beyond Computation

ACS models and CS models substantially differ. Both use reflection and representation models. Prescription ACS models must be as precise and accurate as only possible if they are used as blueprint, plan, antetype, archetype, and realisation worksheet. They must also be more flexible due to adaptation to given environments. For documentation models, the analogy is a homomorphic mapping. ACS models are also used as steering, usage, and guiding models. On the other hand, CS models can also be description, conceptual, investigation, and explanation models. Due to the difference in their function, their adequacy and dependability follow different schemata, norms, and usages, i.e. different canons.

ACS models cannot be context-free or context-limited. They represent a variety of aspects and are thus typically model suites. The UML approach demonstrates this approach by separation of concern. CS models are typically more monolithic and follow the global-as-design approach. For instance, in CS we prefer to use UML class diagrams as the lead model from which the class aspects in other models can be derived as specific views. Additionally, engineering is a

[21] The word fields in German and English languages are different.

construction discipline and thus less – if at all – theory-oriented. However, it is more oriented on technological realisability. Engineering has to produce robust products with well-designed error-tolerance.

Proper Theory of Modelling Languages

Both CS and ACS make intensive use of formal or somehow narrative but orthonormalised languages. Formal languages follow a rigid syntax-first-semantic-second approach. The classical approach to define models within a modelling language that is again defined by a meta-model language is far to strict whenever essential constructions and constructors are not theoretically based (see the OMG approach with models M_1, meta-models M_2, meta-meta-models M_3 etc.). For instance, we know more than half a dozen different IsA relationships while database modellers are used to only one of them. Narrative languages are less rigid and less correct. A typical example of the last kind are standards, e.g. the BPMN standard. In this case, the definition of a formal semantics often becomes a nightmare and requires intensive foundational work in order to become rigid, e.g. [4, 5].

We do not know a theory of language development neither for conceptual languages nor for so-called domain-specific languages although parts and pieces are currently under development, e.g. [15, 42]. Languages must, however, be based on a proper theory whenever languages are used as an enabling tool for modelling in the model triptych approach. Nowadays, languages must also be supported by advanced tools and advanced guidance for their use.

5 Tasks for Next Decades

Enough words have been exchanged,
Let me finally see deeds;
While you are paying compliments,
Something useful can happen.
Goethe, *Faust I*

Overcoming Limits, Obstinacies, and Problematic Solutions

We often hear a claim that ACS&CS are at their dead end. The claim is near to truth due to problematic paradigms and postulates that must be changed soon in the form as envisioned in [9, 25, 42][22]. The AI 4.0 hype illustrates the wild search for a solution of new paradigms. The first step will be a thorough reconsideration of the computation paradigms. For instance, data massives do not need algorithmic number crunching but rather sophisticated 'water supply systems'.

[22] The software crisis has been a crisis 1.0. Nowadays we have a data crisis, a (large and embedded) system crisis, an infrastructure crisis, and an energy crisis. For instance, it is estimated that one third of the world-wide produced electro energy is consumed by computers by 2025.

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

Web computation is based on symbol crunching. Neural networks must be build on the real way how neurons work. We do not need optimal solutions. Instead we can live with approximative solutions near to the optimal. Programming-in-the-small was successful for the current infrastructure. Programming-in-the-large is not yet properly understood. Programming-in-the-world cannot be supported within the current approach. A path towards a solution is proper modelling and model-based reasoning. It might also solve problems of dehumanized algorithmic machines. Sophisticated systems such as AI systems operate without feelings, without heart, without compassion, without conscience, and without ethics. They are simply machines. Human infrastructures need however a different environment based on models that humans use.

Programming-in-the-large and programming-in-the-world have to use new kinds of models. The classical modelling approach to programming-in-the-small has to be incorporated. It needs, however, a deep reconsideration of modelling in the future. Model suites with a number of well-associated to each other models can be one of the solution. As a minimal requirement to a model suite, models representing human approaches and use have to become an integral part.

Modelling to Program and Modelling as Programming

The main usage of models and modelling in CS&ACS is models-for-programming (M4P). There are many initiatives that extend this usage towards modelling-to-program (M2P): Model-driven development, conceptual-model programming, models@runtime, universal applications, domain-specific modelling, framework-driven modelling, pattern-based development, round-trip engineering, model programming, inverse modelling, and reference modelling.

These initiatives are the starting points for a programme for true fifth generation programming that starts with models and then uses models as source code for a program beside of being useful for program specification, i.e. modelling-as-programming (MaP). It is similar to second and third generation programming where programmers are writing programs in a high-level language and rely on compilers that translate these programs to machine code. We propose to use models instead of programs and envision that models can be translated to machine code in a similar way. The approach has already been used in several projects, e.g. for ASMbacked C++ programming and for database development [21, 35].

Models will thus become *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, *validatable* and *verifiable*, and *maintainable*.

Humanised Systems

Web system development [34] taught us a really sad lesson: user are not judging the system by its functions but its form at the first step. They ask whether a system is intuitively understandable and usable. They judge on colours and other

completely irrelevant features instead of functioning issues. Civil engineering has already properly learned this lessons long time ago and found a way how to cope with the problem: design of everyday things, e.g. [31]. With the broadband application we have to go away from the philosophy that the administrator is the only king and the user is at its best the slave.

Models of systems must, thus, be based on the cultures of users, especially on their models.

Application-Governed ACS

CS and ACS experts are used to be the ruler and the guidance for human behaviour. This orientation has been very useful for the first half century of CS existence. Computers became now an essential part of the everyday infrastructure. As such, they are services in applications. A fist step to primary consideration of application objectives instead of requirements analysis is the abstraction-layer macro-model [34] for web system information systems.

Applications bring in their own culture[23] including national and regional, community-backed and corporate, good and problematic habits, professional and educational, ethic and generational, and finally gender, class, and ideological ones. Therefore, models must be adaptable to the specific cultures.

Applications found their way of incorporating very different people with their specific perception, reflection, and imagination. People come with their models that might vary a lot. The common binder is the business. The binder can be used as the starting point for an association schema among the models used. In this case, we have to properly and flexibly handle and adapt models in a model suite.

Applications continuously change and adapt to new ways of operating. That means that change management will be an essential feature for model suites too. Changes might be evolutionary changes, modernisation, migration, or partial replacement with partial cannibalisation. The change with a model suites must also be so robust and flexible.

Paradigms and Postulates for CS and ACS Models

Models, activities to model and to use models, and modelling (MMM) has not yet found its theoretical underpinning. Parts and pieces are already developed in different disciplines [41]. An MMM theory or MMM culture is one of the challenges. CS models can be stereotyped. Modelling can be systematised into a smaller set of canons and moulds by grouping MMM success stories according to those stereotypes. This approach is similar to the design science approach. The rigor cycle [11, 18] aims at summarisation, generalisation, and systematisation

[23] Culture is a "a collective phenomenon, which is shared with people who live or lived within the same social environment, which is where it was learned; culture consists of the unwritten rules of the social game; it is the collective programming of the mind that separates the member of one group or category of people from others."[19]

of modelling experience. Exaptation is some kind of extrapolation of known solutions to new problems in other context.

From the other side, modelling can be based on separation into normal modelling and inheritance of already existing deep models from the same kind of application [39]. Deep models are the inner and inherent part of a model together with corresponding methodologies, techniques, and methods, i.e. its matrices. Modelling, model development, and model utilisation can be concentrated around the normal model.

Modelling themselves follows a number of paradigms, postulates, and principles. They are often implicitly assumed in CS. Reusable postulates (e.g. separation into state, evolution, and interaction for information systems), paradigms (e.g. global-as-design for database structures and views), and principles (e.g. incremental constructivity) support efficient development of normal models. Database structure and functionality modelling reuses and accepts a standardised matrix with development and utilisation methods and some robust methodology.

6 Finally

A marshland flanks the mountain-side,
Infecting all that we have gained;
Our gain would reach its greatest pride
If all noisome bog were drained.
I work that millions may possess this space,
If not secure, a free and active race.
Goethe, *Faust II*

Towards Modelling 2.0 and Modelling 3.0

We may consider the current state of art as the first generation of modelling. The body of MMM knowledge is not yet synthesised into a subdiscipline of CS and ACS. The MMM discipline has to be based on its specific foundations, its specific reasoning methods, its methodologies, and its practices. Model suites are a good starting point for multi-view, multi-abstract, multi-level, and multi-culture modelling. This approach has to be generalised, well-founded for all its extensions, and properly supported by tools and workbenches. Modelling in our discipline is mainly language-based. We, thus, need an extensible but well-integrated collection of languages. Some of them can be oriented on general purpose while most will be domain-specific. Models should be easy to modify and to modernise. Model quality management includes also model analysis depending on functioning in scenarios. This BoK should also include means for certification and industrial licensing.

Models are almost useless if they are one-way-products. Currently, some modelling platforms (e.g. ADOxx) support generation of programs from models, at least for some of the modelling approaches. Modelling 2.0 could be considered as modelling-in-the-world. Modellers come with various cultures, commonsense, thought schools according to the branches of CS and ACS, and various languages.

Collaborative model development cannot be based on a unform language environment. It can use some kind of generic approach that allows to derive specific languages from some source in such a way that the specific adaptation can be reversed and then mapped back to the other partner, i.e. we use an approach with model institutions. Shared model usage is another real challenge. Modelling 3.0 could use model spaces based on solution libraries from which a generic and adaptable model can be extracted similar to the concept of namespace for injection of the meaning of words in Web x.0 .

As stated, models are the fourth dimension of modern CS and ACS. The associations to the other dimensions will become explicit. For instance, MMM 2.0 or 3.0 will be then based on MMM for collaboration or more specifically for communication, for cooperation, and for coordination in various forms and adaptable to various kinds of applications.

Modelling Becomes an Education Subdiscipline in CS and ACS

Modeling is currently an art. It should become a craftsmanship with integration into CS and ACS education. So far, modelling is taught as a side-method for system development. Each subdiscipline has its approach to integrate models and modelling. Some of them use models in an implicit form, some of them in an explicit form. Each subdiscipline founds its specific approach and way of acting. Best practices are used for demonstrating the potential of models. A compilation of this experience is one of the lacunas of current CS and ACS.

Systematic and well-guided modelling should be taught in a specific and separate module in any CS and ACS programme. Modelling will become an object in education and continuing education. A discipline of modelling needs its basis, its methods, its methodologies, and its theory. All this must be adaptable to specific application cases. A modelling discipline includes also proper quality management and handling of errors.

Each subdiscipline has its conception, its concept spaces, its methods, its terminologies, its success and failure stories, its construction approach, its accepted practices, its educational demonstration models, its competencies and skills, its approaches, and its theoretical underpinning. MMM as a module will be not different.

Deploying the Full Capacity and Potential of Models

The utility, potential and capacity of models is not yet fully understood and deployed. Models are used for complexity reduction, for communication within the communities of practice, for efficiency increase during development, and for mediating during development. We may launch out model suites into risk minimisation, generation of solutions directly from models, and handling interoperability among different systems. Models should become an integral component of delivery software and hardware.

Models will become a proper companion during all development steps for integrated team collaboration, for quality management, for forecasting performance

Logic, Computation and Rigorous Methods LNCS 12750, 301-325

and bottlenecks, for generating of neatly integrable partial solution. Model suites have to reflect a large variety of viewpoints in a well-associated form since applications become more and more multi-disciplinary and interdisciplinary. Modelling-in-the-world should allow to tolerate heterogeneity in organisations that are outside the control and understanding of the modelling team.

Heritage (currently often called 'legacy') models and reference models can be used for experience propagation. Componentisation of models shall support re-usage and recycling of already existing high-quality solutions. Domain-specific languages may be used for development of inheritable and adaptable domain-specific model suites and their integration into solutions. Models in a model suite also represent a variety of viewpoints within a community of practice. Tight association of models in suites allows to concentrate on certain issues and aspects without losing the coherence in the model suite. Model suites have to support multi-abstraction, multi-levels, and multi-cultures among all partners.

In order to serve the variety of viewpoints and cultures within a community of practice, a model suite should integrate a number of informative models for each member as well a number of representation models in a variety of languages. These different models may represent the same aspects and features but in different forms due to the differences in the languages used and preferred by some members in the community of practice. Models for theoreticians might be qualitative. Models might also concentrate around quantitative aspects for data practitioners. We need then a 'matching' theory for the two orientations.

Call for Contribution and Research

Wolfgang Hesse (see also [16, 17]) summarised the consternation about modelling in the Modellierung 2009 workshop:

- ... but they do not know what they do ...;
- Babylonian language confusion and muddle;
- "It's not a bug, it's a feature", de-facto-standards and lobbyists;
- Why I should cope with what was the state of art yesterday;
- Each day a new wheel, new buzzwords without any sense, and a new trend;
- Without consideration of the value of the model;
- Competition is a feature, inhomogeneity;
- Laokoon forever;
- Dreams about a sound mathematical foundation;
- Take but don't think - take it only without critics;
- Academia in the ivory tower without executable models;
- Where is the Ariadne thread through?

This lead directly to a number of research and development issues: Can we develop a simple notion of adequateness that still covers the approaches we are used in our subdiscipline? Do we need this broad coverage for models? Or is there any specific treatment of dependability for subdisciplines or specific deployment scenarios? Which modelling methods are purposeful within which setting? Which model deployment methods are properly supporting the function

of a model within a utilisation scenario? How does the given notion of model match with other understandings and approaches to modelling in computer science and engineering? What is the background of modelling, especially the basis that can be changed depending on the function that a model plays in some utilisation scenario? Language matters, enables, restricts and biases. What is the role of languages in modelling? Which modelling context results in which modelling approach? What is the difference between the modelling process that is performed in daily practice and systematic and well-founded modelling? Are we really modelling reality or are we only modelling our perception and our agreement about reality? What is the influence of the modeller's community and schools of thought?

The current situation is not really different from 2009. We need a lot of good research contributions.

What you put off today will not be done tomorrow;
You should never let a day slip by
Let resolution grasp what's possible
And seize it boldly by the hair;
Then you will never lose you grip,
But labor steadily, because you must.
Goethe, *Faust I*

References

1. D. Abts and W. Mülder. *Grundkurs Wirtschaftsinformatik : Eine kompakte und praxisorientierte Einführung*. Vieweg, 2004.
2. S. Alter. Work system theory and work system method: A bridge between business and IT views of IT-reliant systems in organizations. In *Proc. ISEC 2017*, page 211. ACM, 2017.
3. E. Börger and R. Stärk. *Abstract state machines - A method for high-level system design and analysis*. Springer, Berlin, 2003.
4. E. Börger and B. Thalheim. A method for verifiable and validatable business process modeling. In *Software Engineering*, LNCS 5316, pages 59 – 115. Springer, 2008.
5. E. Börger and B. Thalheim. Modeling workflows, interaction patterns, web services and business processes: The ASM-based approach. In *ABZ*, volume 5238 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 2008.
6. N. Bourbaki. Foundations of mathematics for the working mathematician. *J. Symb. Log.*, 14(1):1–8, 1949.
7. F. Brentano. *Psychologie vom empirischen Standpunkte*. Leipzig: Dunker & Humblot, 1874.
8. S. Chadarevian and N. Hopwood, editors. *Models - The third dimension of science*. Stanford University Press, Stanford, California, 2004.
9. R.S. Cohen and T. Schnelle, editors. *Cognition and fact: Materials on Ludwik Fleck*, volume 87 of *Boston Studies in the Philosophy of Science*. Dortrecht, Reidel, 1986.
10. A. Dahanayake and B. Thalheim. Co-evolution of (information) system models. In *EMMSAD 2010*, volume 50 of *LNBIP*, pages 314–326. Springer, 2010.

11. A. Dahanayake and B. Thalheim. Development of conceptual models and the knowledge background provided by the rigor cycle in design science. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 3–28. College Publications, 2018.
12. P. J. Denning, D. Comer, D. Gries, M. C. Mulder, A. B. Tucker, A. J. Turner, and P. R. Young. Computing as a discipline. *Computer*, 22(2):63–70, 1989.
13. R. S. Engelschall. Schönheit und Unzulänglichkeit von Software-Architektur. In *Software Engineering 2021*, volume P-310 of *LNI*, page 19, Video accessible via https://www.youtube.com/watch?v=AMOmvtsCRoU, 2021. Gesellschaft für Informatik e.V.
14. R.S. Engelschall. Manifesto for true software engineering. http://true-manifesto.org/, 2021. Accessed March 8, 2021.
15. U. Frank. Domain-specific modeling languages: Requirements analysis and design guidelines. In *Domain Engineering, Product Lines, Languages, and Conceptual Models*, pages 133–157. Springer, 2013.
16. W. Hesse. Modelle - Janusköpfe der Software-Entwicklung - oder: Mit Janus von der A- zur S-Klasse. In *Modellierung 2006*, volume 82 of *LNI*, pages 99–113. GI, 2006.
17. W. Hesse and H. C. Mayr. Modellierung in der Softwaretechnik: eine Bestandsaufnahme. *Informatik Spektrum*, 31(5):377–393, 2008.
18. A. Hevner and S. Chatterjee. *Design research in formation systems*. Springer, Berlin, 2010.
19. G. Hofstede, G.J. Hofstede, and M. Minkow. *Cultures and Organizations: Software of the Mind: Intercultural Cooperation and Its Importance for Survival*. McGraw-Hill, New York, 2010.
20. H. Jaakkola and B. Thalheim. Cultures in information systems development. In *Information Modelling and Knowledge Bases XXX*, volume 312 of *Frontiers in Artificial Intelligence and Applications*, pages 61–80. IOS Press, 2019.
21. H. Jaakkola and B. Thalheim. Models as programs: The envisioned and principal key to true fifth generation programming. In *Proc. 29'th EJC*, pages 170–189, Lappeenranta, Finland, 2019. LUT, Finland.
22. M. Kangassalo and E. Tuominen. Inquiry based learning environment for children. In *Information Modelling and Knowledge Bases XIX*, volume 166 of *Frontiers in Artificial Intelligence and Applications*, pages 237–256. IOS Press, 2007.
23. R. Kaschek. *Konzeptionelle Modellierung*. PhD thesis, University Klagenfurt, 2003. Habilitationsschrift.
24. D.E. Knuth. *The art of programming I-III*. Addison-Wesley, Reading, 1968-1973.
25. T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, Illinois, 1962.
26. A.V. Lebedev. *The Logos Heraclitus - A reconstruction of thoughts and words; full commented texts of fragments (In Russian)*. Nauka, Moskva, 2014.
27. B. Mahr. Information science and the logic of models. *Software and System Modeling*, 8(3):365–383, 2009.
28. B. Mahr. Intentionality and modeling of conception. In *Judgements and Propositions - Logical, Linguistic and Cognitive Issues*, pages 61–87. Berlin, 2010.
29. B. Mahr. Modelle und ihre Befragbarkeit - Grundlagen einer allgemeinen Modelltheorie. *Erwägen-Wissen-Ethik (EWE)*, Vol. 26, Issue 3:329–342, 2015.
30. H. C. Mayr and B. Thalheim. The triptych of conceptual modeling. *Softw. Syst. Model.*, 20(1):7–24, 2021.
31. D. A. Norman. *The Design of Everyday Things*. Doubleday, New York, 1990.

32. J.E. Safra and J. Aquilar-Cauz et. al, editors. *Encyclopædia Britannica Ultimate Reference Suite*, chapter Computer Science. Chicago: Encyclopædia Britannica, 2015.
33. A. Samuel and J. Weir. *Introduction to Engineering: Modelling, Synthesis and Problem Solving Strategies*. Elsevier, Amsterdam, 2000.
34. K.-D. Schewe and B. Thalheim. *Design and development of web information systems*. Springer, Chur, 2019.
35. J. Schmid. Compiling Abstract State Machines to C++. *J. Universal Computer Science*, (11):1069–1088, 2001.
36. H. Simon. *The Sciences of the Artificial*. MIT Press, 1981.
37. H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.
38. B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.
39. B. Thalheim. Normal models and their modelling matrix. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 44–72. College Publications, 2018.
40. B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, pages 123–139. Springer, 2019.
41. B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.
42. O. Thomas, S. Hagen, U. Frank, J. Recker, L. Wessel, F. Kammler, N. Zarvic, and I. J. Timm. Global crises and the role of BISE. *Bus. Inf. Syst. Eng.*, 62(4):385–396, 2020.
43. K. Twardowski. *Zur Lehre vom Inhalt und Gegenstand der Vorstellungen: Eine psychologische Untersuchung*. Wien: Hölder, 1894.
44. S. Wenzel. Referenzmodell für die Simulation in Produktion und Logistik. *ASIM Nachrichten*, 4(3):13–17, 2000.
45. B.L. Whorf. *Lost generation theories of mind, language, and religion*. Popular Culture Association, University Microfilms International, Ann Arbor, Mich., 1980.
46. L. Wittgenstein. *Philosophical Investigations*. Basil Blackwell, Oxford, 1958.
47. W. Zimmermann and B. Thalheim. Preface. In *ASM 2004*, number 3052 in LNCS, pages V–VII, Berlin, 2004. Springer.

Remark: See too the presentations in https://vk.com/id349869409 or in the youtube channel "Bernhard Thalheim" for the theory and practice of modelling.

The Triptych of Conceptual Modeling

A Framework for a Better Understanding of Conceptual Modeling

Heinrich C. Mayr[0000-0001-5770-8091] (Alpen-Adria-Universität Klagenfurt, Austria) and
Bernhard Thalheim[0000-0002-7909-7786] (Christian-Albrechts-Universität Kiel, Germany)

Abstract

We understand this paper as a contribution to the "anatomy" of conceptual models. We propose a signature of conceptual models for their characterization, which allows a clear distinction from other types of models. The motivation for this work arose from the observation that conceptual models are widely discussed in science and practice, especially in computer science, but that their potential is far from being exploited.

We combine our proposal of a more transparent explanation of the nature of conceptual models with an approach that classifies conceptual models as a link between the dimension of linguistic terms and the encyclopedic dimension of notions. As a paradigm we use the triptych, whose central tableau represents the model dimension. The effectiveness of this explanatory approach is illustrated by a number of examples. We derive a number of open research questions that should be answered to complete the anatomy of conceptual models.

Keywords: Conceptual Modeling, Modeling Languages, Model Characteristics, Model Hierarchies, Language Hierarchies, Concept, Notion, Term

1 Introduction

Perception and abstraction, i.e. "modeling", and reasoning on models are basic human capabilities for coping with, understanding, and influencing the environment. Over time, many types of modeling have evolved: from completely intuitive to highly controlled ones that apply a specific set of terms forming the semantic instruments of a (modeling) language.

Natural language enables us to describe, communicate or understand perceptions and thus supports a moderately controlled modeling: the language elements (words, phrases, texts, icons), their composition and meaning are tacitly agreed upon by the users and, to a certain degree, are shared among them. The assignment of meaning to language elements, however, is sometimes ambiguous, the syntactical rules are not strict throughout. Elements, syntax and interpretation change over time.

In contrast to that, scientific disciplines, in particular mathematics, introduce strict formal languages and propose semantic interpretations to the lexical elements and their syntactic composition. An illustrative example of such a formal approach is the Petri Net Language as initially introduced by Carl Adam Petri [Pet62]: A special type of bipartite directed graphs is provided together with some composition rules, and a family of functions (“marking” and “transition”). Applying standard Linear Algebra mechanisms to this leads to a powerful calculus. However, this calculus has no semantics at all! In order to make Petri Nets usable for modeling we need to provide a “net interpretation”, i.e. to associate semantics to the language elements. Most popular is to interpret one type of nodes (the places) by Conditions and the other type by Events. The marking functions then describe possible situations by means of valid or invalid conditions; the transition function describes occurrences of events and their consequences.

In natural or technical sciences, this approach is reversed: initially, a conceptualization[1] of the domain of interest is established and subsequently one or more (textual and/or graphical) languages are defined for representing its elements and relationships. Think for instance of a conceptualization of electrical components that are represented using electrical circuit diagrams. The Unified Modeling Language UML comes with a conceptualization of abstract elements like class, attribute, relation, state, activity for describing domains of interest. Similarly, the Business process Model and Notation BPMN comes with a conceptualization abstract elements like actors, activities, or decisions. A branch of Knowledge Engineering deals with so-called action languages that are based on the claim that "*action theories always model - explicitly or implicitly - the general notions of time, change and causality*" [PP19].

Often, such languages are called "*conceptual modeling languages*" and their use as "*conceptual modeling*" – although despite countless attempts, there is no generally used strict definition of what constitutes conceptual modeling and what does not.

One group of such definition attempts are variants of "*Conceptual Modeling is Modeling with Concepts*" [Kan15, Tha18], and

- introduce these concepts via more or less rigid ontological frameworks, or by simple explanation using natural language; [vF72, vF91] called this latter approach "a priori semantics";
- propose more or less formalized constructs for representation, i.e. a 'modeling language";
- and often call the approach "semiformal", an awful wording per se as it just indicates, that the proposed framework does not fulfil the criteria demanded for a consistent calculus that can be used for correctness proofs etc.

This way of defining "conceptual modeling", however, does not provide hard criteria for differentiating it from other modeling methods in individual cases. For example, most conceptual modelers would say that programming or relational database design is not conceptual modeling (see, e.g. [Myl20]). Nevertheless, programming languages or the SQL DDL work with conceptualizations, the latter for instance featuring elements like "Relation" or "Attribute" that have some basic semantics and therefore might be seen to be concepts in the above definition's sense. From a practical point of view, this open question is not a real problem. However, the term "Conceptual Modeling" is widely used, and there has been an international conference with this name for 39 years. Therefore, it would be desirable to have a definition or at least a set of criteria at hand that would allow us to define more precisely what is and what is not a conceptual model. We will return to this question in section 4.

Recent initiatives (e.g. [DLPS18, GGM20, Tha18]) try to compile and analyze systematically existing definitions and opinions in order to filter out a better understanding of the nature of conceptual modeling. [Myl20] offers "*three complementary theses, answers to the question 'What is a conceptual model?'*". These theses essentially state that conceptual models are (1) computational because they are stored in computers and are analyzed and justified by computers, (2) artifacts, so they should have requirements dictated by Engineering, and (3) social artifacts, because they must capture the common conceptualization of a group.

In fact, from the very beginning, conceptual modeling was propagated as a means to improve the design and implementation of whatsoever software system, especially with regard to a

[1] http://www.webster-dictionary.org/definition/conceptualisation (accessed on August 3rd 2020): "*A conceptualisation is an abstract, simplified view of the world that we wish to represent*"

comprehensive and as clear as possible elicitation and analysis of system requirements. Until now, however, the practical use has mostly taken place at the level of mere drawings, which do not play a major role in the further development process and are rarely adapted to changes. Consequently, the developed software (nota bene: again a model) usually deviates considerably from what was originally modeled. The situation in Business Process Modeling is not much different, independent of the modeling method used (like BPMN, Adonis, Event Chains etc.). This means that the potential of conceptual modeling is far from being exploited.

MDA/MDSD approaches [KWB03] and models@runtime [BG18] are enforcedly more aligned with the system life cycle as they use the models for generating or driving the targeted software. Usually, they work with well-defined subsets or variants of known modeling methods [FR07, GR19, and PR18]. However, also these approaches do not have a breakthrough in practice.

Worse still, university graduates who highly motivated join a company often quickly lose their enthusiasm when they are told that modeling is too expensive in terms of effort and cost, not paid for by the customer, and has no impact on the quality of the software development process, since "agile developers" know what they are doing.

We assume that all this is mainly due to the fact that inventors and propagandists of conceptual modeling languages like ourselves have so far failed to make the anatomy of conceptual modeling and its benefits transparent to users. Instead, we invented hundreds of variants of "modeling languages" always believing that it should be a must for the targeted user to acknowledge and happily exploit the miracle we presented to her/him. Moreover, uncountable papers present what they call "ontologies" and expect the readers to internalize and share these without contradiction. Others implicitly equate conceptual modeling with "graphical modeling" (KM20) and thus not only add to the confusion but also distort the view of the essential.

We believe, therefore, that in order to make conceptual modeling more attractive for practitioners, we have (1) to provide a clear conception of what we are speaking about, (2) to make the anatomy of conceptual modeling transparent with its principles, paradigms, postulates, assumptions, particularities, specifics, potential, capacity and limitations, and (3) to allow the modelers to easily create and use their own domain and culture tailored modeling language and method instead of forcing them to learn and deal with ours.

With this paper we would like to make a contribution to these To-do's. We offer here our understanding of what modeling, in particular, conceptual modeling is about, and how we can clearly distinguish it from other modeling approaches. The perspective presented reflects four decades of dealing with conceptual modeling in research and practice, countless discussions with colleagues and practitioners, the rich body of knowledge published up to now, as well as long and intensive working meetings the authors had over the last two years. But we have neither the intention to improve the world nor to provide an n+1st definition of what "conceptual modeling" is. Rather, we present a "signature" of conceptual modeling in the sense of a framework of characteristics by which conceptual modeling can be categorized. In other words, we will offer an explanatory framework that could help to better understand the nature of conceptual modeling.

The paper is structured as follows: Section 2 explores specifics of models and introduces six characteristics that can be observed for models. Section 3 refines these six characteristics in terms of a list of criteria that can be used to determine the nature of CM. In section 4 we summarize the two previous sections and discuss the first conclusions that can be drawn from them. This will provide the basis for section 5, where we present the core message of the paper: the triptych

paradigm of conceptual modeling together with its dimensions and model/language hierarchies. The paper ends with a conclusion and an outlook on open research challenges in section 6.

We will reference related work where appropriate but, intentionally, there will be no separate section on related work. Instead, we refer to the rather comprehensive overview given by Thalheim in [Tha18], to [Wik17] and to attempts to define the term "model" [TN15a].

Finally, we would like to point out that, for the sake of readability, we also adopt the usual homonymous use of the term "model" in this paper: From an epistemological point of view, a model is a mental object. In practice, however, the representation of a model introduced into the perceivable world is also referred to as a model, like, e.g. an Entity-Relationship diagram. We adopt this homonymy because the particular meaning will result from the respective context.

2 Characteristics of Models

Across disciplines, the number of publications dealing with models, modeling and abstraction are unmanageable. Even for the notion of "conceptual model" more than 60 different definitions can easily be found [Wik17, Tha18, Myl20]. None of these, however, allows for a robust and unequivocal differentiation between conceptual and non-conceptual models. This is also true for an interesting definition that recently emerged in a side-piece discussion at ER 2017: "*A conceptual model is a partial representation of a domain that can answer a question*". For, it only highlights one aspect.
We, therefore, try to elaborate the essence of conceptual modeling in the form of a taxonomy of characteristics that may help to better delimit the semantics of the term "conceptual model".

Before we can do this, we first need to take a closer look at the terms "concept", "notion" and "term". The reader will have noticed that we have avoided their use as much as possible so far. The reason for this is that the meanings of these terms in literature and in encyclopedias are not sharply delineated, so that there are overlapping or synonymous definitions. For the purposes of this paper, however, we need a more precise distinction (which will be further specified in section 5). We, therefore, assume the following meanings in the subsequent sections:

- A ***Concept*** is a mental construct formed by mentally combining characteristics of general or abstract ideas gained by cognition. It is seen as a pair of an intension and its extension. The intension describes the concept as such, the extension consists of all objects that might be used as an example for the intension. This definition is based on [We20] (a concept is "*something conceived in the mind*"), Wordnet [MBF90] ("*an abstract or general idea inferred or derived from specific instances*"), the Stanford Encyclopedia of Philosophy[2] ("*concepts are constituents of thoughts*"), [BMS86, Kan15] and [Mur01].
- A ***Notion*** is a general inclusive concept in which some confidence is placed; i.e. a notion is a specific kind of concept[3]. This definition is based on Wordnet ("*a notion is a general understanding, vague idea or a general inclusive concept in which some confidence is placed*") and [We20] who propose "*arriving at the notion of law*" as an example for the interpretation of notion as a general inclusive concept.

[2] https://plato.stanford.edu/entries/concepts/
[3] With this interpretation *notion* corresponds to the German "*abstrakter Begriff*" [KB71] as "*mental and abstract reflection of a class of individuals or classes on the basis of their invariant characteristics … i.e. specific concepts as abstract essences … (ideas)*".

- A ***Term*** is an item of a (possibly formal) language formed for denoting, designating, or naming something. "Language" is understood here in a very broad sense, i.e. it can be textual, graphical but also material. Terms can refer to concepts and/or represent them for recognition by linguistic perception processes, i.e. processes mapping a term/symbol to a mental object. Note that this interpretation of term is inspired by one of the definitions given in [We20] ("*a pronounceable series of letters having a distinct meaning especially in a particular field*"), but differs from others. We use it here to clearly separate "term" from "concept" and "notion". Also, for the sake of clarity, we will not use any other word with a similar meaning throughout the paper (such as for example "sign").

Second, as conceptual models are models, we have to agree on the key characteristics of models before specializing and extending these to determine what the characteristics of conceptual models are. For this purpose, we adopt the main criteria provided by [Mah05] that may be summarized by "*A model is the synthesis of a conceptual idea, a form of expression and the assumption of a role through which it fulfils a function*"[4].

Model Characteristic 1: Models are related to (a collection) of "origins" or "originals". A model is a model of something[5], i.e. it is a proxy of a natural, artificial or mental original; in particular, the original of a model may be a model itself. As originals may change in time, the model/original relationship may change in time as well [Sta73]. Models are results of cognitive processes (perception) [vF72]. The mission of a model is that of transporting a "*cargo*", namely the perceived properties of the original that are considered to be relevant within the perception's context. Mahr sees this function as the key criterion for a "*model being a model*" [Mah15]. The transport occurs with the usage of the model, precision and transport warranties distinguish models and metaphors [Mah08].

Model Characteristic 2: **Concern and Usage**. We distinguish three different main concerns that are coupled to most kinds of modeling: (1) understanding, (2) communicating, and (3) agreeing as a process of consolidation, manifestation, and consensus. With the usage, a model unfolds its power: "*We place models between ourselves as perceiving, recognizing, understanding, judging or acting subjects and the world as perceptible, observable, effective, to be judged or produced exterior. The impact of models results from the role that models play through their transport function in work processes, cognitive processes, business processes. The power of models is the result of their power to act"*[6][7] [Mah05]. In general, the usage of a model will be directed by its initial concern. However, this is not mandatory, because the using individual can do what she/he wants with a model.

Model Characteristic 3: **Purpose and Function**. Given its concern and usage, a model serves a particular purpose: to understand/analyze/assess the origin, to plan/design a new original, to explain or predict properties of the original, to communicate about perceptions and ideas, and

[4] Original quotation in German: „*Ein Modell ist die Synthese einer begrifflichen Vorstellung, einer Ausdrucksform und einer Einnahme einer Rolle, durch die es eine Funktion erfüllt*." Bernd Mahr cites here George A. Millers work „ *The science of words* ", which was not accessible for us directly [Mil91].
[5] "*Every mental phenomenon has an object towards it is directed*" [Bre74].
[6] Original quotation in German: „*Wir stellen Modelle zwischen uns als wahrnehmende, erkennende, verstehende, urteilende oder handelnde Subjekte und die Welt als wahrnehmbares, beobachtbares, wirkendes, zu beurteilendes oder herzustellendes Äußeres. Die Wirkungsmacht von Modellen ergibt sich aus der Rolle, die Modelle durch ihre Transportfunktion in Werkprozessen, Erkenntnisprozessen, Unternehmensprozessen spielen. Die Macht von Modellen ist das Ergebnis ihrer Wirkungsmacht".*
[7] Translated with www.DeepL.com/Translator

similar. The usage determines the function(s) of a model, for example to support explanation. It therefore makes sense to see a model's function as that of an "*instrument*" [Tha19, TN15b].

Model Characteristic 4: **Domain and Context**. For the concerns of modeling, we distinguish the following three domains:

a. the domain of interest, experience, and perspective of a human,
b. the application domain or world domain to which a community of practice refers,
c. the domain of discourse among some particular people.

The first domain is concerned with understanding and thinking. So is the second one which additionally is concerned with realization (in the sense of implementation). The third domain is concerned with communication.

A model is created, modified or refined in particular contexts: The personal context of the modeler, the environmental context in which the modeling process takes place, the social context, i.e. the particular community of practice, and the spatio-temporal context (time, duration, location, and movement etc.) [MM13, MM16].

Clearly, a model's cargo as well as its interpretation depends on the given concern, purpose, domain and context.

Model Characteristic 5: Focus. A model reflects, for a given purpose, the "relevant" but not all aspects of its origin(al). In particular, "*The objectual properties may recede behind the consideration of their rational-functional relationships*" [Wol96]. Note, that this is a more general view than that of [Sta73], who emphasizes on reduction, i.e. differentiates between "modeled attributes" and "neglected attributes" of the origin(al).

Model Characteristic 6: Representation. For communication/transportation purposes, a model needs an associated "physical" representation; examples are an acoustic signal, a toy railroad, a diagram, a XML statement, an OWL file, a spoken/written natural language text, and so forth. These representations allow models to be recognized and understood by communication partners; in the case of a human partner, recognition is enhanced by "linguistic perception" [vF72]. The representations should be dependable, understandable by the involved actors (humans and/or systems), and thus be agreed within the community of practice. George A. Miller explained the relationship between a model and its representation as follows: "*To have a model means to be able to produce or recognize a physical symbol carrier that represents a model, and to understand the meaning of the model*" [Mil91].

Clearly, this taxonomy is not complete, as the literature addresses many more characteristics. For example, see the "Kiel house of modeling" [TN15a][8]. However, it should not be a problem to classify most of them in relation to the characteristics presented.

3 Characteristics of Conceptual Models

Conceptual models are models, conceptual modeling is (a kind of) modeling. Consequently, the characteristics described in section 2 also apply to conceptual models. So we need to identify what constitutes the specialization "conceptual". To this end, we will now, wherever possible, specialize

[8] http://bernhard-thalheim.de/ModellingToProgram/

the above model characteristics and introduce two more that we believe are specific to conceptual models.

CM Characteristic 1: Conceptual models are related to (a collection of) origins or originals. As there is no restriction on the entirety of origin(al)s conceptual models may relate to, this characteristic does not provide an indication for differentiation.

CM Characteristic 2: **Concern and Usage**. In the discipline of Informatics the term "Conceptual Modeling" has been initially used for a database design method, later on for requirements modeling and since the 90ies for business process modeling and software specification. In all cases, the mapping from conceptual models (represented using languages like the ERM, UML, BPMN, SysML etc.) to an implemented system language (SQL DDL, programming languages, workflow languages etc.) has been a key issue until today. Model Driven Software Development (MDSD) [SK03], Model Driven Architecture [KWB03] as well as models@runtime [BG18] all start from conceptual models and aim at materializing and automating that mapping. Model Centered Architecture (MCA) [MMR17] advocates, for any aspect of a system under development, the use of Domain Specific Modeling Languages (DSML), i.e. focuses on models (and their metamodels) in any design and development step up to the running system. In summary, conceptual modeling has a strong (although not mandatory) orientation to a subsequent implementation/realization of artifacts or products. It, therefore, is widely used as a means for requirements modeling and analysis. Consequently, we may add the **concern (4) "specifying"** to the list of concerns. Unlike [Myl20], however, we do not claim that conceptual models are 'computational' - and that they have only existed since computers have existed.

CM Characteristic 3: **Purpose and Function**. As a consequence of extending the concern we supplement the purpose "plan/design a new original" with "plan/design/realize".

CM Characteristic 4: **Domain and Context**. There is no principal limitation regarding domains and contexts of conceptual modeling so that also this characteristic provides no hard criterion for differentiation. However, in practice, conceptual modeling has been mainly used so far in domains and contexts that deal with discrete objects (things, actions), their properties and relationships. Again, however, this is no strong criterion for differentiation.

CM Characteristic 5: Focus. Conceptual models have no noteworthy peculiarity regarding this general model characteristic. In practice, however, their focus has been mainly on aspects that can be realized or implemented.

CM Characteristic 6: Representation. Conceptual models transport semantics by terms that denote concepts. As terms are elements of languages, conceptual modeling uses linguistic representations in the broadest sense: these may originate from a diagrammatic language (e.g. ER diagrams), a natural language, an artificial language (e.g. XML), a mathematical or formal language (e.g. Petri nets in the sense of algebraic structures). Such languages provide a set of literals and a set of rules for composing literals to terms, terms to phrases, and phrases to sentences and so on. If the members of a certain community of practice have agreed on the meaning of terms or patterns and their combination, they can infer from these to the transported concepts.

CM Characteristic 7: Concept Space. From its beginnings, conceptual modeling had a strong relation to semantics. Partly, "semantic modeling" using terms that are associated with concepts from a "concept space" is even used as a synonym [FGH92] of conceptual modeling: A community of practice agrees on the terms and concepts, which it will consider, as well as on the association

between these terms and concepts, and thus establishes an instrument for communication. The terms used for representing models thus have a meaning, the "a priori semantic" [vF91].

So we can consider conceptual models as models that are "enhanced" by concepts from a concept space. I.e., the decision to compile and accept a set of concepts and to use its elements for relating them to models opens the entrance into the world of conceptual modeling (we will discuss this in detail within the next chapters). This characteristic governs all others and reminds to the definition "*Conceptual Modeling is modeling with concept*s" cited in the introduction. Reflecting the considerations presented so far, a more apposite description could be "*Conceptual Modeling is modeling with concepts from an associated concept space*".

Such association provides a semantical basis supporting understanding communicated models within a community of practice (see CM characteristic 2): as a prerequisite, this community agrees in advance on a set of concepts to be used for modeling, their meaning and representation (controlled vocabulary). Usually this is done informally in natural language, i.e., relating an explaining natural language phrase to the given concept. As an example think of Peter Chen's explanation of the concepts "entity" and "relation" [Che76]: "*An entity is a "thing" which can be distinctly identified. A specific person, company, or event is an example of an entity. A relationship is an association among entities. For instance, "father-son" is a relationship between two* 'person' entities". I.e., the semantics of natural language - and thus its intrinsic a priori [Pla03, Kan81, Lat17] knowledge - are used to determine (the meaning of) concepts.

Therefore, the degree of common understanding of the elements of a concept space by the members of a community of practice depends on the degree of equivalence of their understanding of the natural language used. As such equivalence cannot be formally derived or proven without a reference mechanism like an ontology or a set of axioms, the "a priori semantics" [vF72] of conceptual models provide a practically useful but formally inaccurate means for communication. Some people, therefore, call conceptual models "semiformal" as has been mentioned in the introduction.

CM Characteristic 8: Concept Relationship. Concepts can be related to each other. Typical concept relationships are the "abstractions" [SS77] Mereology (Aggregation), Generalization, and Intension, each of them having an inverse: Disassembly (into components), Specialization (by additional concept attributes), Extension (denominating the elements characterized by their intension concept) [LMN93, LMW79]. Other concept relationships are, e.g., synonymy, homonymy, troponymy, hyponymy; however, these only concern the level of assigning linguistic denoters to concepts. Therefore, such relationships can be found in thesauri and encyclopedias, since they occur at the linguistic level.

Note that a conceptual modeling language that offers explicit means for modeling the intension/extension relationship[9], supports "multi-level" modeling. For, models then are not mere extensions of a given meta-model but may consist themselves of intension/extension concept hierarchies. In the field of Domain Specific Modeling this possibility is often neglected: Metametamodel abstraction relations are just used for relating metamodel concepts, but metamodels often do not explicitly provide such relations for allowing the same on the modeling level, i.e., the relationships are not introduced as part of the concept space in question.

[9] Intension/Extension are the concept relationships establishing model hierarchies with levels like metametamodel |metamodel | model | instance as provided, e.g., by the OMG MetaObject Facility [OMG] or the ISO Information Resource Dictionary System [ISO90.]

Table 1 summarizes these considerations on model characteristics.

Characteristics	**Model**	**Conceptual Model**
Relation to origins	A (conceptual) model is a model of something	
Concern and Usage	(1) understanding (2) communicating (3) agreeing	(1) - (3) + (4) specifying
Purpose and Function	(1) understand, analyze, assess (2) plan, design (3) explain, explore, predict, use	(1) - (3) + (4) realize
Domain and Context	Domain: (1) domain of interest, experience, and perspective of a human (2) application domain or world domain accepted by a community of practice (3) domain of discourse among some people Context: (1) personal context of the modeler (2) environmental context in which the modeling process takes place (3) social context, i.e. the particular community of practice, and (4) spatio-temporal context	
Focus	aspects of the origin(al) that are "relevant" for a given purpose	not mandatory but lived practice: aspects that can be realized or implemented
Representation	by physical symbol carriers	lived practice:-(in the broadest sense) linguistic terms
Concept Space		is associated with concepts from a concept space: a-priori semantics
Concept Relationship		semantic relationships between concepts induce semantic relationships between conceptual models.

Table 1: Characteristics of Models and Conceptual Models

4 Some initial results of using the characteristics

In this section, we use some examples to show how the previously introduced CM characteristics can be used to decide for a given model whether it is conceptual or not.

First, however, we note that models based on model hierarchy frameworks such as the Information Resource Dictionary IRDS [ISO90]) or the MetaObject Facility MOF [OMG] are not conceptual per se, although the model hierarchies are induced by concept relationships according to CM characteristic 8. For, a metamodel (on a hierarchy level H^{n+2}, e.g. on MOF Level M2) specifies modeling elements ("modeling concepts" in [LM78]) and their relationships but neither automatically nor explicitly associates these with a concept space in the sense of CM Characteristic 7. This would also not change if we cast the whole thing in languages, i.e. create linguistic means of expression for the formulation/representation of models on level H^{n+1} or model extensions on level H^n by defining corresponding grammars[10].

[10] Please note that we use a more general form of model hierarchy here, which can have any number of levels, possibly even nested ones. For practical purposes, especially for system design, the MOF or IRDS levels are of course sufficient, i.e. n=0 in this case.

The **Entity-Relationship-Model** (in the sense of a metamodel) thus only becomes "conceptual" when the meaning of the terms “entity set”, “relationship set”, “attribute” etc. is at least colloquially explained (a priori semantics) in an associated concept space. The same applies to the metamodel of the **UML**. In other words, modeling with UML is conceptual if the a priori semantic explanation of what is meant by "class", "relation" etc. is associated with the models. Pure drawing of diagrams or "graphical modeling" is not conceptual.

But what about the question we raised in the introduction: “*Is the **Relational Data Model** a conceptual one?*” Traditionally, answering this question was avoided by introducing the notion of „logical” model, i.e. a representation that is based on a “logical” language and the semantics defined by usage (or implementation). [Myl20] makes a more specific statement on this: “*Relational schemas are not conceptual as well, because they say nothing about the meaning of data in a database, only about its structure*.” At first sight, one seems to be able to get along with this statement, but we do not want to accept it as a generally valid one. For, the situation is identical to the situation described above regarding UML: if there is (within the given community of practice) a common accepted view on the meaning of the terms “relation”, “column”, “row”, “attribute” and so forth, the Relational Data Model (a metamodel) is a conceptual model, as it’s elements have an associated concept space. A relational schema represented in SQL DDL (on level H^1) and introducing common concepts (e.g. a table called „client“) then allows us to infer that client is an extension of the concept relation and has attributes (columns) describing clients’ properties, and rows describing particular clients.

If, at this stage, we chose the names of the columns from denominators that are well-known in our natural language NL (e.g., name, birthday …) then we can exploit the NL a-priori knowledge in order to intuitively interpret the tables. This was already recognized in the year 1977 by John and Diane Smith [SS77]: “*Since databases are usually designed to model the real world as we understand it, we can safely require that all object names in a relation definition be natural language nouns. These nouns then provide the bridge between our intuitive understanding of the real world and its intended reflection in the relation definition. If natural language nouns are not used, any discussion of the meaningfulness of a relation definition seems moot*.” Formally, however, these H^1 level concepts become related to the respective Universe of Discourse only, if the respective denominators („client“, „name”, etc.) and their a priori semantics are added to the concept space. This is often achieved by use of a data dictionary or by establishing an ontology.

Another question that sometimes gives our students headaches is: “*What is a **balance sheet** from a modeling perspective?*” Of course, our students first would ask us to specify more precisely, what we mean by "balance sheet":

(1) The usual components (concepts) of a balance sheet and their interrelationships as taught in a lecture on business administration, i.e. something that could be considered a metamodel, hierarchy level H^2?
(2) The balancing scheme of a company C, on the basis of which balance sheets for C can be drawn up at any reporting date, i.e. something that could be regarded as an extension of (1) and thus as (the representation of) a model, hierarchy level H^1?
(3) The balance sheet of enterprise C as at 31.12.2019, i.e. something that could be regarded as an extension of (2) and therefore as (the representation of) a model, hierarchy level H^0?

So far so good. But now we ask: Is a balancing scheme according to (2) a conceptual model? Again, this question can be answered with the help of the characteristics: Yes, it is a conceptual model, if the underlying metamodel associates with its elements the concept space of business administration with

concepts like "assets", "liabilities", "cash on hand", equity capital, borrowed capital etc. and their relationships.

A somewhat easier to answer question than the previous one concerns the **Petri nets** already mentioned in the introduction: are they conceptual models? Again, for answering the question we have to specify more precisely, what we are concretely referring to by the term “Petri net”. So let’s restrict ourselves to the classical “marked Petri Net”[11]. Given the definition in the footnote, we are dealing here with a purely formal structure with which no semantics are associated. However, if we associate, as Petri suggested, a "net interpretation", i.e. a concept space, with (the elements of) the marked Petri Net it becomes a conceptual model. This, by the way, on hierarchy level H^n, since a change of marking leads to a new extension. On H^{n+1}, one could, as an example, consider (P,T,I,O) together with the set of all possible mappings M as the intension of the H^n model.

Comparable considerations can also be made about **circuit diagrams** in electrical engineering. It should be clear that such a diagram is not a pure drawing or formal graph structure but the graphical representation of a planned circuit or the description of a realized circuit. In the German term "Schaltplan" (literally translated as "circuit plan") this model character is clearly expressed. The diagram for a concrete circuit is located on hierarchy level H^0, but of course H^1 plans are also common as intensions, namely when they are generic, so that several concrete extensions can be derived from them (see Figure 1). But are circuit diagrams also conceptual models? We can answer this question clearly with yes: The metamodel uses concepts exclusively from the concept space of electrical engineering (power source, resistor, line, switch, lamp, etc.), the (graph-)grammatical composition rules for the symbols of the model representation language correspond to the physical rules of concept space’s universe of discourse. Thus, the model in Figure 1 represents a conceptual model for circuits in which a lamp and a switch are connected in series to a Battery as a power source.

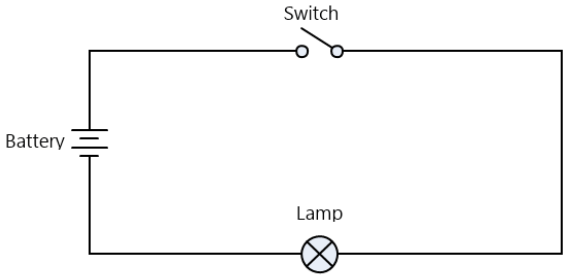


Figure 1: Simple circuit diagram

Another example concerns a **temple stele** (Figure 2) that is over 3000 years old, about which historians say the following: Basically, this is the "material manifestation" (a representation) of the contexts of a religious imagination. The stele represents a religious concept that was new at the time, namely a God who hears and answers to prayers (see the ears on the right side of the picture), with the king acting as "mediator". In addition, social conditions are described: In the group of adorants the stele founder comes first, then his wife, then their children, ranked by age (importance). Clearly, the stele represents a model of abstract and concrete originals (ch 1). It’s concern and usage is communicating the new religious concept to viewers (ch 2) with the purpose (ch 3) that these understand the concept. The focus (ch 5) is on the idea of a listening God, the representation is graphical with some symbols referencing concepts. We interpret this stele as the representation of a hierarchy H^1 level model, as at least God and king may have various extensions.

[11] A *Marked Petri Net* is a quintuple (P,T,I,O,M), where (P,T,I,O) is a bipartite graph with disjoint node sets P and T and two relations $I,O \subseteq P \times T$ such that $0 < |P \cup T| < \infty$ and $(P \cup T, I \cup O)$ is a connected graph; $M: P \rightarrow N_0$ is a mapping called *marking* (N_0 denoting the set of natural numbers including 0).

If this model is a conceptual one, cannot be said with a hundred per cent certainty. For there is no explicitly assigned concept space with corresponding concept relations handed down with. However, it is not impossible that artist and viewers were aware of such a concept space at the time.

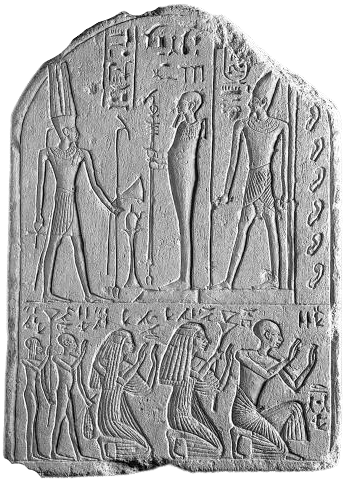


Figure 2: Stele of Seth-er-neheh, Roemer- und Pelizaeus-Museum, Hildesheim

For our last example, think of a musical score. This is a linguistic representation of the (complex) model created by a composer using a common musical language. Members of the musical community that have agreed on the meaning of terms or patterns and their combination of the musical language can infer from the score elements and their sequence to the transported concepts. For instance, think of "The Art of the Fugue" by Johann Sebastian Bach. In this case, the score represents a conceptual model. This model in turn may have many different extensions (created, e.g. through performances).

To sum up: the existence of a concept space is a precondition for models being conceptual ones; the degree to which a component of a conceptual model has UoD related, interpretable semantics depends on its associated vocabulary elements. For comparison, consider the difference between WEB 1.0 and WEB 3.0 („Semantic Web" [W3C]): In WEB 1.0 we can interpret website content based on the natural language terms used exploiting NL's a priori knowledge. WEB 3.0 pages are intended to provide a vocabulary defining the semantics of the page content; consequently WEB 3.0 pages are representations of conceptual models (mainly on hierarchy level H^0), the concept space being defined, e.g., exploiting schema.org [Sch19].

5 The Triptych: Dimensions of Conceptual Modeling

Our considerations as presented so far have inspired us to create a paradigm for conceptual modeling, namely the triptych [12]: For with this paradigm the transition from the linguistic description of phenomena to modeling and then to conceptual modeling can be described vividly through the successive opening of wings. In this section, we first explain the paradigm and then go into detail about the three dimensions that we attribute to conceptual modeling with this paradigm.

[12] A triptych is a piece of art made of three (panel) paintings connected to each other in a way that allows the two outer ones to fold in towards the larger central one (see https://dictionary.cambridge.org). I.e., when folded, the inner panel is not visible.

5.1 The Paradigm

The Closed Triptych: The intuitive perspective

Let's start with the closed triptych as depicted in Figure 4[13] : we see the backs of the two outer wings, which in this state cover the middle tableau. Let's associate this situation with the everyday situation of dealing with information without explicit conceptualization and modeling**:** humans reason on the basis of their observations on the perceivable world due to their senses, feelings and beliefs. They build their mental worlds based on their perceptions which typically differ. They live in their social worlds with their agreements. On the other side, humans use a variety of languages as an instrument for narrative representations. The "*enabling language tableau*" on the right hand side shows us, that we can use very different languages. The "*sensing, mental and social tableau*" on the left hand side symbolizes the diversity of aspects and things that can be grasped and communicated through language: (i) observations, (ii) beliefs, perspectives, trust, and cognition, and (iii) agreements.

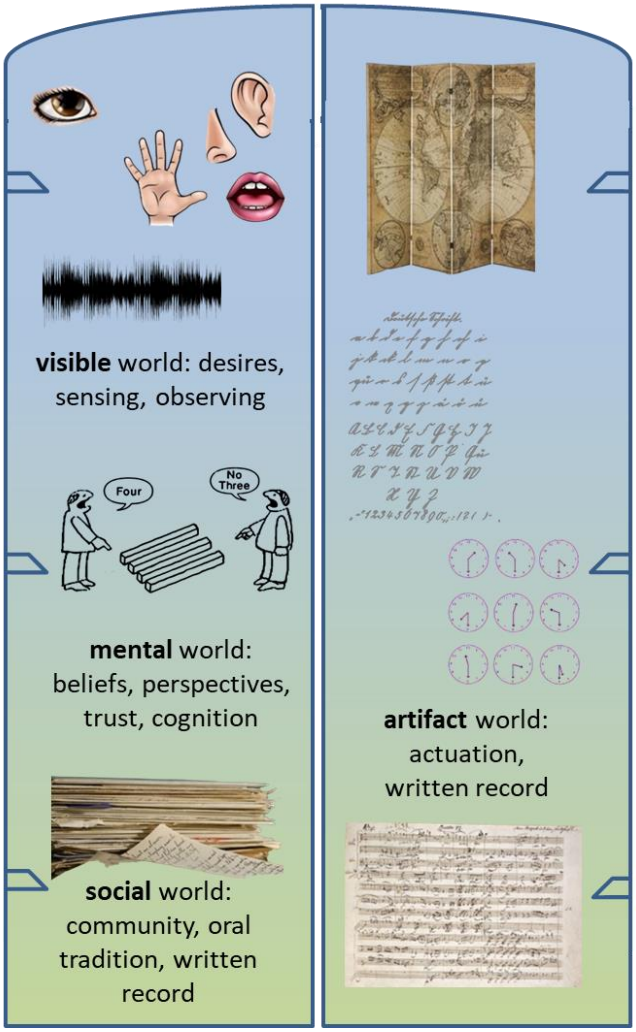


Figure 3: The two tableaus of the closed triptych: Languages (right) enabling the narrative representation of observations, mental reasoning and social agreement (left)

[13] We are well aware of the fact that we cannot match the artistic skills of the painters of real triptychs.

The Triptych with it's right wing opened: The model perspective

When we open the right wing, we see its front side and at the same time the right half of the previously hidden middle tableau as is depicted in Figure 4. This opens the way to modeling: on the now visible part of the middle tableau we see models of different levels of abstraction (meta^n-models and their extensions down to the (lowest) meta^0-level, the instance level). Their origins are the elements on the backside of the closed left (sensing, mental and social) tableau. The message of the right tableau remains the same, except that we are now dealing not only with natural languages and traditional symbolic languages but also with modeling languages or model representation languages. The choice of language is a matter of preferences, education, and practices within the community of practice. Usually, ortho-normalized languages are used for this purpose, such as an entity-relationship language based on a common language foundation.

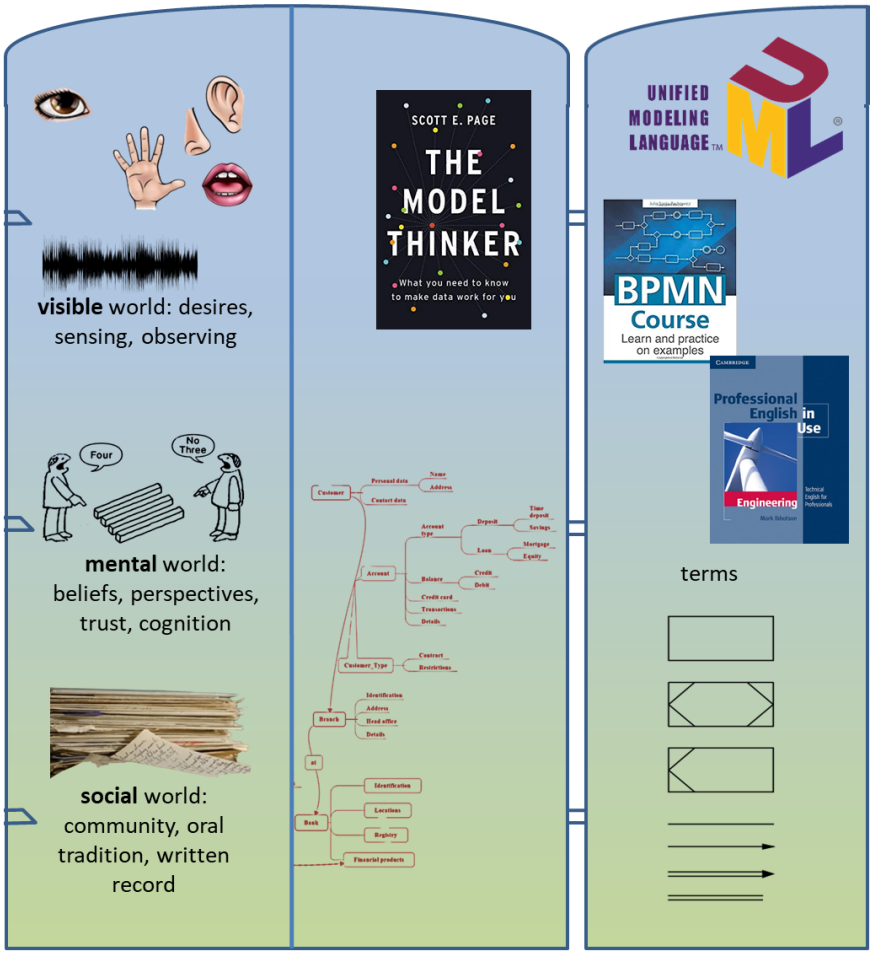


Figure 4: The triptych with its right wing opened and left wing closed

The Triptych with both wings opened: The conceptual model perspective

Opening the left wing makes the world of conceptual modeling shine in all its beauty, as the left tableau now shows us a concept space whose elements are assigned to the models on the now fully opened middle tableau (see Figure 5). The concept space brings order and structure to the world of observations, beliefs, agreements, etc., which we know are located on the back of the left wing. It may be organized by ontologies, thesauri or other kind of encyclopedias supporting conceptualization. We call this tableau the "encyclopedic tableau". It allows us to define the semantics and pragmatics of conceptual models and to relate the models to the human world.

Within this setting, the linguistic tableau supports conceptualization based on terms. We note that, apart from conceptual modeling, modeling does not need the encyclopedic tableau.

The triptych paradigm illustrates that conceptual modeling has three essential dimensions:

- The linguistic dimension: Conceptual modeling is made possible by a language that is generally accepted in a community of practice and that is semantically based on the perception and understanding of the members of the community of practice;
- The „encyclopedic" dimension: Conceptual models codify notions from the "user's encyclopedia" and express those through linguistic terms.
- The model dimension: Conceptual modeling connects the two other dimensions.

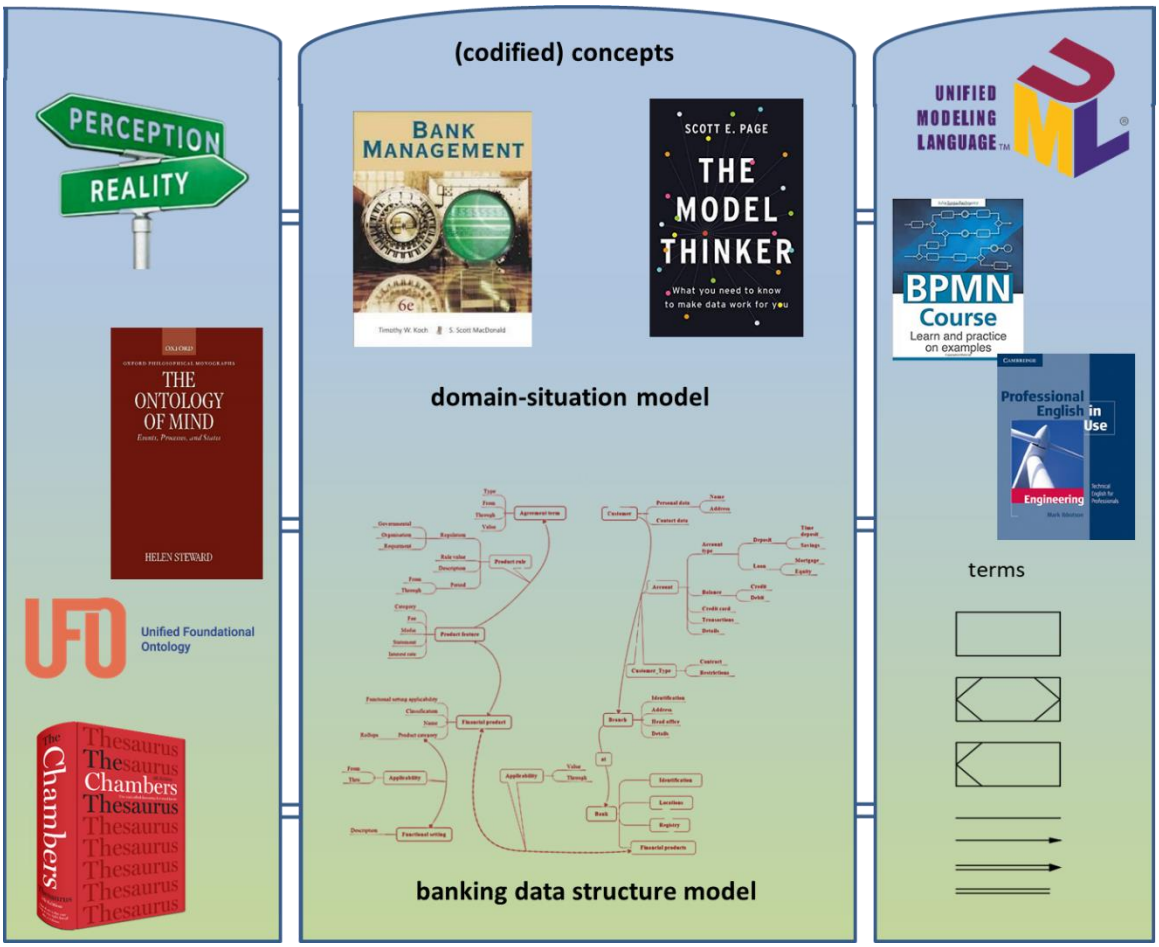


Figure 5: The open triptych
Left: The encyclopedic dimension for grounding models and their representation languages in concepts
Center: The conceptual model dimension
Right: The language dimension

In other words: we base our consideration of conceptual models on a separation of (i) language, (ii) knowledge, personal perception, and (iii) modeling as a separate activity. This separation allows us to distinguish between terms from certain languages and notions used for expressing perceptions or knowledge in the encyclopedic tableau and for enhancing models by concepts. It is thus the key to our distinction between models and conceptual models.

In the following subsections 5.2-5.4, we discuss the three dimensions in detail.

5.2 The Linguistic Dimension: The Term Space

All forms of communication take place by means of terms[14], which are exchanged via a carrier medium (apart from metaphysical phenomena). Terms can be images or image sequences, sounds or tone sequences, texts or elements of a formal language, etc. and of course mixed forms. Term sets often consist of a set of basic forms (literals) from which more complex terms can be constructed based on grown or defined composition rules, i.e. a grammar. Think, for example, of natural languages, in which words, phrases, sentences and texts are formed from the letters of the respective alphabet and some special characters. Or think of the score of a classical symphony: it consists of notes and special characters arranged along staves. The same applies to any formal language, such as the characters used in graph theory or UML.

What all languages have in common is that they can only be properly used as a means of communication if

- the communication partners know the grammar, i.e. the literals and the composition rules: a person who can't read notes can't do anything with a score except perhaps admire it because it is calligraphically appealing;
- the terms used for content description have a relative similar meaning for all partners;
- the information content transported by a term is conscious and intended by the sender and can be accessed by the receiver [vF72, FGH92].

Communication thus requires that a community willing to communicate explicitly or implicitly agrees on the literals and character set rules used as well as on the assignment of meaning to terms. For example, with the first language acquisition of our mother tongue we implicitly accept it as a means of communication and successively learn the available terms, how they are composed and what meaning is usually attributed to them. If such a means of communication comprises definitional elements, it can be used to create new language elements (e.g. new literals, new rules) and to define or propose their meaning, so that an agreement process can take place in the community: This corresponds to Gruber's original definition of ontology development ("shared conceptualization"). But we do not need full agreement.

If we now consider the model characteristics discussed in section 2, we can conclude that the essence of communication is the exchange of models based on terms. If there are rules about the composition and permissibility of the terms used, and if the represented models are associated with elements in the encyclopedic tableau, we speak of a controlled vocabulary.

5.3 The Encyclopedic Dimension: The Notion Space

People form a certain consolidated understanding of the world on the basis of their own cognition. Cognitive Scientists speak of a '*cognitive structure*'[15] that is created by '*cognitive processes*' [Kol07, vF03]: observation and perception, and activities of thinking like comparison, reflection, idealization, context expansion, abstraction, and separation. Consequently, the main ingredients of a person's cognitive structure are ideas[16] that are usually strongly interlinked.

---

[14] As announced at the beginning of section 2, we only use the word term instead of "sign" to avoid misunderstandings.-

[15] Note that the term "cognitive structure" is used in the literature with different meanings. We use it here to denote the outcome of cognitive processes but not the structure of the processes. [Kol07] calls this interpretation "conceptual structure".

[16] We use 'idea' here in the sense of 'conception', which best relates to the German word 'Vorstellung' [Bol37, Bre74, Twa94], respectively to 'mental concepts' as used in [Kol07].-

The first level of the encyclopedic dimension is thus the cognitive structure of a person. In order to communicate about it, ideas and their connections must be represented and conveyed by terms of any language. A person who perceives such terms then interprets them according to her/his personal cognitive structure. This makes it clear that with this (traditional) form of communication, a complete agreement is not possible and cannot be proven.

However, the situation can be improved by externalizing the encyclopedic dimension and formalizing it in the form of explicit thesauri, lexicons, or ontologies. Explicating the encyclopedic dimension corresponds to the opening of the left wing of our triptych. For us, therefore, this externalization is the moment when ideas become concepts. I.e. we can now sharpen our understanding of "concept" and "notion" from section 2 as follows:

1. A *concept* (in the encyclopedic dimension) is a mental construct
   - that is formed by combining characteristics of general or abstract ideas gained by cognitive processes (see section 2) and
   - that is externalized and explicated in an encyclopedic structure.[17]
2. Consequently, also a *notion* (being a general inclusive concept) is externalized and explicated in an encyclopedic structure.

If communication partners agree on the common use of such encyclopedic structures, for example a shared [Gru93] ontology, the probability of communication free of misunderstandings increases. However, it is of course still not possible to prove that the mutual understanding is identical.

5.4 The Conceptual Model Dimension: The Link between Term and Notion Spaces

Usually, a concept space is specific to a certain area of application and is based on an understanding of the perceptions of things and coherences in that area. The utilization, exploration and application of concepts depend on the user and her/his community of practice (e.g. users' education profile), usage and context.

Based on what has been said so far, we can now formulate somewhat sloppily: A conceptual model selects, uses, reconsiders, orders, and integrates parts of a notion and a term space and thus establishes a structured view on that notion space (analogously e.g. to views on databases).

Composition and structure of such a view correspond (according to CM7 and CM8, see section 3) to the given conceptual model, which in turn is an extension of a certain metamodel (more precisely: of the metamodel elements and relationships) like the ER Model. The relationships may satisfy a number of axioms and lead to poly-hierarchically ordered concept structures, typically with layers. Such poly-hierarchical structures arise in particular if the metamodel in question contains abstraction relations like generalization/specialization, aggregation/decomposition, clustering, and intension/extension, which can be instantiated on the model level.

The view, in turn, consolidates the meaning of the elements of the model and determines the linguistic meaning of terms (designators and annotations) which is an inherent but hidden aspect of the concept space.

Even though we have declared at the beginning not to present a (new) definition of "conceptual model", we can't hold back from formulating our understanding of it here: *A conceptual model is a*

[17] Note that for externalization, a concept is assigned one term or a construct of terms.

concise and purposeful consolidation of a set of concepts that are presented by means of terms in a predefined linguistic format. As such it establishes a view of a given notion space.

Let's explain all this using a very simple example, in which – for further simplification - we use a graphical grammar as known from UML for representation purposes on all levels except the lowest one, and have omitted an explicit definition of roles and multiplicities:

a) For creating a model, first of all, we have to define which modeling elements we want to use to build that model. This is done with the help of a metamodel. However, for metamodeling we need modeling elements again, so we have to create these on a Metameta level, i.e. hierarchy level H[3].
b) Figure 6 shows such very simple Metametamodel: it provides us with the possibility to define (in a metamodel) model elements that may linked by IS-A, part-of and relation connectors.
c) Figure 7 shows a metamodel that is an extension of the metametamodel of Figure 6: it introduces the modeling elements Class, Association, IS-A, Attribute and Type as well as some relationships between these elements.
d) Figure 8 shows a model that refers to the retail sector as an application area. It is an extension of the metamodel given in Figure 7. The model introduces classes Person and Organization, defines these as specializations (IS-A) of class Client which is associated with class Article in an m:n relationship. The classes have some attributes that come with type specifications for their extensions.
e) Figure 9 shows some extensions of the model given in Figure 8 on the next lower level which is usually called object or data level. Here we used a self-explanatory text-oriented grammar for representation.

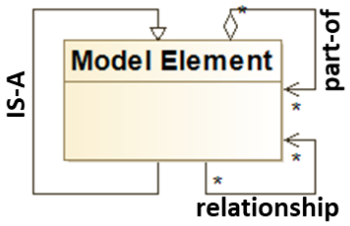


Figure 6: A simple metametamodel[18]

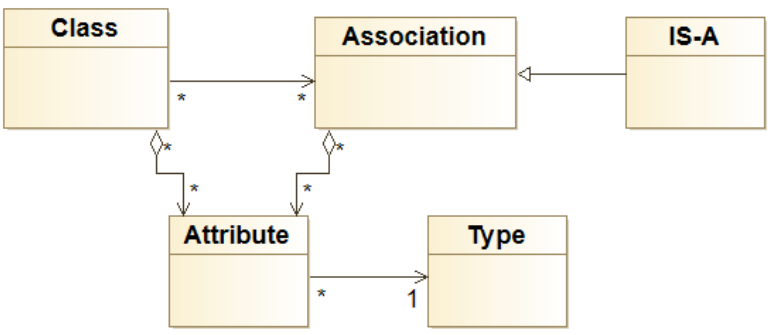


Figure 7: A metamodel created as an extension of the metametamodel given in Fig. 6

[18] Figures 6-8 were created using the Modelio tool. https://www.modelio.org/

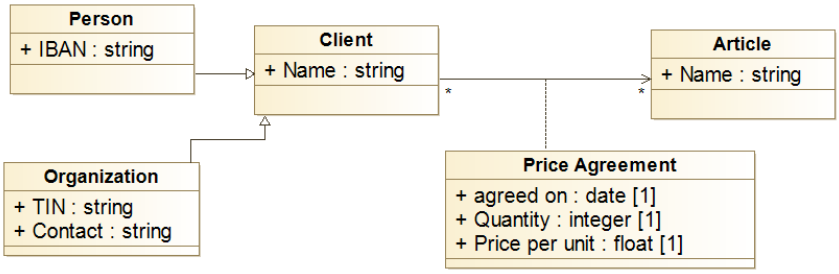


Figure 8: A Model created as an extension of the metamodel given in Fig. 7

```
Person(<Name: Frank Muller>, <IBAN: DE99 0909 9090 0909 9090
09>)

Organization(<Name: Buyers Ltd.>, <TIN: ATU99999999>,
<Contact: Frank Bourbaki>)

Article(<Name: Mouth and nose protection mask>)

Price-Agreement(<client: Frank Muller><article: Mouth and
nose protection mask>, <agreed on: 05.08.2020>, <Quantity:
10>, <Price per unit: 2,50>)

Price-Agreement(<client: Buyers Ltd.><article: Mouth and
nose protection mask>, <agreed on: 02.09.2020>, <Quantity:
1000>, <Price per unit: 0,82>)
```

Figure 9: Some extensions of the model given in Fig. 8

Please note that nothing we have presented so far in our example can be called a conceptual model. Rather, we find ourselves - metaphorically speaking - in front of the triptych with open right and closed left wing. In other words, we are dealing with models

- that are structured,
- are in intension/extension relationships, and
- have a certain intuitive meaning since we have used words from natural language and from the environment of UML.

However, the use of words, symbols and structuring mechanisms that we know from conceptual modeling does not automatically lead to the creation of conceptual models. E.g., drawing an UML class diagram is not conceptual modeling per se.

To make the (meta-)models of Figure 6-9 conceptual ones, we therefore have to associate concepts with each of their components. I.e., we need a notion space that explicates and explains the meaning of all elements (from “Model Element” down to “Price Agreement”) on all levels including the connections/relationships.

It then becomes clear, that for instance if we associate with the components of the model given in Figure 8 notions commonly used in the retail sector, this model defines a specific view on this sector in terms of its structure and the selection of what is considered relevant. We can also say that the conceptual model “codifies” the respective concepts of the application domain.

5.5 Model and Language Hierarchies

Model hierarchies are based on the duality of intension and extension [BMS86, Kan15] and thus reflect levels of abstraction. They are well known through the considerations of Information Resource Dictionary systems [ISO90] or the MetaObject Facility [OMG].

We therefore do not want to go into further details of model hierarchies in this paper. On the other hand, the model representation languages to be defined for this purpose deserve a closer look. They have to provide suitable syntactic artifacts to represent the semantic artifacts (the models).

These representation languages in turn form a hierarchy, which, however, is not isomorphic to the model hierarchy. Rather, we distinguish three levels as shown in Figure 10 [MMR17], [MMS18]:

(1) Grammar definition level (top level): contains the means of defining the language grammars. In our research, we use a specific version of the extended Backus Naur Form EBMF, compatible with the ANTLR grammar definition language [Par13].
(2) Language definition level: defines grammars for the representation languages (RL) related to the (possibly domain specific) modeling languages under consideration: meta-metamodel RLs, metamodel RLs, model RLs and instance/data RLs.
(3) Language usage level: representations of the models of all levels. For example, it is possible to use OWL 2 as a representation language this level.

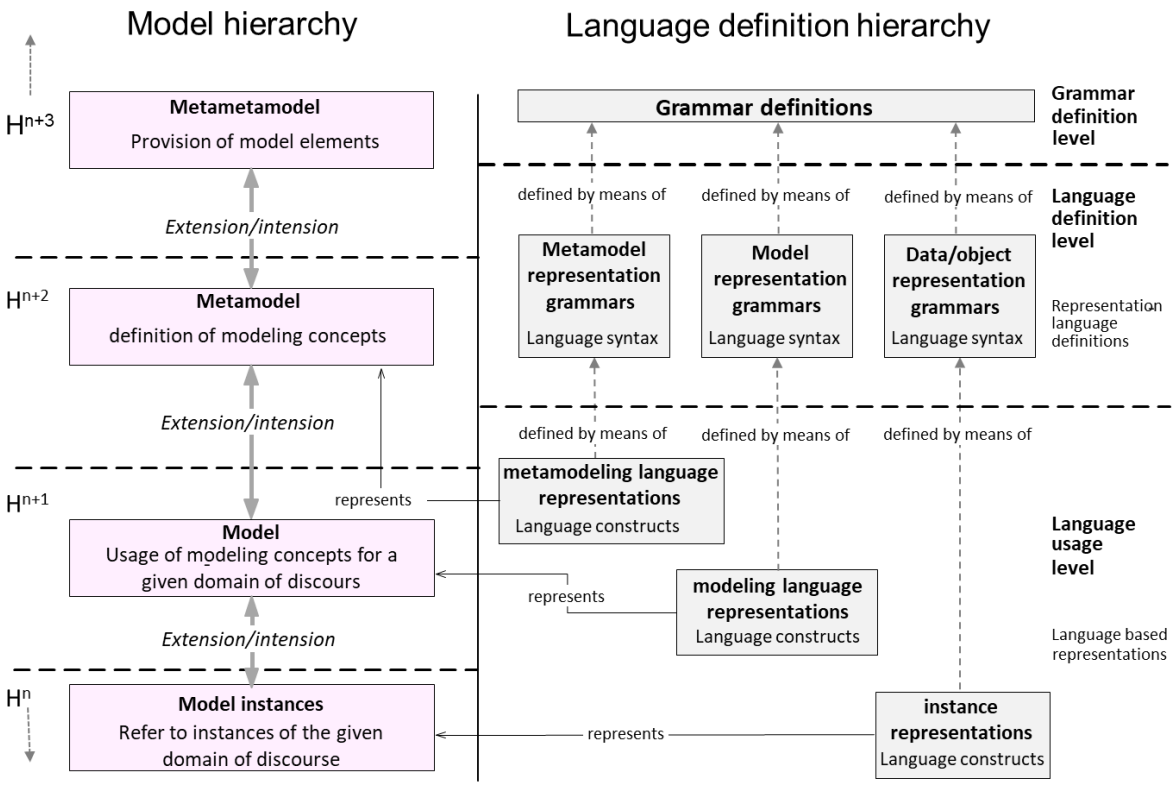


Figure 10: Language hierarchy and its connections to the model hierarchy (A^i: Abstraction Level i)

6 Conclusion and Future Research

Modeling is as old as human advanced civilisation. The bible remembers models already in the Chronicles (1, 28, 11)[19] or Torah/Genesis (2 Moses, 25, 9 or 1 Moses, 1, 27). Around the same time, Heraclitus developed the tenet of logos that consists of concepts.

Conceptual modeling is one of the kernel activities in information systems engineering. For instance, conceptual schemata are widely used since the advent of database technology and explicitly named as such with the introduction of the entity-relationship modeling language. The first model we know that might be called a conceptual one relates back to the myth of Ptah who builds the world according to his worldview and doctrine. As far as we know for sciences, simulation explicitly uses the term "conceptual model" since 1950 [RAB15]. Other disciplines use the wording "conceptual model" with quite different meaning, see for instance [SNW13].

Today, the term "conceptual model" is widely used and needs a proper systematisation. We start this systematisation with eight characteristics for a signature of conceptual model. The first six characteristics (related to origins; concern and usage; purpose and function; domain and context; focus; representation) already belong to the signature of models. These characteristics are extended for conceptual models by two additional characteristics (concept space; concept relationship). The second and the third characteristics are extended for conceptual models.

We differentiate between notions as general inclusive concepts, terms as representations referring to concepts, and concepts in the narrow sense as codified abstract ideas. In systems engineering, concepts are those in the narrow sense. The histories of notions, concepts, and terms are different. Notions have been used as reasoning instruments. Terms are bound to languages. Concepts have a history of at least 3-4000 years. The separation into notions, concepts, and terms can be used for a proper introduction of a paradigm of conceptual modeling: the triptych that consists of three wings which represent

(1) the notion or "encyclopaedic" dimension as the supporting foundation of concepts,

(2) the term or linguistic dimension as the enabler for model specification, and

(3) the concept and model dimension as the result of modeling.

The outer two wings can be used without the third one. They form then a closed triptych, i.e. a diptych. The middle part of the triptych – the model dimension – is supported by the "encyclopaedic" dimension and is enabled by the linguistic dimension.

Modeling has been systematised by abstraction levels. The separation by abstraction is typical for artificial languages. The ground H^n level represents things of interest. The H^{n+1} level is used for models; the H^{n+2} level for metamodels (i.e. essentially the structure of the modeling approach), and the H^{n+3} for metametamodels (i.e. essentially the framework of the modeling approach). This strict separation by abstraction is blurred in the linguistic dimension and almost not existing in the "encyclopaedic" dimension. Conceptual modeling thus supports defining properly structured views on the encyclopaedic dimension.

We did not plan to propose a new definition of the term "conceptual model", especially as more than 60 such definitions exist to our knowledge. The introduced signature, however, together with the Triptych paradigm provides a means and explanation of the essence of conceptual modeling.

[19] The Zwingli bible translation explicitly uses the word „Modell", whereas the Luther translation uses "Vorbild" (antetype, archetype).

We understand this as a first step towards a general theory of conceptual modeling. We encounter a good number of problems to be solved in forthcoming research. The list below is ordered according to our plans for the future; collaboration and contribution from the community are more than welcome:

(1) Modeling is based on abstraction, modularity, and other *modeling principles.* The selection of promising and useful principles is still art that compromises between model capacity, cognitive economy for the community of practice, and inferential utility. We need a proper systematisation of principles. Abstraction goes beyond structural abstraction (e.g. [SS77]) and considers advanced hierarchies beyond IsA relationships.

(2) Conceptual model characteristic 7 relates models to their *concept space*. The theory of concept spaces distinguishes concepts, notions, and terms. The corresponding spaces need a deeper exploration. Classical intension-extension Galois lattices are too strict for terms and notions.

(3) Conceptual models use *languages* as enablers. In natural languages, words or terms have their language specific semantic (or word) fields, i.e., a lexical set of words that share a common semantic property [Bri00]. These fields can be different for different languages. Conceptual modeling is not bound to a singleton language. Multi-language modeling can be based on synset approaches commonly used for WordNet.

(4) Similar to *generic models* which allow specialisation of models to more appropriate ones, concept spaces can also be based on generic concept spaces with a specialization theory that allows to adapt the concept space to a specific application, context, and community of practice.

(5) Conceptual model characteristic 8 is based on *concept relationship*. There may be various types of relationship such as one-to-one or many-to-one. Concept spaces are typically structured. The impact of this variability is an open issue.

(6) Modeling is steered by the *purpose* and *function* of the model. We know so far a good variety of model functions in different scenarios where models are used on purpose. Functions can be categorised. This categorisation can be used for categorisation of conceptual models and for stereotyping of models. These stereotypes have then a common grounding and basis that is inherited by most models of such a stereotype.

(7) We typically use a number of models of the same origins in a coherent manner. Some models are derived from other models in such *model ensemble*. Conceptual model transformation, model coexistence, and model coevolution need a theoretical underpinning.

(8) The *focus* of a conceptual model is based on the directed and concentrated attention that is steered from one side by the model's function and purpose and from the other side by the potential and capacity of the encyclopaedic support and enabling language. The impact of these governing dimensions need a proper exploration.

(9) We considered so far the four most important *concerns and usages.* There are further concerns and usages which result in different kinds of conceptual models, specific quality requirements to conceptual models, and specific variability of the model. A model is also serving a weighted overlaying combination of concerns and usages.

(10) Conceptual models do not reflect all potential *origins* of a given universe of discourse but only most likely or most typical ones. Whether the selection of such set of origins is the

most appropriate *for a given modeling target* is a difficult question. Models can also be origins of models, e.g. mental models and domain models. The plasticity and stability of a model against the selection of origins is a difficult research issue.

(11) Applications, infrastructures, origins, and user communities continuously evolve. *Evolution* of models needs a proper modernisation strategy, evolution tactics, and a realisation approach including handling of heritage (legacy) models. Models will become adaptable and self-adapting.

(12) The *context* of (conceptual) models and of (conceptual) modeling includes aspects of time, disciplines, (thought) schools, applications, experience, education, and in general of cultures. Models differ in dependence on this context. We need powerful transformation techniques that allow to become partially context-independent.

(13) Can recommendations for the development of domain-specific conceptual modeling methods be derived from all this?

This list is far from being complete. It demonstrates, however, the potential of the signature approach by systematic treatment of open issues in (conceptual) model research.

References

[BG18] T. Brand and H. Giese. *Towards software architecture runtime models for continuous adaptive monitoring*. In *Proc. MODELS 2018 Workshops,* vol. 2245 of *CEUR Workshop Proceedings*, pp. 72-77, 2018.

[Bri00] L.J. Brinton. *The structure of modern English: a linguistic introduction*. John Benjamins Publishing Company, 2000.

[BMS86] M. Brodie, J. Mylopoulos, and J.W. Schmidt (eds.). *On conceptual modeling*. Springer, Heidelberg, 1986.

[Bol37] B. Bolzano. *Wissenschaftslehre: Versuch einer ausführlichen und größtenteils neunen Darstellung der Logik*. Sulzbach: Seidel, 1837.

[Bre74] F. Brentano. *Psychologie vom empirischen Standpunkte*. Leipzig: Dunker & Humblot, 1874.

[Che76] P.P. Chen. *The entity-relationship model: Toward a unified view of data*. *ACM TODS*, vol. 1, no. 1, pp. 9-36, 1976.

[DLPS18] L. M. L. Delcambre, S. W. Liddle, O. Pastor, and V. C. Storey. *A reference framework for conceptual modeling*. In *Proc. ER 2018*, volume 11157 of *Lecture Notes in Computer Science*, pp. 27-42. Springer, 2018.

[FGH92] H. von Foerster, E. von Glasersfeld, and P. Hejl. *Einführung in den Konstruktivismus*. Piper, München, 1992.

[FR07] R. France and B. Rumpe. *Model-driven development of complex software: A research roadmap. Future of Software Engineering*, 2007, 37-54, IEEE Computer Society.

[GGM20] N. Guarino, G. Guizzardi, and J. Mylopoulos. *On philosophical foundations of conceptual models*. In *Information Modeling and Knowledge Bases XXXI*, *Frontiers in Artificial Intelligence and Applications*, forthcoming. IOS Press, 2020.

[GR19] J. Gray and B. Rumpe. *Models as the subject of research*. *Software and Systems Modeling*, 18(6): pp. 3189-3191, 2019.

[Gru93] T.R. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. International Journal Human-Computer Studies 43, pp. 907-928, 1993.

[ISO90] ISO/IEC 10027:1990(en). *Information technology — Information Resource Dictionary System (IRDS) framework*, 1990.

[Kan15] H. Kangassalo. *Definitional conceptual schemata – The core for thinking, learning, and communication*. *Keynote given at 25th EJC Conference*, Maribor, Slovenia, June, 2017.

[Kan81] I. Kant. *Kritik der reinen Vernunft*. Verlag von Johann Friedrich Hartknoch, Riga 1781/87.

[KB71] G. Klaus and M. Buhr (eds). Philosophisches Wörterbuch. Bibliographisches Institut, Leipzig 1971.

[KM20] KEA-Mod: *Kompetenzorientiertes E-Assessment für die grafische Modellierung – Projektvorstellung*. http://butler.aifb.kit.edu/MoHoL2020/KEA-Mod.pdf (accessed August 3rd, 2020)

[Kol07] M.E. Koltko-Rivera. *What Are Cognitive Structures? Are Worldviews Cognitive Structures?* Proc. 115th-Annual Convention of the American Psychological Association, San Francisco, 2007.

[KWB03] A.G. Kleppe, J.B. Warmer, W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc. (2003).

[Lat17] C. Lattmann. *Vom Dreieck zu Pyramiden - Mathematische Modellierung bei Platon zwischen Thales und Euklid*. Habilitationsschrift, CAU Kiel, 2017.

[LM78] P.C. Lockemann and H. C. Mayr. *Rechnergestützte Informationssysteme*. Springer, 1978.

[LMN93] P.C. Lockemann, G. Moerkotte, A. Neufeld, K. Radermacher, and N. Runge. *Database design with user-definable modeling concepts*. *Data Knowledge Eng*ineering, 10, pp. 229-257, 1993.

[LMW79] P. C. Lockemann, H. C. Mayr, W. H. Weil, and W. H. Wohlleber. *Data Abstractions for Database Systems*. *ACM Trans. Database Syst.,* vol.-4, No. 1, pp. 60-75, 1979.

[Mah05] B. Mahr, Was ist ein Modell. *Der Modellbegriff in Natur-und Ingenieurwissenschaften"*. TU Berlin Summer term 2005, http://pdv.cs.tu-berlin.de/gk-magsi/RV05Mahr.pdf, accessed on August 3rd, 2020.

[Mah08] B. Mahr. *Cargo. Zum Verhältnis von Bild und Modell*. In *Visuelle Modelle,* 17--40. Wilhelm Fink Verlag, München, 2008.

[Mah15] B. Mahr. *Modelle und ihre Befragbarkeit - Grundlagen einer allgemeinen Modelltheorie. Erwägen-Wissen-Ethik (EWE),* Vol. 26, Issue 3, pp. 329-342, 2015.

[MMR17] H. C. Mayr, J. Michael, S. Ranasinghe, V.A. Shekhovtsov and C. Steinberger. *Model Centered Architecture*. In: Cabot J., Gómez C., Pastor O., Sancho M., Teniente E. (eds) *Conceptual Modeling Perspectives*, pp. 85-104, Springer, 2017. https://doi.org/10.1007/978-3-319-67271-7_7

[MMS18] H. C. Mayr, J. Michael, V. A. Shekhovtsov , S. Ranasinghe and C.-Steinberger. *A Model Centered Perspective on Software-Intensive Systems.* In Proc. 9th Int. Workshop on Enterprise Modeling and Information Systems Architectures, CEUR Vol. 2097, pp. 58-64, 2018.

[MBF90] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. *Wordnet: An on-line lexical database*. *International Journal of Lexicography*, 3, pp. 235-244, 1990.

[Mil91] G.A. Miller. *The science of words*. Scientific American Library, 1991.

[MM13] J. Michael and H. C. Mayr. *Conceptual modeling for ambient assistance*. In *Proc. ER 2013*, LNCS Vol. 8217, pp. 403-413. Springer, 2013.

[MM16] J. Michael and H. C. Mayr. *The process of creating a domain specific modeling method* (extended abstract). In *Proc. {EMISA} 2016*, vol. 1701 of *CEUR* Workshop Proceedings, pp. 40-43. CEUR-WS.org, 2016.

[Mur01] G. L. Murphy. *The big book of concep*ts. MIT Press, 2001.

[Myl20] J. Mylopoulos. *Philosophical Foundations of Conceptual Modeling: What is a Conceptual Model*? ER Online Summer Seminars. https://eross2020.inf.unibz.it/wp-content/uploads/2020/07/UniBZ-John.pdf, accessed August 3rd, 2020.

[OMG] MetaObject Facility Specification. https://www.omg.org/mof/, accessed August 10th, 2020.

[Par13] T. Parr. *The Definitive ANTLR 4 Reference*. The Pragmatic Bookshelf, 2013.

[Pet62] C. A. Petri. *Fundamentals of a theory of asynchronous information flow*. In *Proc. 2nd {IFIP Congress 1962*, pp. 386-390. North-Holland, 1962.

[Pla03] R. Platon. *Platonis rempublicam recognovit brevique adnotatione critica instruxit*. Oxford 2003.

[PP19] D. Plexousakis and T. Patkos. *Modeling knowledge action and time: Action theories and their application in dynamic domains*. *NEMO Summer School 2019*, University of Vienna, Faculty of Computer Science, Vienna, 2019.

[PR18] O. Pastor and M. Ruiz. *From requirements to code: A conceptual model-based approach for automating the software production process*. *EMISA International Journal on Conceptual Modeling*, pp 274-280, 2018.

[RAB15] S. Robinson, G. Arbez, L.G. Birta, A. Tolk and G. Wagner. Conceptual modeling: Definition, purpose and benefits. Proc. of the 2015 Winter Simulation School, IEEE, pp. 2812-2826, 2015.

[Sch19] W3C *Schema.Org Community Group*. Welcome to schema.org. https://schema.org/, 2019.

[SK03] S. Sendall and W. Kozaczynski. *Model transformation: the heart and soul of model-driven software development*. IEEE Software, vol. 20, no. 5, pp. 42-45, Sept.-Oct. 2003, doi: 10.1109/MS.2003.1231150.

[SNW13] J. Shreffler-Grant, E. Nichols, C. Weinert. *The Montana State University Conceptual Model of Complementary and Alternative Medicine Health Literacy. Available from: https://www.researchgate.net/publication/253331927_The_Montana_State_University_Conceptual_Model_of_Complementary_and_Alternative_Medicine_Health_Literacy [accessed September 3, 2020].*

[SS77] J. M. Smith and D. C. P. Smith. *Database abstractions: Aggregation and generalization*. *ACM TODS* 2(2), pp. 105-133, 1977.

[Sta73] H. Stachowiak. *Allgemeine Modelltheorie*. Springer, Berlin, 1973.

[Tha18] B. Thalheim. *Conceptual model notions - a matter of controversy; conceptual modeling and its lacunas*. *EMISA International Journal on Conceptual Modeling*, pp. 9-27, 2018.

[Tha19] B. Thalheim. *Conceptual modeling foundations: The notion of a model in conceptual modeling*. In *Encyclopedia of Database Systems*. Springer US, 2019.

[TN15a] B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.

[TN15b] B. Thalheim and I. Nissen. *Ein neuer Modellbegriff*. In *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*, pp. 491-548. De Gruyter, Boston, 2015.

[Twa94] K. Twardowski. *Zur Lehre vom Inhalt und Gegenstand der Vorstellungen: Eine psychologische Untersuchung*. Wien: Hölder, 1894.

[vF72] H. von Foerster. *Perception of the future and future of perception*. *Instructional Science*, vol. 1, no. 1, pp. 31-43, 1972.

[vF91] H. von Foerster. *Zur Konstruktion der Wirklichkeit. Die erfundene Wirklichkeit. Wie wissen wir was wir zu wissen glauben*, pp. 39-60, 1991.

[vF03] H. von Foerster. *Understanding Understanding: Essays on Cybernetics and Cognition*. Springer-Verlag New York, Inc., 2003.

[W3C] W3C. *Semantic Web*. https://www.w3.org/standards/semanticweb/; accessed on August 13, 2020.

[We20] Merrian-Webster Dictionary, https://www.merriam-webster.com/dictionary/; accessed on August 3rd, 2020

[Wik17] Wikiquote. *Conceptual model*. https://en.wikiquote.org/wiki/Conceptual_model, 2017. Accessed Nov. 21, 2017.

[Wol96] G. Wolters. *Modell.* In J. Mittelstraß, editor, *Enzyklopädie Philosophie und Wissenschaftstheorie*, Vol. 2, pp. 911-913, J.B. Metzler, Mannheim,1995.

Conceptual Modelling and Humanities

Yannic Ole Kropp[1] Bernhard Thalheim[2]

Abstract: Humanities are becoming a hyping field of intensive research for computer researchers. It seems that conceptual models may be the basis for development of appropriate solutions of digitalisation problems in social sciences. At the same time, humanities and social sciences can fertilise conceptual modelling. The notion of conceptual models becomes enriched. The approaches to modelling in social sciences thus result in a deeper understanding of modelling. The main aim of this paper is to learn from social sciences for conceptual modelling and to fertilise the field of conceptual modelling.

1 The Value of Conceptual Modelling

1.1 Computer science is IT system-oriented

Computer system development is a complex process and needs abstraction, separation of concerns, approaches for handling complexity and mature support for communication within development teams. Models are one of the main artefacts for abstraction and complexity reduction. Computer science uses more than 50 different kinds of modelling languages and modelling approaches. Models have thus been a means for system construction for a long time. Models are widely used as an universal instrument whenever humans are involved and an understanding of computer properties is essential. They are enhanced by commonly accepted concepts and thus become conceptual models. The main deployment scenario for models and conceptual models is still system construction (with description, prescription, and coding sub-scenarios) although other scenarios became popular, e.g. documentation, communication, negotiation, conceptualisation, and learning.

1.2 Learning from Digital Hunanities

Digital humanities is becoming a hyping buzzword nowadays due to digitalisation and due to over-applying computer technology. We have been engaged in a number of projects, e.g. [1, 2, 4, 6, 9]. We step back now and reconsider the challenges to conceptual modelling in

[1] Department of Computer Science, Christian-Albrechts University of Kiel, 24098 Kiel, Germany, yk@is.informatik.uni-kiel.de
[2] Department of Computer Science, Christian-Albrechts University of Kiel, 24098 Kiel, Germany, thalheim@is.informatik.uni-kiel.de

these projects and generalize the experience we have gained in these projects. Let us first present a number of observations:

Observation (1): *The concept spaces used in social sciences underpins the conceptual model. Conceptions are systems of concepts. The concept space is typically structured complex and is used with multiple viewpoints.*

Observation (2): *Conceptualisation has to be co-considered at various abstraction levels at the same time, e.g. at the micro-, meso-, and macro-level.*

Observation (3): *The mould[3] (and methodology) determines model handling and the utilisation scenarios in which a model functions by playing roles. Models incorporate their function.*

Observation (4): *The model consists of a surface (or normal) sub-model and of deep (implicit, supplanted) sub-models which represent the disciplinary assumptions, the background, and the context. The deep models are the intrinsic components of the model. Conceptualisation might be four-dimensional: sign, social embedding, context, and meaning spaces.*

Observation (5): *Models benefit and suffer from the art of omission. Social and cultural embeddings are considered to be obvious and are thus omitted.*

Observation (6): *Models can be materialised and thus might have a material obstinacy due to the chosen material.*

Observation (7): *Conceptual models have to carry at the same time a manifold of understandings and a manifold of domain-situation models.*

1.3 The storyline

These observations and lessons are useful for conceptual modelling in our area. They are mostly not explicitly observed in computer science. They are however implicitly used. Think, for instance, on conceptual database models. We often use a conceptual schema that describes the structure of the entire database system and use additionally a number of conceptual views that describe the viewpoints of users. Therefore, we explain now how conceptual modelling can learn from successful approaches in social sciences. The learning process will enhance the added value of conceptual modelling.

2 Learning from Humanities for Conceptual Modelling

According to [5, 10, 13] we define the model notion as follows:

[3] The mould is a hollow form or matrix or simply frame for giving things (such as models) a particular shape. In production, moulds are used as a shaped cavity for forming fluid or plastic things.

> "*A* **model** *is a well-formed, adequate, and dependable instrument that represents origins and that functions in utilisation scenarios.*"
>
> *"Its criteria of well-formedness, adequacy, and dependability must be commonly accepted by its community of practice (CoP) within some context and correspond to the functions that a model fulfills in utilisation scenarios."*

Well-formedness is often considered as a specific modelling language requirement. The criteria for adequacy are analogy (as a generalisation of the mapping property that forms a tight kind of analogy), being focused (as a generalisation of truncation or abstraction), and satisfying the purpose (as a generalisation of classical pragmatics properties).
The model has another constituents that are often taken for granted. The model is based on a background, represents origins, is accepted by a community of practice, and follows the accepted context. The model thus becomes *dependable*, i.e. it is justified or viable and has a sufficient quality. *Justification* includes empirical corroboration, rational coherence, falsifiability (in our area often treated as validation or verification), and relative stability. The instrument is *sufficient* by its *quality* characterisation for internal quality, external quality and quality in use. Sufficiency is typically combined with some assurance evaluation (tolerance, modality, confidence, and restrictions).

2.1 The notion of conceptual model

A notion of conceptual model might be a slim, light, or concise one depending on the level of detail we need in model utilisation. We will use in the sequel one notion, i.e. the concise notion and refer for slim and light versions to [12, 14].

Concise version:
Conceptual Model $\sqsupseteq$ **(Model** $\bigoplus$ **Concept(ion)s)** $\bowtie$ **Enabler** [7]:

> A conceptual model is a model that is enhanced by concept(ion)s from a concept(ion) space, is formulated in a language that allows well-structured formulations, is based on mental/perception/situation models with their embedded concept(ion)s, and is oriented on a mould and on deep models that are commonly accepted.

The mould and the deep models form the matrix of a model [11]. We notice that a conceptual model typically consists of a model suite in social sciences. Each of the models in a model suite reflects some viewpoint or aspect.

2.2 The added value of conceptual modelling

Models do not have to be conceptual models. Conceptual models do not have to be based on an ontology. The main purpose of conception as a system of concept or of a concept(ion) space is the integration of interpretation pattern that ease the communication, understanding, delivery of a model in dependence on the model functions. The concept(ion) space, the mould of model utilisation, and the explicit knowledge of the social determination provide a means for the correct and sufficiently precise interpretation of the model elements.

2.3 The four dimensions of conceptual modelling

The consideration of the strategic, tactical, and operational sides of modelling and of conceptual modelling drives us to consider the four dimensions in Figure 1. These dimensions

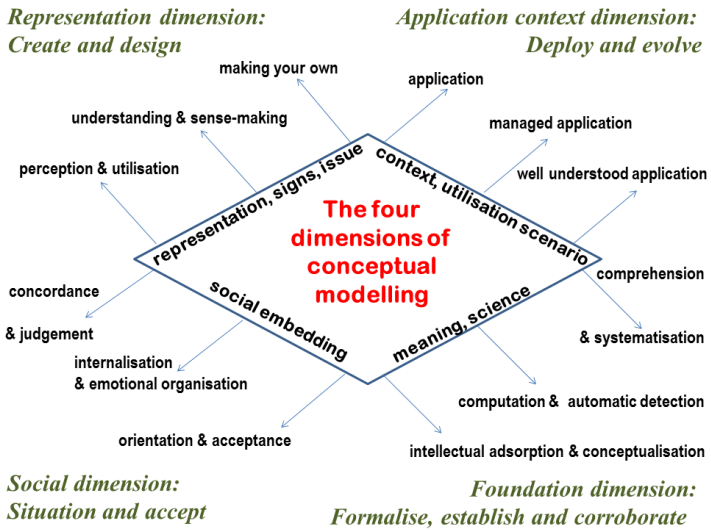


Fig. 1: The representation, application context, foundation, and social dimension of conceptual modelling

cover the application areas in [15] and especially those in humanities. Information systems typically consider the representation dimension and only one of the branches of the foundation dimension. Computer engineering especially considers the application context dimension.

Prescriptive conceptual models that are used as the blueprint for system realisation also consider this dimension. The social embedding is typical for social sciences. The foundation dimension has additional aspects in social sciences since corroboration, comprehension and systematisation are far more complex. Conceptualisation is based on complex concept and conception spaces.

2.4 Handling forgetful mappings to IT and DBMS technology

In is often claimed that conceptual database or data models are mainly descriptive ones. Description is, however, only one of the functions that a conceptual data model has in a system development scenario. Other typical scenarios are documentation, prescription, communication, negotiation, and explanation. These scenarios are also observed for humanities.
In system construction we transform the conceptual data model to corresponding realisation models. This transformation also changes the semantics from rich semantics of conceptual models to lexical semantics which is based on the lexical interpretation of the words used in realisation models according to the meaning in the given application area. It is thus forgetful. The reestablishment of the conceptualisation must thus be handled by a reference to the conceptual model what also means to use a tight bundling of all models in the case of system maintenance (e.g. evolution and migration) and integration. The social dimension and the foundation dimension get also lost during transformation.

2.5 Sophisticated conceptual models are model suites

Based on the observations, we should consider a conceptual model as a model suite, i.e. a coherent collection of explicitly associated models. The associations are explicitly stated, enhanced by explicit maintenance schemata, and supported by tracers for the establishment of coherence [8]. Each model in the model suite has its orientation and its functions in utilisation scenarios. The association schema among the models allows to consider the model suite as a complex but holistic model.
Model suites in most sciences and engineering incorporate some conceptual models. This situation is not different for social sciences. For instance, the CRC 1266 [1, 2] uses as a complex model of transformation a model suite consisting of models for socio-economic formation (cluster B-E), for socio-environmental components of change (cluster F), and for natural science investigation (cluster G). The interplay of these models allows to suppose hypotheses and to draw conclusions. Most models are already conceptual ones. They use, however, different conception spaces. The association among these models is handled by interlinkage groups within the CRC.

2.6 Models as mediating instruments instead of middle-range theories

Middle-range theories [2] are essentially model suites. They are used for an integrating consideration of quantitative sources and theory conceptions. Quantitative sources are used for derivation of quantitative concepts. The theory offer underpins these concepts. Qualitative theory-oriented research uses theoretical concept(ion)s. These concepts are supported by supporting sources which are often generated before and might use the current

quantitative sources. A theory integrates these concepts. We use typically several theories, e.g. for plausibility check, for investigation, explanation, knowledge experience propagation, and discovery scenarios. In a proxy-based research we start with proxy sources that might be underpinned by proxy concepts. This research results in a theory request that can be satisfied by a theory offer.
This approach often results in a gap between qualitative and quantitative research. Models can be used to render the theory offer. At the same time, models may also render a qualitative theory. The rendering procedures are typically different. A model suite can now be constructed by models for theoretical concepts from one side and by models for quantitative concepts from the other side. In this case, we use models for the quantitative theory offers and for the qualitative theories. This approach is depicted in Figure 2.

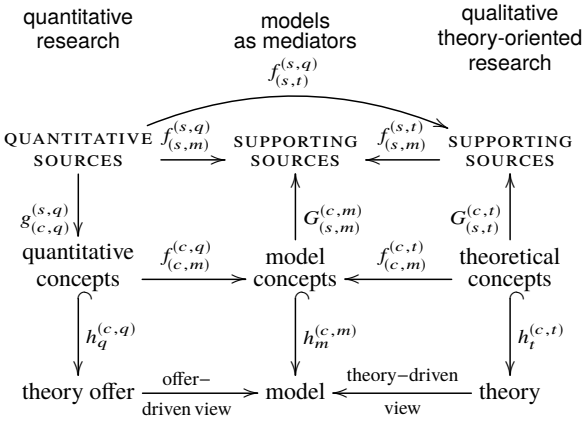


Fig. 2: Models as integrating and mediating instrument for conceptualisation, investigation, explanation, knowledge experience propagation, and discovery

This approach has already used for the investigation in the CRC 1266 [2]. In a similar form we can consider now conceptual models for other application cases.

3 Concluding: Conceptual Modelling Inspired by Humanities

Conceptual modelling is a widely used practice in many science and engineering disciplines. The current practice used for database conceptualisation can be enhanced by a number of insights that we observed in social science research.

- The concept(ion) space is often far more complex structured than finally represented and used for a singleton conceptual model. We should consider conceptual models that orient on different aspects and different levels.
- The context dimension should not be neglected for conceptual models.

- The social dimension and the foundation dimension are equally important as the representation dimension.
- Models and especially conceptual models consist of a number of models and thus form a model suite.

We got now additionally a number of special necessities for conceptual modelling without which conceptual models are of low quality, not justified, and also not adequate.

Deep models: Models consist of normal sub-models and deep sub-models. The first ones are given in an extrinsic and explicit form. The later ones are often concealed.

Model mould: The second element of the matrix of modelling is the mould. We know a number of canonic approaches that guide the modelling process, the modelling outcome, and the capacity of the finally developed model.

Concept-biased modelling: Conceptual models are typically deeply biased by the concepts in a given domain. Concepts such as "village", "settlement" and "center" are essentially representing the same understanding but are used in very different contexts. The same applies to database models, e.g. the concepts of "Person" or "Address" depend on geographic, law etc. assumptions.

Functions of models as the guiding principle: The utilisation scenarios determine the functions that a model has in such scenarios. The model is an instrument in these scenarios. Whether it is a proper and fit-to-use instrument depends on the function the model has (and thus on the purpose and the goal).

Bibliography

[1] CRC 1266. Scales of transformation - Human-environmental interaction in prehistoric and archaic societies. Collaborative Research Centre. http://www.sfb1266.uni-kiel.de/en/, accessed May 13, 2018.

[2] V.P.J. Arponen, S. Grimm, L. Käppel, K. Ott, B. Thalheim, Y. Kropp, K. Kittig, J. Brinkmann, and A. Ribeiro. Between natural and human sciences: On the role and character of theory in socio-environmental archaeology. *The Holocene*, 29(10, Special Issue "Scales of Transformation: Human-Environmental Interaction in Prehistoric and Archaic Societies"):tba, October 2019.

[3] A. Dahanayake and B. Thalheim. Development of conceptual models and the knowledge background provided by the rigor cycle in design science. In *Models: Concepts, Theory, Logic, Reasoning, and Semantics*, Tributes, pages 3–28. College Publications, 2018.

[4] S. Deicher. The language of objects. BMBF Project KunstModell in Egyptology, https://www.bmbf.de/files/Kurztexte_SdOIII.pdf, 2018.

[5] D. Embley and B. Thalheim, editors. *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*. Springer, 2011.

[6] GSHDL. Graduate school at Kiel University Human Development in Landscapes. http://www.gshdl.uni-kiel.de/, accessed July 26, 2019.

[7] H. Jaakkola and B. Thalheim. Cultures in information systems development. In *Information Modelling and Knowledge Bases XXX*, pages 61–80. IOS Press, 2019.

[8] M. Skusa and B. Thalheim. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*, chapter Kohärente Multi-Modell-Entwicklung, pages 431–454. De Gryuter, Boston, 2015.

[9] The DigiCULT team. DigiCULT: a portal of the museum network at Schleswig-Holstein and Hamburg. http://www.digicult-verbund.de/de and http://www.museen-sh.de/, accessed July 26, 2019.

[10] B. Thalheim. The conceptual model ≡ an adequate and dependable artifact enhanced by concepts. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 241–254. IOS Press, 2014.

[11] B. Thalheim. General and specific model notions. In *Proc. ADBIS'17*, LNCS 10509, pages 13–27, Cham, 2017. Springer.

[12] B. Thalheim. Conceptual model notions - a matter of controversy; conceptual modelling and its lacunas. *EMISA International Journal on Conceptual Modeling*, February:9–27, 2018.

[13] B. Thalheim. Conceptual modeling foundations: The notion of a model in conceptual modeling. In *Encyclopedia of Database Systems*. Springer US, 2019.

[14] B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, pages 123–139. Springer, 2019.

[15] B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.

Remark

We thank the reviewers for their remarks, suggestions, and critics. The paper is based on our previous papers on models, e.g. on the generalisation of approaches used in design science research [3]. The compendium [15] presents model notions and modelling used in agriculture, archaeology, arts, biology, chemistry, computer science, economics, electrotechnics, environmental sciences, farming, geosciences, historical sciences, languages, mathematics, medicine, ocean sciences, pedagogical science, philosophy, physics, political sciences, sociology, and sports at Kiel university. It is based on a decade of Tuesday-evening-open-end discussions on models and modelling in sciences. We selected only four of our collaboration projects from humanities research and discussed some of the modelling lessons.

Acknowledgement

This research was performed in the framework of the CRC 1266[4] '*Scales of Transformation - Human-Environmental Interaction in Prehistoric and Archaic Societies*' which is funded by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation - Projektnummer 2901391021 - SFB 1266)[5]. We thank both institutions for enabling this work.

[4] http://www.sfb1266.uni-kiel.de/en/
[5] http://www.dfg.de/en/index.jsp

Proc. FoIKS 2020, LNCS 12012, 1-20

Schema Optimisation Instead Of (Local) Normalisation

Bernhard Thalheim

Christian-Albrechts University at Kiel, Department of Computer Science, D-24098 Kiel, Germany

Abstract. Classical normalisation theory has a number of lacunas although it is commonly and widely accepted and it is the basis for database theory since the 80ies. Most textbooks and monographs follow this approach since then despite the good number of open problems. Moreover, modern object-relational DBMS offer far better capabilities than the systems that have been built in the past based on the strict relational paradigm. Constraint maintenance has been oriented on transformation of structures to structures that are free of functional dependencies beside key constraints. The maintenance of coherence constraints such as two-type inclusion constraints has been neglected although this maintenance might be the most expensive one. In reality normalisation is local optimisation that exclusively considers functional dependency maintenance.
We thus need a different normalisation approach. This paper develops an approach towards optimisation of schemata and global normalisation. This approach results in a denormalisation and object-relational database schemata.

1 Normalisation - The Good, The Bad, The Ugly

Normalisation is considered to be one of the pearls of database theory. There is almost no database course that does not teach this part of a theory. The main results have been achieved during the 70ies, 80ies and early 90ies. Since then the theory is considered to be completed although new DBMS (database management systems) and new database paradigms have been developed since then. There are very few publications on object-relational structures. XML approaches have mainly be following this research paradigm.

1.1 Local Vertical Normalisation Based on Functional and Other Dependencies

Local database normalization aims at the derivation of database structures that can easily be supported by the DBMS. In the past, DBMS supported keys, domain constraints and referenced-key-based inclusion constraints (so-called foreign-key-constraint). Therefore, it was a goal to derive another equivalent schema to the given one which has a set of integrity constraints that can be supported by the DBMS used for implementation. This approach can be understood as a descriptive approach to optimisation of database structuring depending on the platform for implementation. Normalisation is typically considered to be vertical normalisation.

Proc. FoIKS 2020, LNCS 12012, 1-20

Three kinds of normalisation. Normalisation is mainly considered to be *vertical normalisation* on the basis of projection sets as mappings and join as the restoration operation. *Horizontal normalisation* [PBGG89] is based on selection and union. *Deductive normalisation* [Tha91] is based on reduction of classes according to tuple-generating constraints and extended completion using this set of tuple-generating constraints. It is the most storage effective and the best computational method for normalisation as long as the tuple-generating dependency used for decomposition is acyclic [Tha91,Tha00]. The latter two normalization methods have not yet got a proper support by the database systems vendors. A common treatment for these three kinds has not yet developed.

Normalisation theory is a relict of the 80ies. The relational database design book by [Yan86] is ne of the most comprehensive surveys on kinds of normal forms. It essentially considers almost 30 kinds. Some of them have been superseded by other, some of them did not find practical solutions or their specific kinds of applications.

Normalisation starts with the requirement of the first normal form, i.e. all attributes are atomic and do not have an inner structure. Consider, however, the *ZIP* as an attribute of an *address*. It has an inner structure which is guilty for the non-BCNF normalisation of addresses [MR98]. Meanwhile DBMS suppport user-defined data structures (UDT's). These data structures have their specific functions and predicates. The *address* example is an academic one while in practice address data are checked against public address databases. A similar observation can be made on the second normal form and others. They can be neatly supported by modern DBMS. The support goes far beyond what non-first-normal-form (NF^2) research provides.

Beside the flaws of the synthesis algorithm for normalisation discussed below we should pay attention whether the *collection step* (step 3 in normalisation: collect all FD's with the same left side into a singleton new relation type) is really appropriate. Some of the FD's may have a different flavour. Consider, for instance, an *organisation unit* which is *chaired by somebody*, has its *postal address*, has a main *financial funding*, a secretariat or *office*, and a *contact*. Why it should be represented as a singleton type? It is far better to use a star kind structure since each of the associations has its meaning and its usage in the database application. Such structures are also causing evolution problems.

Normalisation approaches typically do not consider the different meanings of the dependencies [Tha19b] but treat the set of all constraints under consideration as a set of elements which are of equal importance. Moreover, normalisation is based on classes of constraints such as functional dependencies and multivalued dependencies. As we shall see below, normalisation should however been based on sets of constraints from different classes and on the meaning of such constraint sets.

Another obstacle of normalisation theory is the assumption that relation classes are sets. SQL allows multi-sets. The normalisation theory for multi-sets has a different setting [KL18].

1.2 Local Vertical Normalisation

Local vertical normalisation plays a prominent role in normalisation theory. Research started with the introduction of FD-based normal forms (first, second, third in a good

Proc. FoIKS 2020, LNCS 12012, 1-20

variety of notions, Boyce-Codd) and led to a good body of knowledge[1] Later functional dependencies have been generalised to domain, multi-valued, and hierarchical dependencies and the generalisation of the last one to join dependencies. This research resulted in introduction of further normal forms, e.g. fourth and fifth. The sixth normal form has already been introduced with the DBMS MIMER in the mid-70ies.

Local normalisation is a good approach as long as we restrict the consideration to strictly equality-generation dependencies such as functional dependencies and singleton relational schemata with atomic attributes. In this case the so-called third normal form is achievable. The Boyce/Codd Normal Form (BCNF) is not achievable in any case. [MR98] has shown however that all known counterexamples are based on ill-defined structures. It can be shown [Tha00] that either hierarchical decomposition or refined granularity of attributes result in BCNF structures.

Reasons to normalise. [Tha00] surveys the main targets of normalisation: (1) avoiding inadequate behaviour such as anomalies, (2) elimination of unnecessary redundancy, (3) prohibiting inconsistent data, (4) stability of database schemata during application evolution, (5) optimising database performance, and (6) maintenance of abstraction levels within a schema. We refer to [Tha00] for discussion of other problems encountered for normalisation such as adequate BCNF representation, adequacy of decomposed schemata, competing normalisations of the same schema, and inadequacy of multivalued dependencies and other tuple-generating dependencies within the relational database model.

Did we achieve the six targets? The answer is no. We achieved the first target and partially achieved the second target. The third target can only be achieved if all potential constraints and their influence on consistency is handled properly. The fourth target has not yet found good research solutions. After evolution schemata suffer from mannerism and look similar to Gothic cathedrals or chaotic sandcastles. After normalisation, database performance might be good for data manipulation operations. Normalisation might result in a far worse behaviour for database querying. The advent of data warehouses is a reaction on this problem. The sixth target is not supported by current languages that force us to stay on one abstraction level.

1.3 Inclusion Constraint Maintenance After Decomposition

Literature often neglects the set of additional inclusion constraints that must be maintained after a decomposition of a class. Given a multivalued dependency $X \twoheadrightarrow Y$ for a partition X, Y, Z of the set of attribute of a relation type R. The class R^C can be decomposed into $R_1^C = \pi_{X \cup Y}(R^C)$ and $R_2^C = \pi_{X \cup Z}(R^C)$ in such a way that $R^C = R_1^C \bowtie R_2^C$ iff the multivalued dependency is valid in R^C. We note that this multivalued dependency is implied by a functional dependency $X \rightarrow Y$.

This vertical decomposition of R^C into R_1^C and R_2^C must be maintained by pairwise inclusion dependencies $\pi_X(R_1) \subseteq \pi_X(R_2)$ and $\pi_X(R_2) \subseteq \pi_X(R_1)$. In the relational

[1] We restrict the citations to the most essential ones for this paper and restrain to give a full survey of the research.

Proc. FoIKS 2020, LNCS 12012, 1-20

DBMS setting the pairwise inclusion constraint should be maintained by foreign key constraints, i.e. X should be a key in both R_1^C and R_2^C.
Question 1. Is there any good approach to inclusion constraint maintenance after decomposition?

A solution to this foreign key requirement is proposed in the RM/V2 model [Cod91] by introduction of a third relation type $R_0^C = \pi_X(R^C)$.
Observation 1. This approach results in a number of additional auxiliary relation types what limits the effect of normalisation. [2]

We need to add to the normalisation approach also an *extended dependency preservation rule* that is often neglected in the literature:
Principle 1. The decomposition based on vertical normalisation adds to the decomposed types pairwise inclusion dependencies on intersecting attributes.

1.4 Constraint Sets Instead of Sets of Constraints

The classical approach of computer science introduces syntax first. And then semantics is defined on top of syntax. As again discussed in [Tha19b], this approach is nice for computational handling and for inductive and incremental construction but completely unnatural for normal languages. Syntax, semantics, and pragmatics form a unit. The *syntax-semantics-separation* principle finds its rigid continuation in the separation of integrity constraints into classes that have some uniformity in their definition structure.

The main deficiency is the constraint acquisition problem. Since we need a treatment for sets a more sophisticated reasoning theory is required. One good candidate is visual or graphical reasoning that goes far beyond logical reasoning [DMT04a].

Star and snowflake structures used in OLAP approaches are the basis for an approach that handles structures as a complex within its structure and its semantics in the complex. With the advent of object-oriented and XML languages we learned lessons on object identification [BT99] and the co-use of set-based classes with pointers. These approaches can be considered as a starting point.

Let us extend the open problem (TIC7) (Real-life constraint sets [Tha13]):

Problem 1. Provide a reasoning facility for treatment of heterogeneous sets of constraints instead of constraints from a given constraint class. Classify 'real life' constraint sets which can be easily maintained and specified.

In [DMT04b] we realised that the classical Hilbert-type reasoning (premises allow to derive a conclusion) should be replaced by another schema: some premises which are supported by other constraints allow to derive a conclusion. This set-of-support reasoning can be based on graphical reasoning means or spreadsheet reasoning schemata.

1.5 The Storyline of the Paper

Several reasons can be observed why local normalisation may be inadequate. We shall discuss some of them in the next Section. We restrict the discussion to relational database

[2] We forbear from postulating these observations as theorems. They are rather simple and easy to check statements.

Proc. FoIKS 2020, LNCS 12012, 1-20

technology and to conceptualisation through the extended entity-relationship model [Tha00]. Some of the pitfalls of object orientation [Web95] and of XML orientation have similar causes but are outside the scope of this paper. Normalisation theory is so far exclusively built as a theory of vertical local normalisation. We might ask whether we should consider global vertical normalisation. Or at least other kinds of local normalisation as well. The main target of normalisation of optimisation of the overall database for all six targets. Instead of poly-optimisation for some of the six criteria we might use a less strict form by optimisation of some of the database types and by denormalising others.

Since poly-optimisation is typically unsolvable we develop a number of corrections to normalisation approaches and a general approach to denormalisation as a kernel for a general theory of optimisation.

2 Solutions for Classical Normalisation Approaches

2.1 Refining Synthesis Algorithms

Rigidity of classical synthesis algorithms. The third step of the classical synthesis algorithm typically groups all attributes that can be inferred from the same set of attributes by functional dependencies. This approach groups then attributes that are conceptually completely independent into one group. We may observe, however, that some separation must be maintained.

Let us consider a simple example [Tha19b] of a relational type R: given attributes $attr(R) = \{A, B, D, F, G, I\}$ and a set of functional dependencies $\Sigma_R = \{A \longrightarrow IG, D \longrightarrow FG, IAB \longrightarrow D, IF \longrightarrow AG\}$.

This FD set can be represented by the graph on the left side of Figure 1. This set can be reduced by deleting $IF \longrightarrow G$ from the graph since it is derivable through the edges representing $IF \longrightarrow A$ and $A \longrightarrow G$. Furthermore, the set ABI can be reduced since the edge representing $A \longrightarrow I$ already supports subset reduction. No other reduction can be applied to the graph. We use the calculus for graphical reasoning [DMT04c] that is complete and sound. We use dotted lines for the subset relationship among subsets of attributes and arrows for functional dependencies.

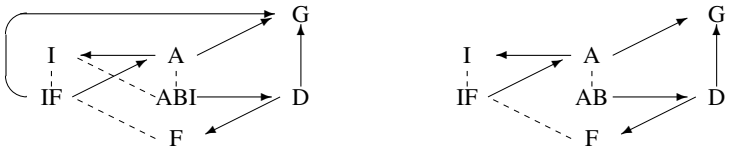


Fig. 1. The graph of the functional dependencies and the reduced cover of this set

We may directly derive a normalisation according to this graph reduction. Each constraint must be covered. We arrive with the synthesis algorithm to[3]:

[3] If we require that all inclusion dependencies are referential integrity constraints then we need 12 types for normalisation that results in a foreign-key-faithful decomposi-

$R_1 = (\{A, G, I\}\ ,\ \{A \longrightarrow GI,\ R_1[AI]\ \subseteq\supseteq\ R_2[AI]\})$,
$R_2 = (\{A, F, I\}\ ,\ \{A \longrightarrow I, FI \longrightarrow A,\ R_2[F]\ \subseteq\supseteq\ R_4[F]\})$,
$R_3 = (\{A, B, D\}\ ,\ \{AB \longrightarrow D,\ R_3[D]\ \subseteq\supseteq\ R_4[D],\ R_1[A]\ \subseteq\supseteq\ R_3[A]\})$,
$R_4 = (\{D, F, G\}\ ,\ \{D \longrightarrow FG,\ R_1[G]\ \subseteq\supseteq\ R_4[G]\})$.

The set $\{A, B\}$ is a key. We thus do not need an additional key type for the normalisation.

If we however take into account constraint maintenance and redundancy then we arrive at a smaller and better schema with the type:

$R_1' = (\{A, G\}\ ,\ \{A \longrightarrow G,\ R_1[A]\ \subseteq\supseteq\ R_2[A]\})$

due to the validity of the following derivation (reduction rule):

$$\frac{R_1[A, I]\ \subseteq\supseteq\ R_2[A, I],\ R_2 : A \to I,\ R_1 : A \to GI}{R_1 \bowtie R_2\ = R_1' \bowtie R_2,\ R_1' : A \to G}\ \ R_1' = \pi_{A,G}(R_1) \qquad .$$

This rule is based on general deduction rules and on equalities for the relational algebra:

Theorem 1 (*General deduction for closed Horn formulas*). *A sound and complete axiomatisation for closed Horn formulas* $\forall ...(\alpha \to \beta)$ *consists of*

axioms

$$\frac{}{\alpha \to \beta} \qquad \text{for all facets of substructures } \beta \preceq \alpha$$

,

augmentation rules for super-structures α^+ ***and sub-structures*** β^- $\frac{\alpha \to \beta}{\alpha^+ \to \beta^-}$
for either $\beta^- \preceq \beta$ *and* $\alpha \preceq \alpha^+$ *or as well as* $\alpha^+ = \alpha \sqcup \gamma$ *and* $\beta^- = \beta \sqcap \gamma$,
and

transitivity rules $\frac{\alpha \to \beta,\ \beta \to \gamma}{\alpha \to \gamma}$ *for all connecting pairs* ($\alpha \to \beta$,$\beta \to \gamma$).

The proof of the theorem is based on the Boolean representation of the open first-order predicate calculus and on properties of implications. The completeness uses the same arguments as the classical FD completeness proof.

An alternative proof of the reduction rule is based on algebraic dependencies [PBGG89,Tha91].

We observe that graphical synthesis would result in a better behaviour. This structure is represented in Figure 2. The hyper-network approach [PP13,Tha19a] uses nodes as Venn diagrams of subsets of the set of attributes and directed edges between the nodes. The constraint set in Figure 1 is given by the hyper-network representation in Figure 2. Compare this representation to the decomposition hypergraph for the classical synthesis algorithm. The hyper-network representation is rather simple to read. Each of the edges must be represented by some new relation type. Moreover, the set $\{A, B\}$ is a key due to the graph node closure. Otherwise we may use a combination of nodes for the graph node closure. The second minimal key is $\{I, F, B\}$ which is not a node and thus would have been added to the decomposition if we would not have represented the first one. We notice that graphical reasoning is simpler for implications than Hilbert-type calculi.

tion: $R_1[\underline{A}], R_1[\underline{I}], R_1[\underline{A}, I], R_1[\underline{A}, G], R_2[\underline{F}], R_2[A, \underline{I, F}], R_3[\underline{B}], R_3[\underline{A, B}], R_3[\underline{A, B}, D]$, $R_4[\underline{D}], R_4[\underline{G}], R_4[\underline{D}, F, G]$ where the key of each new type $R_i[X] := \pi_X[R_i]$ is underlined.

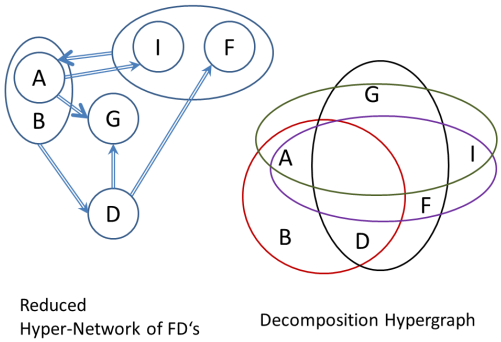


Fig. 2. Graphical normalisation depicted as hyper-network and decomposition hypergraph.

Questions one might ask for normalisation theory and their research agenda. The classical normalisation theory is based on functional and multi-valued dependencies. Normalisation synthesis algorithms are deterministic. The result depends on the order of attributes and on the order of constraints considered in the algorithm. The minimal cover is not unique for a given set of constraints (even not polynomial according to the number of attributes in the worse case). We, thus, have a good number of opportunities for a normalisation.

Question 2. Which normalisation opportunity should be the best one?

Let us consider a very simple set Σ of functional dependencies for

$R = (\{A, B, C, D, E, F, G, H\}, \Sigma)$ with

$\Sigma = \{A \to B \to C \to AD, D \to E \to F \to DG, G \to H \to G\}$.

Σ has more than 50 minimal covers. Similar examples can be given for results of normalisation according to synthesis algorithms. *Observation 2. Attribute sets which are FD-equivalent can be given in an abstract form, i.e. we consider in a set of constraints the complex* $[A, B]$ *instead of* $\{A \to B, B \to A\}$. The attribute-wise consideration might be appropriate for first normal form defintions but it complicates reasoning on constraints.

We thus represent Σ by the rather simple FD set system $\{[A, B, C] \to [D, E, F] \to [G, H]\}$. It has 18 different BCNF normalisations.

Question 3. Why we should consider so many minimal covers and normal forms?

We note that multivalued dependencies are defined in the relational database theory in a mathematical manner. They are far better expressed by entity-relationship modelling languages [Tha03] and far simpler to capture and to develop.

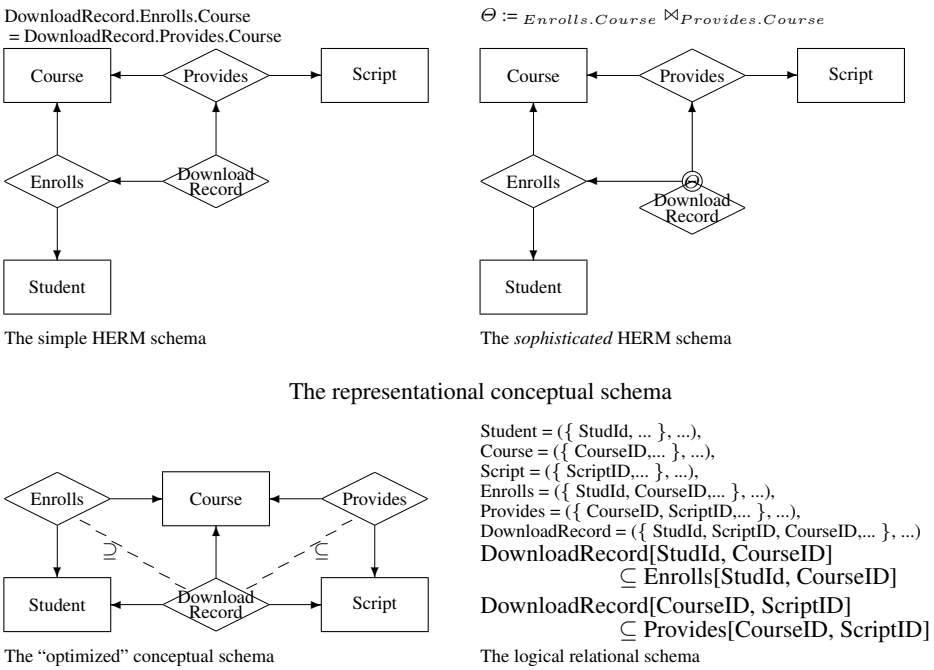
Question 4. Should we better develop a normalisation approach for entity-relationship schemata? Should we better consider a schema for normalisation instead of type-wise normalisation?

Good HERM schemata are typically the best normalisation. Folklore of ER modelling claims that the best normalisation is obtained if the main target of conceptual modelling

is the specification of an ER schema. This claim is not valid in general since extended entity-relationship modelling languages such as HERM [Tha00] are not cognitively complete. The ER approach provides however a far better solution to normalisation of relational schemata than normalisation theory, e.g. for multivalued and hierarchical dependencies. Moreover, the structure is more natural and thus better to comprehend.

A flaw of the first normalisation algorithms was corrected by a *key composition rule*, i.e. if the decomposed relational structure does not have a constructed type which contains some key of the old schema as a (sub-)structure then a new type is created for one of the minimal keys and added to this relational structure. This rule is nothing else as a decomposition for which a relationship type is added with a key that has the property that it is overwriting the key product of related decomposed types. The corresponding hyper-network has then nodes which are not incrementally layered and thus need a connecting element which can be then used as a key. The hyper-network approach also allows a generalised and less strict key composition rule.

2.2 Balancing Between Conceptualisation and Programming Adequacy



The simple HERM schema

The *sophisticated* HERM schema

The representational conceptual schema

The "optimized" conceptual schema

The logical relational schema

The conceptualisation of the relational schema through a simple but inadequate ER schema

Fig. 3. The 'Janus' schema cluster for conceptual modelling

Database design and development is often based on the three-layer architecture. This architecture requires that the user can be supported by views defined on top of the

Proc. FoIKS 2020, LNCS 12012, 1-20

conceptual schema. The conceptual schema is mapped to the logical and to the physical schemata. The last two are considered to be internal or implementation schemata. It is well known in database design [SW05,Ste00] that the conceptual schema be represented by a 'Janus' schema, i.e. one schema (*representational conceptual schema*) that is used for conceptual representation, for view formulation and for conceptual programming and another schema (*optimized conceptual schema*) that is used for implementation issues including logical and physical programming. The two schemata are equivalent to each other and are tightly associated with each other by transformation mappings. A typical example of these two schemata is given in Figure 3. The example is taken from a script server project. Students enrolled in a course may download scripts that are provided by the course.

The optimised conceptual schema can be easily mapped to a structure that supports smooth operating of the database. We can deduct from this schema the internal representation, supporting structures such as indexes (in various kinds), internal concatenation or splitting of relations, introduction of generalisations, the navigational structure for access, the management of derived structures (derived attributes, views, macro-data on top of the micro-data of the database), criteria for the selection of internal storage and computational structures, the necessity for the introduction of artificial or surrogate keys, and for clustering or separation of records. These parameters are used for tuning and physical optimisation of the database. The sophisticated HERM schema uses the Θ-join for the correct building of the relationship type that records downloads. The optimised conceptual schema is equivalent to this schema due to the equivalence of the join decomposition and the inclusion constraints [Tha00].

2.3 Accuracy of the Internal Database Structure

The internal database structure is ruled by the DBMS. The mappings from the conceptual schema to the internal schema must preserve a number of properties:

Preservation of content: The conceptual schema has been established by modelling the application domain and highlights data that are going to be stored in a database. The internal schema represents the same and only the same data that can be recorded through the conceptual schema and is based on the database modelling language of the platform assumed for implementation.

Preservation of access and modification: The access pathes and modification pathes that are implicitly or explicitly assumed for the conceptual schema are mapped to access and modification pathes that are entirely supported by the platform assumed for the implementation.

Preservation of maintenance efficiency: Integrity constraints are given in a declarative form in conceptual schemata. The efficiency of their maintenance is not considered. The (economic) value of constraints is neglected. Internal schemata must provide mechanisms for efficient integrity maintenance.

The first property is an element of any database course and well considered in most database books. The property can be treated on the basis of non-losslessness and dependency preservation. It might be enhanced by the requirement that each projection

Proc. FoIKS 2020, LNCS 12012, 1-20

(or other range structure) should be needed for the reconstruction process. The second property is mainly solved by the professional experience of database professionals. It is typically not discussed in scientific publications and is an element of database operator education. The third property is often neglected in database research. Database operators have a very simple solution for this property: they switch off all those integrity preserving processes that become a bottleneck at database production time and switch on these constraints for a short time at the maintenance phase.

Preservation of access and modification is a fuzzy criterion since we often may not assume that any access and modification can be forecasted at design time. The co-design approach to database modelling [Tha00] also takes into consideration functionality that can be envisioned. Optimisation of the internal schemata is based on profiles of functions, stored procedures and transactions (shortly processes) and on the cardinality profile of the relations. The first profile provides information on the kind of the operation, the frequency, the relations affected by the processes, the type of access (online, batch, prefetch or ad-hoc) and the changes applied to the databases. The cardinality profile of relations provides detailed information on the size of the relation (minimal, maximal and average), on the changes of the size over time and the associations among relations that must be maintained. Both profiles are compared with the modus of computation (bath, online, ad-hoc), with performance expectations (execution time, throughput, priority), with visibility of performance gaps (depending on operations frequency, organisation level of users, business process interaction) and with computation strategies for the operations (kind of operation, scheduling, auxiliary facilities, selection and storage alternatives, set sizes).

Decomposition approaches generate structures that easily support some of the constraints such as key constraints, domain constraints and key-based inclusion constraints. Typically, the maintenance complexity of such constraint sets is not taken into account. Moreover, decomposition algorithms may generate a large variety of decompositions that are semantically equivalent but pragmatically and technologically different. Typical normalisation algorithms are deterministic for a given set of functional dependencies, for an order of attributes and an order of the dependencies. Changes in the the last two orders result in different solutions of those algorithms.

2.4 Infomorphisms among Schemata

We use the notion of infomorphisms as the general foundation for schema optimisation. Infomorphisms have been used for schema modernisation in [KT11,WT13]. Let us consider two *database* ***schemata*** $\mathcal{S}_1$ and $\mathcal{S}_2$ consisting of *database types* of the form $T = (struc(T), \Sigma, \Sigma^*)$ with a structure definition, inner integrity constraints Σ defined on the type, and outer integrity constraints Σ^* that constrain the type by means of other types. Structure elements of types that are not defined by constructors are called *basic*.

Let us consider only *complete schemata*, i.e. those which types are complete relative to the outer constraints. Given furthermore, *basic domain types* $\mathcal{B}$ for the value foundation of the database. We use the abstract data type approach for basic domain types and presume for these types their value collections, their operations, and their predicates.

An *extended database schema* $\mathfrak{D} = (\mathcal{S}, \mathcal{B}, DOM)$ consists of a database schema and an assignment DOM of basic elements of its types to basic domain types.

The set of all $MOD((\mathcal{S}, \mathcal{B}, DOM))$ of all finite databases on $\mathfrak{M}$ consists of finite collections of classes for each type for which all constraints are valid, which values of objects in a class are given by DOM.

Let us now associate databases for different extended database schemata $\mathfrak{D}_1$ and $\mathfrak{D}_2$ by mappings mappings $\widehat{put}_{1,2}$ and $\widehat{put}_{2,1}$. These two mappings form an ***infomorphism*** of $MOD(\mathcal{D}_1)$ and $MOD(\mathcal{D}_2)$ if for i, j with $\{i, j\} = \{1, 2\}, i \neq j$ and for each database DB_i on $MOD(\mathcal{D}_i)$ there exists a database DB_j and on $MOD(\mathcal{D}_j)$ such that $\widehat{put}_{i,j}(DB_i) = DB_j$.

We may extend this notion also to views defined on each of the database schemata. The association among views can be based on the extract-transform-load (ETL) approach where extraction is based on a query language of the first schema, transformation is given by an infomorphism, and loading uses views which allow updates on the second database schema.

This notion is very general one. Infomorphisms are essentially transformations of one database to another one. These transformations are *information-invariant* in the sense that any database object collection can be associated with one and only one database object collection from the other extended database schema.

The infomorphism notion can be based on HERM schema operations ([Tha00], Chapter 9.2.) in the case that we consider only classes with set semantics. The Σ^* dependence among types also includes inclusion constraints. Therefore, vertical normalisation can be directly expressed in this approach. In this case, we can represent transformations as graph-grammar rules which are defined on sub-schemata. Horizontal normalisation uses separating selection σ_{α_i} predicates which define a partition of singleton classes. Deductive normal forms use for the mapping a reduction operation from one side and a chase-like completion procedure for the other mapping.

Observation 3. Vertical normalisation, horizontal normalisation, and deductive normalisation as specific variants of infomorphisms.

We conclude now that a theory of infomorphisms can subsume the classical relational normalisation theory, especially vertical normalisation. It is, moreover, better since the pairwise inclusion constraints after decomposition must be integrated into the decomposed schema. Infomorphisms can be partially supported by schema construction rules for extended entity-relationship schemata. These rules follow the graph grammar approach.

2.5 Global and Local Vertical Normalisation

?

?

?

HERM book

Proc. FoIKS 2020, LNCS 12012, 1-20

3 Denormalisation

... There are many database experts, particularly the more academic ones, who feel that any talk about denormalising a database is like a race car driving – a great way to mangle or kill yourself even if you know what your are doing. [Cel99]

3.1 State-Of-the-Art for Denormalisation

We observe two camps where the first one is well acknowledged.

No denormalisation at all! Almost all[4] textbooks and monographs in the database area require strict normalisation. Local (vertical) normalisation of a singleton database type is well reflected in most database books (e.g. [AHV95,Bis95,Leo92,Yan86]) and publications, most database courses, and in actual database practice. It is considered as one of the pearls of database research and known to almost everybody who knows database technology. The provenance and acknowledgement is based on the facility it provides: keeping as much as possible locally and globally supporting only those processes that are inherently global. Both independence concepts of databases (conceptual independence and implementation independence) are based on localisation.

[Dat11] advocates lazy normalisation based on relevant FD's, i.e. ***graceful normalisation***[5] for which not all functional dependencies that are valid in database schema are considered but only the important ones[6]. Additionally, almost valid FD's might be more important than FD's that happens to be valid. The treatment of such dependencies would be based on the introduction of artificial identifiers, i.e. a heavy object identity pollution. A far better solution is horizontal decomposition with a class for which all identities are valid and an exception class in which the few exceptions are recorded. Horizontal decomposition can be combined with union views as long as the exceptions are disjoint from the normal case.

Graceful denormalisation whenever it is really necessary: Very few papers and books advocate or consider at least denormalisation (e.g. [BFG+08,Cel95,Cel99] or the discussion in [Tha00]). The three central quality criteria for database installations are, however, performance, performance, and performance. The classical vertical local normalisation is useful as long as any casual user may query by any casual query at any time without considering performance. However, a database system contains of a (or a number of) DBMS with a number of databases on top of which

[4] We know so far only less than a handful books that do not require such.

[5] Many constraints can be omitted since integrity is also often managed through proper interfacing and exchange procedures without a chance for inconsistency as long as the data modification is exclusively based on interface or exchange view data. The development of a theory for this approach is one of the lacunas of database theory.

[6] We avoid the exponential size trap for sets of functional dependencies with this toleration of incompleteness of constraint sets. We consider only essential ones and completely or partially neglect others. This approach can be extended to a *theory of robust normalisation*.

Proc. FoIKS 2020, LNCS 12012, 1-20

a large massive of business procedures has been developed. These business procedures form the main part of the profile of the database. Casual queries are rather exceptions. The definition that is used for denormalisation is typically based on application of the natural join operator to relational types[7].
Our consulting experience and also observations on the why's for OLAP and data warehouse applications drives us to a completely different picture in many applications. The first setting of a database application is very often based on normalisation. This database becomes then step by step denormalised after the database is fully populated and operating. Already after one year of full operation, the database is partially normalised and also partially denormalised.

3.2 A Matter of Definition

Date's defs
definition is a problem

3.3 Integrity Maintenance Cost

inclusion constraints are not for free except 6NF form (or Codd's form)
not only FD is the target, IC cost
better Codd separation R-Type (key type with subtypes for dependent attributes

3.4 Denormalisation Driven By Optimisation

3.5 Denormalisation Seems to Ruin Normalisation

Strict local normalisation may be inadequate. A theory of denormalisation has not yet been proposed as far as we know. Instead, a number of heuristic rules for denormalisation are provided. These rules are based on observations for performance traps for some of the platforms and often use the 80/20% rule.

It is well known that denormalisation is bad for updates since they are harder to formulate and can jeopardise database integrity. Denormalisation may also lead to complications for query formulation. It becomes easier to formulate incorrectly a query to a request meaning that the query does not correspond to the request. Typically, tricky view creation is used for denormalised tables. Therefore, it seems that normalisation is the best way for optimisation of database behaviour.

Despite these advantages of normalisation the denormalisation is considered a method for performance improvement. It is based on precomputing derived data, on minimising the need of joins, on reducing the number of foreign keys in relations, on reducing the number of indexes and on reducing the number of relations. OLAP and data warehouse techniques rely on denormalisation. We shall, however, detect in the next sections that denormalisation can be minimised if global normalisation is applied. [LTN07] lists some key effects of thoughtful denormalisation: definite improvement in query time, a potential increase in update time or in storage space, a potential loss of data integrity due

[7] The validity of pairwise inclusion constraints is also neglected in this case.

Proc. FoIKS 2020, LNCS 12012, 1-20

to certain deletions, the necessity for program transformations for all relevant queries and the overhead needed to reorganise some tables.

Physical database design should follow logical database design. Logical database design should follow conceptual database design. We thus need the 'Janus' schema cluster for translating conceptual queries and programs to logical programs. Denormalisation is an approach to improve efficiency and performance of the database. Since we advocate co-design of structuring and functionality we develop a broader view to optimisation. Typical denormalisation techniques are

- the introduction of controlled redundancy for avoiding joins, e.g. by copying attributes to other tables,
- the introduction of nested types such as repetition groups,
- the composition or join of tables which are separated by functional or multivalued dependencies and which would not be separated if exceptions for these dependencies were not be separately stored, and
- the maintenance of lookup tables that keep referenced values of other tables without having additional attributes.

The first technique is supported by restrictions to modifications of redundant components and by flooding their values from the base table. The second technique is supported by modern object-relational systems. If nested types are used in minimal keys then surrogate key can be introduced. The third technique uses horizontal decomposition into the (rare) exceptional cases together with vertical decomposition of the exception table. The fourth technique uses enumeration domains and thus supports domain constraints.

3.6 Semiotic Units

as complex

4 Conclusion

4.1 Summarising

Normalisation is considered to be one of the pearls of database theory and technology. We mainly consider, however, local vertical normalisation instead of global normalisation or horizontal normalisation. It seems that normalisation theory is a body of knowledge that is completely settled and well understood in most of its aspects. We discuss on the basis of simple examples that this impression is not valid. Normalisation is not well understood. It needs a lot of extensions and corrections. It must also be completely revised for the modern DBMS technology. One essential revision is the flexible choice for set-based or multi-set-based semantics. This extension opens the path towards list, pointer, multi-list, etc. semantics that is supported nowadays by systems.

Normalisation is often a performance bottleneck. Repairing this bottleneck is often done on the fly. In practice, skilled consultancy uses here a hands-on, experience-backed approach. The DAMA[8] community and database forums widely discuss in

[8] www.dama.org

Proc. FoIKS 2020, LNCS 12012, 1-20

closed groups the experience some people got. We claim that most larger database applications allow coexistence of partial normalisation (in both vertical and horizontal style) and partial denormalisation.

This paper aims now to highlight the path to a coherent theoretical underpinning for this kind of coexistence. We first discussed problems of classical normalisation based on the verticality and locality approach for simple constraints such as functional and multivalued dependencies. The problems discussed can be resolved by pragmatical approaches. Some of them are discussed in the paper. We are not capable to present a full theory which would require a two-volume monograph. So, we restricted only on some parts of this theory.

Normalisation can be understood as a special kind of optimisation. Optimisation of schemata is based on some kind of equivalence. We use informorphisms as one solution for treatment of equivalence. This solution requires deep knowledge of the the given database application. It can be extended to handling of robust constraint sets what is, however, an open issue. It can also be extended to handling by basic-structure normalisation that is neatly supported by interface and exchange tolerance as long as the interfaces and the exchange means provide a support for the other optimisation (or more specifically normalisation) requirements.

4.2 Open Problems

The list of open problems is slowly shrinking and quickly expanding at the same time. We have collected open problems since MFDBS'87[9], have extended this list, and observed whether some of them have been resolved. The latest version in [Tha13] contains 22 open problems which are directly related to normalisation theory.

Normalisation theory is currently a theory for system structures in the small. Global normalisation will be theory for system structures in the large. The world is now changing to systems in the web and systems that are based on completely different performance challenges such as big data massives. *Normalisation in the world* is a really big issue for future research. It goes far beyond theories we know for distributed databases.

Revolution instead of disgraceful extension. Already research on the OO identifier and the OID pollution has been demonstrating that parts and pieces of database theory must be revised. Many assumptions taken for granted are not valid anymore and will never be valid again for challenging applications such as big data massives. Set semantics was a nice tool in the past. It is not the right one - at least for SQL applications and multi-sets in practice. we might ask why not also to use multi-list semantics. Big data requires a different FD logic.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, Reading, MA, 1995.

[9] With 21 open problems from which 13 are not yet solved.

[BFG+08] S. Buxton, L. Fryman, R. H. Güting, T. A. Halpin, J. L. Harrington, W. H. Inmon, S. Lightstone, J. Melton, T. Morgan, T. P. Nadeau, B. O'Neil, E. J. O'Neil, P. E. O'Neil, M. Schneider, G. Simsion, T. J. Teorey, and G. Witt. *Database Design - Know It All*. Morgan Kaufmann, 2008.

[Bis95] J. Biskup. *Foundations of information systems*. Vieweg, Wiesbaden, 1995. In German.

[BT99] C. Beeri and B. Thalheim. Identification as a primitive of database models. In *Proc. FoMLaDO'98*, pages 19–36. Kluwer, London, 1999.

[Cel95] J. Celko. *Joe Celko's SQL for smarties - Advanced SQL programming*. Morgan Kaufmann, San Francisco, 1995.

[Cel99] J. Celko. *Joe Celko's Data and Databases: Concepts in Practice*. Morgan Kaufmann, 1999.

[Cod91] E. F. Codd. *The relational model for database management (version 2)*. Addison-Wesley, Reading, MA, 1991.

[Dat11] C.J. Date. *Go Faster – The transRelational approach to DBMS implementation*. C.J. Date & Ventus Publishing ApS, 2011.

[DMT04a] J. Demetrovics, A. Molnar, and B. Thalheim. Graphical and spreadsheet reasoning for sets of functional dependencies. In *Proc. ER'2004*, LNCS 3255, pages 54–66, 2004.

[DMT04b] J. Demetrovics, A. Molnar, and B. Thalheim. Graphical and spreadsheet reasoning for sets of functional dependencies. Technical Report 0402, Kiel University, Computer Science Institute, http://www.informatik.uni-kiel.de/reports/2004/0402.html, 2004.

[DMT04c] J. Demetrovics, A. Molnár, and B. Thalheim. Graphical reasoning for sets of functional dependencies. In P. Atzeni, W. W. Chu, H. Lu, S. Zhou, and Tok Wang Ling, editors, *Proc. Conceptual Modeling - ER 2004*, volume 3288 of *LNCS*, pages 166–179. Springer, 2004.

[KL18] H. Köhler and S. Link. SQL schema design: foundations, normal forms, and normalization. *Information Systems*, 76:88–113, 2018.

[KT11] M. Klettke and B. Thalheim. Evolution and migration of information systems. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 12, pages 381–420. Springer, Berlin, 2011.

[Leo92] M. Leonard. *Database design theory*. MacMillan, Houndsmills, 1992.

[LTN07] S. Lightstone, T. Teorey, and T. Nadeau. *Physical database design*. Morgan Kaufmann, 2007.

[MR98] J. A. Makowsky and E. V. Ravve. Dependency preserving refinements and the fundamental problem of database design. *DKE*, 24(3):277–312, 1998. Special Issue: ER'96 (ed. B. Thalheim).

[PBGG89] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The structure of the relational database model*. Springer, Berlin, 1989.

[PP13] G.P. Popkov and V.K. Popkov. A system of distributed data processing (In Russian). *Vestnik Buryatskogo Gosudarstvennogo Universiteta*, (9):174–181, 2013.

[Ste00] M. Steeg. *RADD/raddstar - A rule-based database schema compiler, evaluator, and optimizer*. PhD thesis, BTU Cottbus, Computer Science Institute, Cottbus, October 2000.

[SW05] G. Simsion and G.C. Witt. *Data modeling essentials*. Morgan Kaufmann, San Francisco, 2005.

[Tha91] B. Thalheim. *Dependencies in relational databases*. Teubner, Leipzig, 1991.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.

[Tha03] B. Thalheim. Conceptual treatment of multivalued dependencies. In *ER'2003*, LNCS 2813, pages 363–375, 2003.

[Tha13] B. Thalheim. Open problems of information systems research and technology. In *Invited Keynote, BIR'2013. LNBIB 158*, pages 10–18. Springer, 2013.

[Tha19a] B. Thalheim. Conceptual models and their foundations. In *Proc. MEDI2019, LNCS 11815*, page tba. Springer, 2019.

[Tha19b] B. Thalheim. Semiotics in databases, keynote paper. In *Proc. MEDI2019, LNCS 11815*, page tba. Springer, 2019.

[VBS91] R.J. Veldwijk, M. Boogaard, and E. Spoor. Extending the relational model version 2 to support generalization hierarchies. WorkingPaper 1991-78, Faculty of Economics and Business Administration, Vrije Universiteit Amsterdam, 1991.

[Web95] B. F. Webster. *Pitfalls of object-oriented development: a guide for the wary and entusiastic*. M&T books, New York, 1995.

[WT13] Q. Wang and B. Thalheim. Data migration: A theoretical perspective. *DKE*, 87:260–278, 2013.

[Yan86] C.-C. Yang. *Relational Databases*. Prentice-Hall, Englewood Cliffs, 1986.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

From Models_For_Programming To Modelling_To_Program and Towards Models_As_A_Program

Bernhard Thalheim[0000−0002−7909−7786]

Department of Computer Science, University Kiel, Germany
bernhard.thalheim@email.uni-kiel.de

Abstract. The history of programming languages can be separated into four or five generations. Most languages are nowadays at the level of the third or fourth generation. The fifth generation programme failed mainly due to the infrastructure that has been available at that time. We are going to revive this firth generation programming efforts by deployment of models as a source code for compilation of programs. Currently models are used as a blueprint or as some inspiration for programmers. At present, we are able to develop an approach of modelling to program. In future, we might have models as programs. This programme will then result in true fifth generation programming.
This Ansatz requires models at a higher quality level. Models at this level will become executable if they are precise and accurate. This paper summarises and discusses the vision that models will become programs in future. Models are used for programming as a communication mediator. Modelling as an art will be an integrative part of program development. Mastering modelling as technology will result in modelling as programming. In future, models will become itself programs. We present some of the main issues for research.

Keywords: models for programming, modelling to program, modelling as programming, models as programs, true fifth generation programming

1 The Vision: Next Generation Programming

Modelling can be considered as the fourth dimension of Computer Science and Computer Engineering beside structuring, evolution, and collaboration. Models are widely applied and used in everyday life and are widely deployed in our area. Typical scenarios are: prescription and system construction, communication and negotiation, description and conceptualisation, documentation, explanation and discovery for applications, knowledge discovery and experience propagation, and explanation and discovery for systems. Programming is currently based on intentional development models. Models are often explicitly specified. Sometimes they are implicit. We concentrate in this chapter on the first scenario, i.e. models as a means for construction. Generation zero of model usage is

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

the starting point. The current state-of-the-art can be characterised as a movement from *Modelling-for-Programming (M4P)* where models are used as yet-another-development document for programming or throw-away-inspiration for the coding towards *Modelling_to_Program (M2P)* which is essentially a revival or reactivation or renaissance of modelling as an art. The main activity for M2P is model-based development. They are useful for developing the program in its essentials. But there are other ways to code without models.

We may distinguish three generations for this agenda for the system construction scenario after revival, reactivation, and enlivenment of models instead of only using models beside blueprint and inspiration:

First generation: **Modelling to Program** (M2P). Models can also be used as a source code for program generation. The result of such generation process may be enhanced by program refinement approaches. The model must be of a higher quality compared to current approaches. It must be as precise and accurate as appropriate for the programmer's task without loss of its explanation power. It must become adaptable to new circumstances. Maintenance of programs can be supported by maintenance of models. The first generation is thus based on *model-based development* and reasoning.

Second generation: **Modelling as Programming** (MaP). Modelling becomes an activity as programming nowadays. Some models can directly be used as an essential part of sources for programs, esp. executable programs. They are neatly integratable with other source codes. Modelling languages have reached a maturity that allows to consider a model at the same level of precision and accuracy as programs. Models are validated and verified and as such the basis for program validation and verification. Models will become adaptable and changeable. Computer Engineering incorporates modelling.

Third generation: **Models as a Program** (MaaP). Models can be directly used as a source code for or instead of programs. If a separation of concern approach is used for model development then the source for programs consists of a *model suite* with well associated sub-models. Compilers will translate the model or the model suite to executable programs. Maintenance of programs is handled at the model level. Models can be directly translated as programs, i.e. programming can be performed almost entirely through modelling. Third generation modelling is then *true fifth generation programming* which essentially frees the program developer from writing third or fourth generation programs. Everybody – also programming laymen and non-programmers – who can specify models will become a programmer.

The current-state-of-the-art is going to be analysed in the next section. The following sections discuss then our vision for the second and third generation. Figure 1 visualises the vision towards models as programs with the current situation, towards modelling to program and modelling as programming, and finally models as a program.

The final stage of model-based reasoning for system construction will be the usage of models as a program. We shall then partially replace programming by models. The transformation of the model suite to a program (MaaP) will also

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

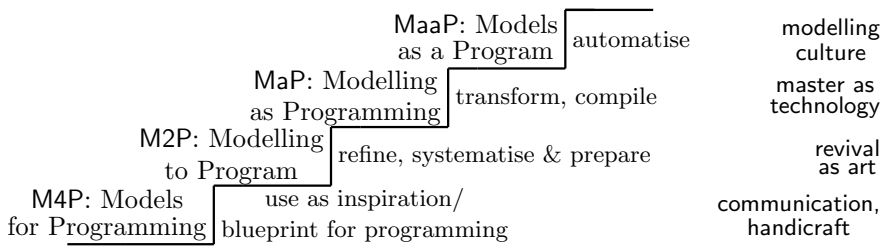


Fig. 1. From current state-of-art to next generation programming as model-based thinking

be based on a compiler. Programming will be mainly modelling. In this case, we may consider modelling as true fifth generation programming.

The Storyline of this Vision Paper

This paper presents the vision for first, second and third generation modelling as an outline for a series of workshops. Visions are visions. We thus need to sketch the path towards a new way of programming that is called true fifth generation programming (5PL) in Section 5. This paper aims at introducing the entire M2P, MaP, and MaaP programme. We abstain from long citation lists in the next sections. Subsection 3.2 summarises the large body of knowledge that is already available.

The current Ansatz in Computer Science is to utilize models as an blueprint for programming or at least as inspiration. Models can be thrown away after programs have been developed. The software crisis, the data crisis, and the infrastructure crises show that the model_for_programming approach must be revised. Section 2 discusses well-known approaches to model-based development, its issues, and its problems. Typical specific approaches are model-driven development, conceptual-model programming, model-driven web engineering, and models@runtime. We realise that model-based development is still often essentially model-backed development. For this reason, Section 3 revisits the current state-of-art. The body of knowledge already supports to step towards modelling to programming. There are tools available. These tools may be combined with heritage programming. Since a model typically focuses on some aspects on certain abstraction level, we have to use a collection of well-associated models, i.e. a model suite. Model suite also enable in layered model-based development.

Section 4 discusses features, approaches, and ideas for modelling to program and furthermore for modelling as programming. These modelling approaches can be supported by literate modelling that uses model suites. Section 5 discusses then some of the deliverables we envision for true fifth generation programming. We might use layering for generation of programs based on a suite of normal models and a landscape initialisation based on corresponding deep models. This approach will be discussed mainly on the experience we have learned with the

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

separation into initialisation and specification. This approach is a typical onion Ansatz that has already been used for LaTeX. We thus generalise this onion Ansatz and discuss its potential. We use two case studies from [50, 65, 74, 99]. for the discussion. A summary is sketched in the final Section 6.

2 Current Approaches to Model-Based Development

Models are used as instruments in many reasoning and especially engineering scenarios. The reasoning and development process is then model-determined. Depending on the function that a model plays in a programmer's scenario we distinguish three general roles for models: (1) models are instruments for reasoning which implies their prior construction and the reasoning necessary for their construction; (2) models as targets of reasoning; (3) models as a unique subject of reasoning and its preliminary. These roles have to be supported by sophisticated reasoning mechanisms such as logical calculi. Modelling uses beside the classical deductive reasoning also other reasoning approaches such as Programming and reasoning must be supported by a mechanism, e.g. logical calculi with abduction or induction.

2.1 M4P: Models for Programming

Model-based reasoning is an essential feature of all mental models such as perception models which are representing some (augmented) reality and domain-situation models which are representing a commonly agreed understanding of a community of practice. Model-based reasoning supports modelling based on data similar to inverse modelling. It comprehends the background of each model considered. It allows to consider the corresponding limitations and the obstinacy of models, esp. modelling languages. As a reasoning procedure, it is enhanced by well-formed calculi for reasoning. Model-based reasoning is compatible with all kinds of development procedures (water, spiral, agile, extreme). It also allows to handle uncertainties and incompleteness of any kind.

The model-for-programming approach uses models as some kind of blueprint and as a description mediator between the necessities in the application area and the program realisation. After programming has

Model-based development and engineering is a specific form of engineering where the model or model suite is used as a *mediator* between ideas from the application domain and the codification of these ideas within a system environment. It has been considered for a long time as a *greenfield* development technique that starts with requirements acquisition, elicitation, and formulation, that continues with system specification, and terminates with system coding. Models are used as mediating instruments that allow to separate the description phase from the prescription phase. Engineering is, however, nowadays often starting with legacy systems that must be modernised, extended, tuned, improved, etc. This kind of *brownfield* development may be based on models for the legacy systems and on

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

macro-models representing migration strategies that guide the system renovation and modernisation. The heritage model (or legacy models) is used as an origin for a sub-model of the target system.

The four supporting means for model-based engineering are the modelling know-how, the experience gained in modelling practices, the modelling theory, and finally the modelling economics. The last two means, however, need a deeper investigation. Specific forms of model-based reasoning for system construction are, for instance,

- *model-driven architectures and development* based on a specific phase-oriented modelling mould,
- *conceptual-model programming* oriented on executable conceptual models,
- *models@runtime* that applicabies models and abstractions to the runtime environment,
- *universal applications* based on generic and refinable models and with generators for derivation of the specific application,
- *domain-specific modelling* based on domain-specific languages,
- *framework-driven modelling* (e.g. GERA or MontiCore),
- *pattern-based development* based on refinable pattern,
- *roundtrip engineering* supported by tools that maintain the coexistence of code and model,
- *model programming* where the model is already the code and lower levels of code are simply generated and compiled behind the scenes,
- *inverse modelling* that uses parameter and context instantiation for model refinement,
- *reference modelling* that is based on well-developed and adaptable reference models, and
- *model forensics* which starts with model detection through analysis of the code, i.e. the model origin is the code.

These approaches develop models by stepwise refinement of the root or initial model, by selection and integration of model variations, and by mutation and recombination of the model. Models are typically model suites.

The mediator function of models is illustrated for 'greenfield' system construction in Figure 2. System construction is a specific kind of modelling scenario that integrates description and prescription. It might also be combined with the conceptualisation scenario. B. Mahr pointed out that origins also come with their theories. We may add also the deep model behind the origin.

During a *relevance stage* (or cycle), we first reason about the ways of operating in an application. This step may also include field testing of current artifacts within the given environment. Next we select, reason about, and revise those properties $\Phi(O)$ that are of relevance for system construction. The next phase is based on classical requirements engineering. We identify requirements (or business needs). These requirements become objectives $\Psi(M)$ which must be satisfied by the model. These objectives are biased by the community of practice (CoP).

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

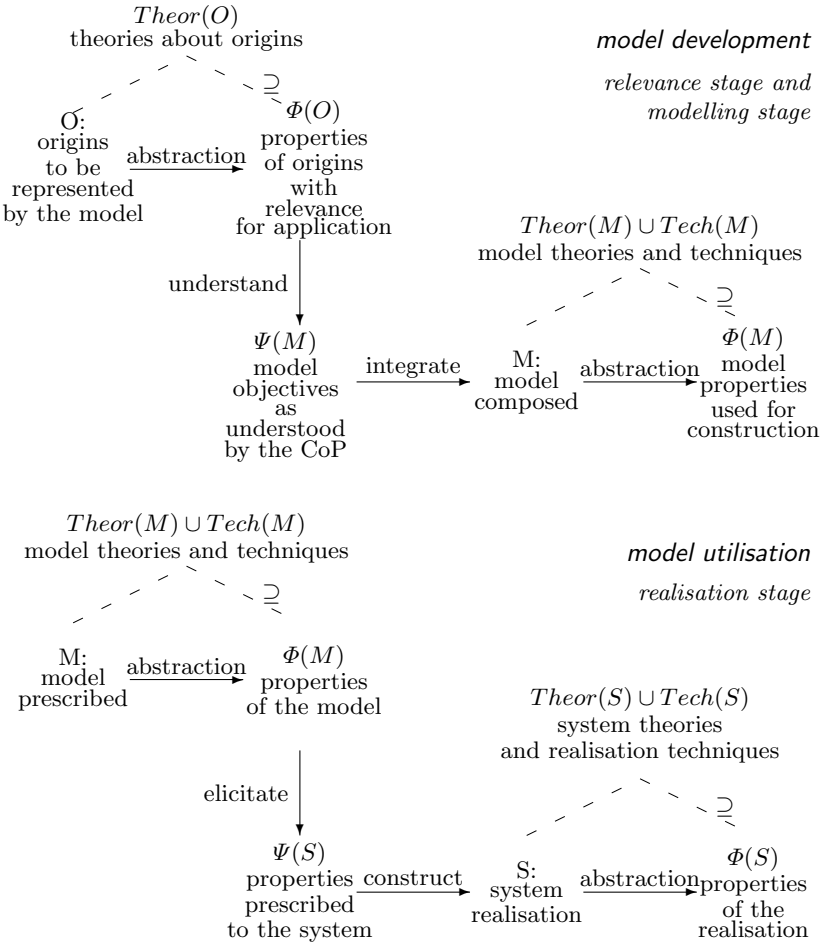


Fig. 2. Straightforward model-based system development starting with description models which are transformed to prescription models for system realisation (modified from [96])

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

The *modelling stage* starts with the objectives as they are understood in the community of practice. Modellers compile and integrate these objectives in a model (or model suite). The 'ways of modelling" is characterised

- by the *modelling acts* with its specifics,
- the *theories and techniques that underpin modelling acts*,
- the *modellers* involved into with their obligations, permissions, and restrictions, with their roles and rights, and with their play;
- the *aspects* that are under consideration for the current modelling acts;
- the *objectives* that are guiding the way of modelling;
- the *resources* that bias the modelling act.

The *realisation stage* uses prescription properties as a guideline for construction the system. Model-based engineering is oriented on automatic compilation of the entire system or at least of parts of it. If the system is partially constructed and thus must be extended then the compilation must provide hooks in the compiled code. These hooks allow to extend the system without revising the rest of the compiled code.
Validation compares the properties of origins $\Phi(O)$ with the properties of the model $\Phi(M)$. *Verification* compares the properties of model $\Phi(M)$ with the properties of the system $\Phi(S)$. *Model assessment* compares the model objectives $\Psi(M)$ with the properties of the model $\Phi(M)$. *System assessment* compares the system objectives $\Psi(S)$ with the properties of the system $\Phi(S)$.

2.2 Model-Driven Development

In general, model-driven development is nothing else than a very specific form of the model development and usage process. We use only the last three in a specific form. The first two levels are assumed to be given.

Essentially, model-driven development (MDD) does not start with a computation-independent model. The process starts with modelling initialisation. The CIM is already based on a number of domain-situation models and some insight into the application situation or a model of the application situation. These models are the origins of the models under development and are then consolidated or toughened. The CIM describes the system environment, the business context, and the business requirements in a form that is used by the practitioners in the application domain. This model becomes refined to the PIM by services, interfaces, supporting means, and components that the system must provide to the business, independent of the platform selected for the system realisation. The chosen platform forms the infrastructure for the system realisation. The PIM is then refined to a platform-specific model PSM. In general, a number of models are developed, i.e. a model suite as shown in Figure 3.

The main concept behind this approach is the *independence* of the models on the platform respectively on the algorithmics setting. This independence allows to separate the phases. A system is then specified independently of the software execution platform and independently of the chosen set of algorithms.

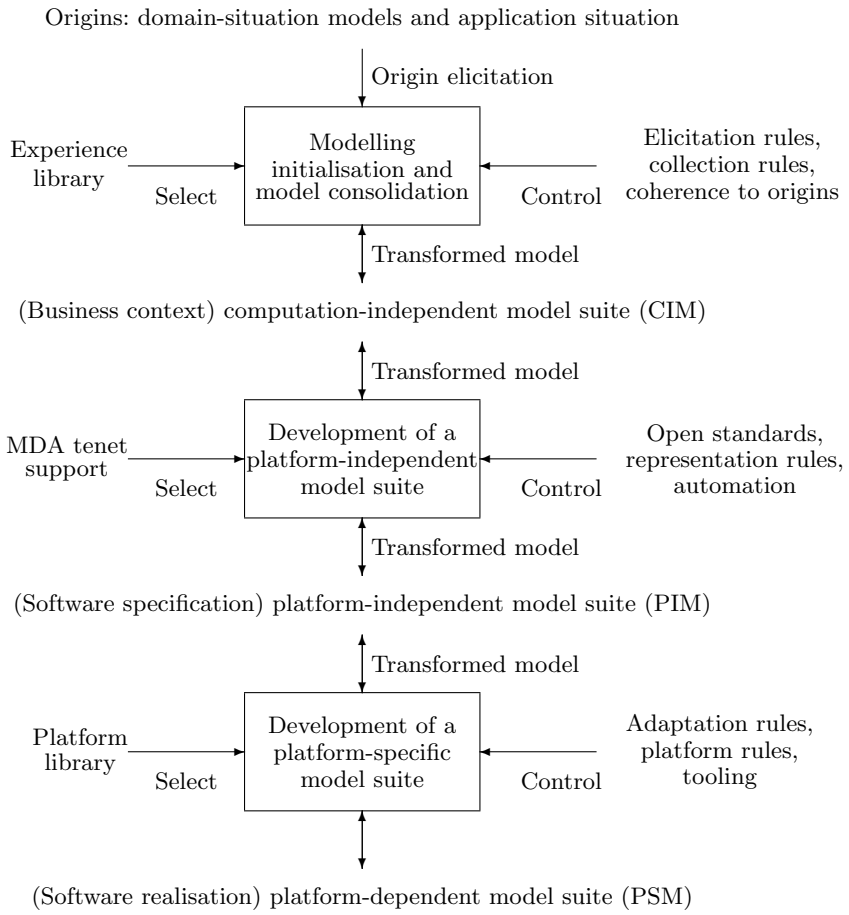


Fig. 3. The three phases in model-driven development for construction scenarios

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

The transformation is typically a two directional. Model-driven architectures are based on three tenets: direct and proper representation of concepts and terminology in the application domain, automation for generation of routine components instead of human interaction, and open standards promoting reuse and proper tool deployment.

Model driven development assumes that the models at each phase can be transformed to each other. This rigid requirement cannot be used in many construction scenarios. For instance, database model development is based on forgetful mappings. The conceptual model contains conceptual information that is irrelevant for the "logical" or "physical" models and thus neglected during development.

2.3 Conceptual-Model Programming

Conceptual-model programming uses a compiler approach to programming. The model will be the source code for the compiler or transformer. The result of compilation should be then code which might be used for another compilation. The compiler assures that the model might already be used as (the final) code. The execution of the code corresponds to the conceptual specification.

Advanced conceptual-model programming is based on three theses:

- Conceptual-modelling languages must provide compiler facilities that allows to directly execute models.
- Conceptual-modelling languages must support co-design of database structure, of user interfaces, and of database access and user interaction.
- "The model-compiled code is beyond the purview of conceptual-model programming programmers — both for initially creating the application system being developed and for enhancing or evolving the application system." [31].

As a consequence, application-system development becomes entirely model-driven. Conceptual-model programming constitutes model-complete software development.

This approach requires that models are of high quality.

- *Models are complete and holistic.* The conceptual model supports all necessary co-design elements such as structuring, behaviour, and user support. The model itself can be a model suite.
- *Models are conceptual but precise.* All model elements must be precisely defined through proper element definition. At the same time, these elements have to be properly associated to the concept space that is used for conceptualisation of the model. Parsimony and economy of model elements guide the notation.

Provided that the model has high quality then evolution of an application can be directly represented through evolution of the model. It, thus, asserts that conceptual-model programming is essentially programming. The model is the kernel of code that can be easily adapted to the specific platform requirements.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

2.4 Model-Driven Web Engineering

Model-driven development and engineering can be based on generic models. These models ease the selection of the normal model in dependence on a computation-independent model suite. They are also early serving as an additional entry at the modelling initialisation phase (Figure 3) and at the landscape (Figure 4) determination. The problem space, focus, scope, and codified concepts will be set through the utilisation of such generic models. Additionally, generic models will be used at the extrinsic (source) reflection layer (Figure 4). Generic models and reference models can be refined by the data and other information on hand.

Website development and web information system development is a typical example of model-based reasoning on the basis of generic models and of refinable specifications. Website construction benefits from the usage of previous models and programs, esp. generic ones. These generic models compile experience gained in similar web engineering projects. This experience elicitation, evaluation, appreciation, and acquisition is a specific *rigor stage* (or cycle). Generic models stem from models that have been already used. The generalisation procedure that led to the generic model allows to derive the specific refinement mechanisms for mapping the generic model to the old specific one. Generic models come then with a *refinable specification* and with a *refinement calculus*.

Model-based reasoning and website model-based development is based on the specification language SiteLang (see below). Website development is page-oriented where pages are essentially media objects with their structuring, their functionality (especially navigation, import/export, search), their runtime adaptation features to actors or users, their databases support, and their specific navigation-oriented flow of work. Websites can be categorised, e.g. business websites for a business with customer collaboration according to the used business culture. This categorisation is the basis for generic model suites of the website. The normal model can be then derived from the generic model and the computation-independent model.

2.5 Models@Runtime

Most model-driven development and most-driven architecture are concerned with the development process. Models@Runtime take a different turn towards support of software after it has been installed. Beside model-based evolution support, models are additionally developed on top of the code. The code will then be the origin. The model suite reflects essential properties of a system. This approach aims at development of proper performance models that characterise the software and its behaviour.

The Model@Runtime approach extends the applicability of models and abstractions to the runtime environment. This approach provides means for handling complexity of software and its evolution if effective technologies exist for code analysis and change in a given environment. Models are used for reasoning about runtime behaviour, for appropriate adaptation of systems at runtime, and

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

for capturing runtime phenomena in systems such as time, memory, energy, location, platform, and personalisation concerns. Models provide meta-information, for instance, in order to automate runtime decision-making, safe adaptation of runtime behaviour, and optimal evolution or modernisation of systems. Modern systems are increasingly complex. This complexity is a challenge that needs to be managed especially in the case of evolving hybrid infrastructures consisting of a manifold of heterogeneous services, resources, and devices.

Typical tasks are the following ones: (a) creating or updating the models suite that represents a system according to evolution of the system or to changes in system's environments; (b) developing appropriate adaptation strategies; analysing and maintaining model suites while parts of the corresponding systems are changing; (c) propagating changes in the model suite back to the system; (d) adaption of emerging systems and their model suites in ways that cannot be anticipated at model development and system programming time; (e) enabling features supporting continuous design, deployment, and quality of service analysis for performance optimisation; (f) optimisation and tuning; (g) reducing uncertainty that arise due to environment changes, due to system integration and migration, due to changes of quality concerns, and due to changes and extensions of the system user communities and evolving viewpoints.

We envision that the modelling-as-programming approach allows to solve some of these challenges. Models@Runtime are an example of model-based reasoning despite the classical system construction scenario. Models are used for exploration, for exploration, for discovery of obstacles, for observation, and improvement of current solution, i.e. the scenarios targeted on system modernisation beside model-based development. It integrates also model checking.

2.6 Lessons Learned with Model-Based Development

Model-driven development highly depends on the quality of models. Although model-driven development will be supported by compilers in the future, it is currently mainly using an interpreter approach. Models must be doubly well-formed according to well-formedness criteria for the model itself and well-formed for proper transformation in the interpreter approach. For instance, database schemata have to be normalised. BPMN diagrams must be well-formed in order to be uniquely interpretable. Otherwise, the interpreter approach will not result in models that have a unique meaning. Model-based development has at least four different but interleaved facets: (i) model-driven development with stepwise adaptation of the model (suite), (ii) model-driven programming as conceptual-model programming, (iii) model-based development with generic or reference models as the starting point, and (v) model-based assessment and improvement of programs. The interpreter approach can be sufficient in the case of high-quality models. The classical approach to database modelling that uses a normalisation after translation, however, shows that compiler approaches are better fitted to model-based reasoning and development and to Modelling as Programming. If a model suite is going to be used as the basis for model-based development then the models in the model suite must be tightly associated. Tracers and controllers

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

for associations among sub-models and coherence within a model suite are an essential prerequisite for proper model-based development. Often model suites are only loosely coupled. A compiler has to support sophisticated integration and harmonisation in the last compiler phase what is also a theoretical challenge.

3 Revisiting The State-Of-Art

3.1 Currently: Model-Backed Development

Model-based development, model-based design, and model-based architecture use a world of models. So far, we did not reach the stage that models are used for generation of programs. Programming is rather based on blueprint models or inspiration models. These models will not be changed whenever the program is under change. They are then throwaway origins in the programming process. Development is therefore model-backed. The potential for model-based development is, however, obvious. Software engineering is now paying far more attention to the needs, cultures, habits, and activities of business users. Users have their own understanding of the application that must be harmonised with the perception of developers.

Model-based development is currently revitalised [1], e.g. in the MontiCore project (see, for instance, [44]). The revival led to a new stage in generative software engineering for domain-specific languages which reflect the worlds of business users. In general however, modelling is still not considered to be a mandatory activity for programmers. It is still considered to be some kind of luxury.

We further observe that model-backed development has already been applied directly with the beginning of programming. Programmers have had their models. These models have, however, been implicit and have been stated rather seldom. Changes in the software did not use the models behind. They rather led to additional hidden models. These implicit models became hidden legacy since documentation of software has been and is still a big issue and is often completely neglected. The explicit model-backed development became important with the advent of the internet software and the turn towards user-oriented software.

3.2 The Body of Knowledge

Our approach to model-based reasoning is based on [20, 69, 77]. Figure 2 follows the reconsideration in [96] of the work by B. Mahr [70] Other variants of model-based development are conceptual-model programming and model driven architectures [31, 81], universal applications with generators for derivation of the specific application [78], pattern-based [6], and many other project like the CodeToGo project. SPICE and CMM added to this approach quality issues and matured co-design methodology [10, 36, 48, 86, 95]. Model-driven development, engineering and architecture (MDD, MDE, MDA) taught some valuable lessons reported about model-driven approaches, e.g. [27, 37] and the list in [108]. Model-driven development can be extended by literate programming [60], database

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

programming [88], programming with GibHub [52], 'holon' programming [29], refinement [13], multi-language approaches [33], and schemata of cognitive semantics [66]. Projects like the Axiom project [25], the mathematical problem solver [83], and RADD (Rapid application and database development) [7, 92, 94] show how MaP can be accomplished.

An essential method for model-based programming is based on refinement styles. A typical refinement style is practised in the abstract state machine approach [15, 12]. A program is specified in a rather abstract block-oriented form based on changeable functions. These function can be refined to more detailed ones without context dependence to already specified ones.

Our vision approach can also be based on interpreters instead of compilers or compiler-compilers. Typical examples are [24, 23] using MetaCASE [55, 54], Come-In-And-Play-Out [43] based on algorithmics [19, 42], and model-driven web engineering approaches (beside [87], for instance, Hera [45], HDM [39], MIDAS [107], Netsilon [76], OOHDM [89], OOWS [80] RMM [47], UWE [61], WAE2 [21], Webile [85] WebML [18], WebSA [73], W2000 [8], and WSDM [104]). The interpreter approach is useful in the case of relatively simple modelling languages.

The interpreter approach to partially (and fragmentary) program generation can be applied as long as languages are strictly layered and there is no dependence among the layers. Optimisation is not considered. First interpreter approach to database structuring followed this approach for the entity-relationship model (based on rigid normalisation of the source schema before interpreting with attribute, entity, relationship layers whereas the last one allows very simple cardinality constraints). Constraint enforcement is a difficult problem which requires compilation, denormalisation, and specific supporting means.

The compiler approach [82, 110] allows to generate proper programs. The rule-based approach to database schema transformation in [63] extends [32]. It uses the theories of extended entity-relation modelling languages [94] and the insight into techniques such as web information systems [87] and BPMN semantification [14, 16]. We use advanced programming techniques and theories like attribute grammars [26], graph grammars [30, 94], database programming through VisualSQL [49], performance improvement [102], and normalisation techniques like those for storyboards in [74].

The fifth generation computer project [2, 3, 34, 75, 106, 105] inspired our approach to modelling as programming. We base our changes on advices by H. Aiso [4] who chaired the architecture sub-committee in the Japanese fifth generation computer project.

3.3 Experience Propagation Through Reference Models

Computer Engineering developed a rich body of successful applications that form a body of experience. This experience can also be used in a completely different development style instead of 'greenfield' or 'brownfield' development. Already by investigating so-called 'legacy' systems, we have had to realise that solutions incorporate tacit knowledge of programmers. We should therefore call these older solutions better *heritage* since they allow us to inherit the skills of

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

generations of programmers despite changing hardware and software. Heritage systems provide a rich body of already available solutions.

A typical direction of heritage system development are reference models. Reference models [9, 35, 71, 101, 91] are generalisations of existing successful system solutions. The experience gained in such applications can be generalised to classes of similar solutions. The generalisation procedure allow to invert to generalisation to a specialisation to the given solution. Universal programming [78] and generic solutions [103] use generation facilities for deriving an essential part of a solution from already existing more general ones.

3.4 Tools for Model-Based Development

The fundamental idea behind MetaCASE and its incorporation to a sophisticated tool support for modelling [23] is the separation of models from their visual representations. MetaCASE is a layered database architecture consisting of four OMG layers: signature, language, model and data.

The computational environment for the approach can be based on systems ADOxx, Eclipse, Eugenia, GMF, Kieler, mathematical problem solvers, MontiCore, and PtolemyII [53, 28, 44, 56, 62, 83, 84]. The two case studies mentioned below have been discussed in [50, 65, 74, 99]. More examples on models in science are discussed in [98].

The compiler-compiler approaches [17, 41, 46, 68] are far more powerful. They have been developed for domain-specific languages (at that time called 'Fachsprachen') since 1973. This approach is our main background for modelling as programming. It can be combined for 'brownfield' development (migration, modernisation) with the strategies in [58]. The directive and pragma approaches have been developed already for FORTRAN and have been extensively used for C, C++, and especially ADA [93]. The layered approach used in this chapter follows the realisations already known and widely applied for programming languages since COBOL and ALGOL60. Layering is also the guiding paradigm behind LaTeX and TeX [59, 67] with a general setup layer, the content layer, the adaptable device-independent layer, and the delivery layer. The compiler-compiler approach additionally integrates generic models [11, 87, 100, 103], reference libraries [35, 101], meta-data management [64], informative models [97], model-centric architectures [72], and multi-level modelling [38].

3.5 The Background: Model Suites

Modelling should follow the *principle of parsimony*: Keep the model as simple and as context-independent as possible.

Models that follow this principle provide best surveyability and understandability. Context-dependence would otherwise require to consider all other models and origins together with the model. The result would be a model that is not really ready for use.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

Another principle is the *separation of concern*: Instead of considering a holistic model which allows to consider all aspects and functions, we use a number of models that concentrate on few aspects and support few functions.

This second principle requires consideration according to aspects such as structure, functionality, behaviour, collaboration as the triple (communication, cooperation, coordination), interactivity, infrastructure, storage, and computation. The classical co-design approach to database system development follows this principle by integrated consideration of database structure models, functionality and behaviour models, viewpoint models, and realisation models. The principle can also be based on the W*H separation [24], i.e who (relating to user profile, habits, personal culture), where (wherein, wherefrom, which orientation), what (whereof, wherefore, wherewith), why (whence, whither, for what, whereto), in what way (how, by what means, worthiness, which pattern, on which basis), when (at what moment and time frame), whom (by whom, whichever), and whereby (which enabling system, infrastructure, languages with their own obstinacy and restrictions ...). Separation of concern enables to consider from one side inner and implicit models as deep models and from the other side outer and explicit models as normal models. Deep models are typically stable and will not change. Normal models are more tightly associated with the origins that are really under consideration.

Following the two principles, we use a collection of models. The entire picture is then derived from these models similar to the global-as-view approach [111]. These models do not need to be entirely integrated. It is only required that the models in the collection are coherent. Utilisation of several models requires support for co-existence, consistency among abstractions, and integration of deep and normal models. Models in such collections or ensembles are also partially governing other models, i.e. we can use a synergetic separation into master and slave models. This separation provides a means to use control models for master models. We do need control models for slave models.

Coherence becomes then a main property of such model collection. This coherence must be maintained. The simplest way for support of coherence is to build explicit associations among these models. Associations can be build on general association schemata. In this case we can develop tracers and controllers for the model association. The association represents then the architecture of the model collection. Associations among models may also based on association among sub-models or on substitutability of a submodel by another submodel or on composers of models.

A *model suite* is a coherent collection of models with an explicit association schema among the models, tracers for detection of deviations from coherence, and controllers for maintenance of the association.

3.6 The Trick: Layered Model Development

The model suite in Figure 4 will be layered into models for initialisation and landscaping, for strategic intrinsic set-up, for tactic extrinsic reflection and definition, for customisation and operationalising including adaptation, and for

model delivery. Figure 4 displays layering for greenfield development. Brownfield development is based on revision for modernisation, integration, evolution, and migration. It uses also reengineering of models that become additional origins with their normal models, deep models, mentalistic and codified concepts. Heritage development does not follow layering in this way.

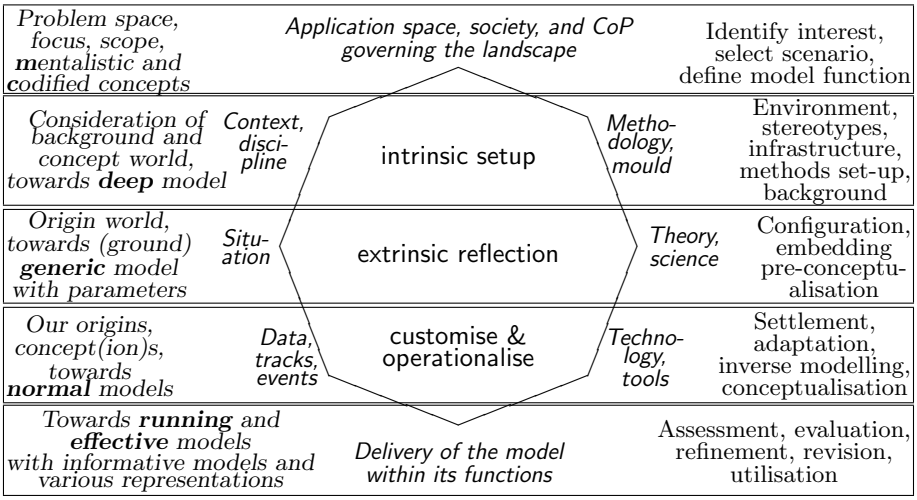


Fig. 4. The layered approach to model suite development and program generation (revised and modified from [50])

We use the W*H characterisation for *landscaping* as initialisation of a layered model suite. Figure 4 separates from one side supporters such as sources, grounding and basis on the left side and from the other side enablers such as methodology, theories, techniques on the right side.

4 Model-Based Thinking

Model-based thinking is going to integrate modelling into programming. Models can be used for 'semantification' of programs. Model-based thinking is different from programming since it uses models of humans together with models of the computing infrastructure. These models can be oriented on the way of programming, controlling, and supporting infrastructure. The way of modelling allows to concurrently consider several variants and alternatives for a solution, to use preference assessment for 'good' solutions, and to separate concerns by well-associated with each other models.

Models are used in everyday life as mental prospect or idea. In this case they are less precise than those used in system construction scenarios. Program development needs a higher level of accuracy and precision. We concentrate here on high-quality models, i.e.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

· with potential extensibility, adaptability, replaceability, efficiency, stability, validatable, testable, analysable, exactness, error-proneness, reliable spacial and temporal behaviour, dependability, maintainable, etc. from one side of external quality and
· with inner coherence, toleratable redundancy, constructivity, modularisation, coherence, and efficiency, etc. from the other side of internal quality.

4.1 The Modelling Method for Programming

Let us briefly remember generations of programming languages and derive that M2P is a natural and obvious continuation of the development of programming languages. First generation programming was oriented on executable processor-dependent code. The second generation brought processor independence and introduced symbolic and mnemotic code. It provided thus some hardware independence. Assembling programs were interpreted by an assembler. They have been oriented on the *way of controlling* the computation. Both generations allowed a very efficient and effective full treatment of all processes performed at the computer. At the same time, programming was rather low level machine-oriented programming and, thus, a challenging task for well-educated specialists.

The third generation languages standardised Von Neumann constructs to higher level constructs that can be used within a standardised structure of programs. These constructs are context-reduced and exempted from the plague of firmware dependence. The languages are oriented on an abstraction of the *way of computing* (or working) of computers. Syntax and partially semantics of languages became standardised and well-defined. Meanwhile, these constructs are near to human languages. Third generation programming became somehow independent on the enabling infrastructure. Structuring of programs has been standardised and component-backed. The main enablers for third generation languages are compilers, debuggers, linkers, and editors. Compilers incorporated also optimising programs for generation of efficiently executable code at the level of second or first generation languages. Programming could be thus performed by everybody who got computer engineering education.

Fourth generation languages provided implementation and groundware independence, e.g. data storage and management independence and management system independence. These languages are macro-based instead of command-oriented and are often script languages. Optimisation is now available for a bunch of concurrently executed programs. Groundware independence can nowadays also provided in networked computation. Third and fourd generation programming can be based on a number of different paradigms such as object-orientation, parallel computing, functional programming, imperative or procedural programming, symbolic computation, collection-oriented (e.g. set- or bag-oriented) programming, event-driven programming, and declarative programming.

Fifth generation languages raised the hope for program development in computational environments that could ‘think’ for themselves and draw their own inferences using background information. The approach was mainly based on representing programs as massives of formulas in first-order predicate logics.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

One of the reasons that this program failed is this restriction to first-order predicate logics. Other reasons are: "it was highly dependent on AI technology, it did not achieve an integration of AI and human-computer interface techniques, ... it tried to provide a universal solution for any kind of programming, it routed granularity to basic program blocks, and it was oriented on one final solution instead of coherent reasoning of coherent variants of final solutions depending on the focus and scope of a user." [50]

Our vision is to integrate the way of programming and the way of modelling. Models are typically model suites and allow to concentrate on certain aspects while not loosing the association among the models in the suite. This approach decreases the complexity of reasoning to a level that a human can easily capture it. Since the model suite is integrated, this reduction does not mean to loose the entire view on a program. Model-based thinking is also based on approximate reasoning since a model can be an abstraction of another more precise or more accurate model. Models incorporate their explanations, the necessary knowledge from the application domain, and the reasoning and thinking approaches of users. If models and model suites can be transformed to programs then maintenance of such programs can be transferred back to maintenance odf models.

4.2 The Near Future M2P: The Modelling_To_Program Initiative

M2P is based on a consolidation of the body of knowledge for models, modelling activities, and modelling. The renaissance of modelling towards a real art of modelling allows to use modelling as a matured practice in Computer Science and Computer Engineering, esp. for system development and software system development. The previous chapters contributed to this development. M2P can be understood as a technology for model-based development and model-based reasoning.

Model-based reasoning and thinking is the fourth dimension of human life beside reflection and observation, socialising and interacting, acting and fabricating (e.g. engineering), and systematising and conceptualising (typical for scientific disciplines). Programs reflect a very specific way of acting with a machine. Programming is so far bound to Von Neumann machines. It will be extended to other machines as well, e.g. to interaction machines [40, 109]. Model-backed development coexists with programming since its beginning, at least at the idea and comprehension level. Documentation or informative models are developed after programming. Modelling_to_program is coordinated modelling within program development and design processes. It enhances programming, contributes to quality of programs and enables in model-oriented and purposive maintenance, change management, and system modernisation. Model and systems complement one other using separation of concern and purpose. Non-monolithic programs and systems use a variety of languages. Their coherence and integration is far simpler if we use model suites and if programs are also following architecture principles similar to the associations in the model suite.

M2P requires high-quality models. Models must be precise and accurate. They must not reflect the entire picture. They can be intentionally incomplete.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

These models may use dummies and hocks for code injection by the translator, e.g. similar to stereotype and pattern refinement [5]. Models may already contain directives and adornments as hints for translation. In most cases, we use an interpreter for transformation. A typical example is the ADOxx+ transformation for HERM++ interpreters discussed in [63]. In this case, views are represented by dummies. Late logical code optimisation is necessary after translation. M2P requires refinement and adaptation of the translated code. Models intrinsically incorporate deep models. Modelling is then mainly normal model development. An early variant of this approach is the RADD toolbox for database structure design and development [94]. Website generation [87] is another typical example of this approach.

Modelling has now reached the maturity to properly support programming at the intentional level. It is also going to be used at the meta-reasoning level [22] for guiding methodologies and methods. Modelling becomes an enabler for development of large and complex systems. It is already a commonly used technique for user-oriented and internet-based web information systems [87]. These systems support a wide variety of user frontend systems according to business user's needs. They integrate a number of different backend systems.

Modelling_to_program does not mean that programs have to be developed only on the basis of models. Models can be the source or pre-image or archetype or antetype of a program. Still we also program without an explicit model. M2P eases programming due to its explanation power, its parsimony, and comprehensibility.

4.3 From Literate Programming to Literate Modelling

Literate programming co-develops a collection of programs with their informative models [97] that provide an explanation of the programs in a more user-friendly way. It can be extended by approaches to MaP with proper compilation of source code on the basis of small programs, program snippets, and a model suite. Programs and the model suite become interwoven. The first approach to literate programming has been oriented on programs with derived libraries of user interfaces and informative models [60]. It became far more advanced. Nowadays programs are interspersed with snippets of models. Most of these models are, however, documentation models. In this case, models formulate the meaning and understanding of programs. They connect this documentation to the program code at a high level and mostly in natural language. Documentation models integrate the code documentation with the developer idea documentation, the usage documentation, and the interface documentation. Literate programming thus overcomes the habit that the documentation is going to live in the program itself. The program is going to live in the documentation model. In a similar way, interface and informative models can be handled.

Literate programming can be based on models for the central program and models for interfacing and documenting. Since the central program is the governing program, literate modelling will essentially be a global-as-design approach for

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

a model suite. Different users might use different models for the same application case.

Literate modelling is essentially modelling with integrated vertical and horizontal model suites. Horizontal model suites consist of models at the same abstraction level. A typical horizontal model suite in the global-as-design approach is the conceptual database structure model together with the view(point) external models for business users. Vertical model suites integrate models at various abstraction and stratification levels. Software development is often based on some kind of waterfall methodology. Models at a high level of abstraction are, for instance, storyboard and life case models. They are refined to models representing data, events, processes, infrastructure, and support systems. Modelling the OSI communication layer structure results in a typical vertical model suite.

Literate modelling incorporates a variety of models such as representation and informative models in a natural language into program development. It provides a high level abstraction and is thus program-language independent. The meaning of programs is provided prior to coding. Many-model thinking [79] can be developed towards model suite thinking. There are high-level introductory models such as informative models. These models are refined to models that reflect certain aspects of an application and to models that serve as origins for implementation models.

4.4 MaP: Towards Second Generation – Modelling as Programming

A central challenge of (conceptual) modelling is to facilitate the long-time dream of being able to develop (information) systems strictly by (conceptual) modeling. The approach should not only support abstract modelling of complex systems but also allow to formalize abstract specifications in ways that let developers complete programming tasks within the (conceptual) model itself. It thus generalises the model-driven and modelling_to_program approaches and develops a general approach to modelling as high-level programming.

Modelling is an activity guided by a number of postulates, paradigms, principles, and modelling styles. Already nowadays, we use paradigms such as global-as-design and principles such as meta-modelling based on generic and reference models. MaP is however dependent on deep models and the matrix. Next generation programming will also allow to be flexible in the postulates and paradigms.

Modelling can be organised in a similar way as structured programming, i.e. following a well-developed methodology and framework within an infrastructure of supporting tools. Models may be based on refinement approaches such as pattern-oriented development [5]. They contain enactor hocks for integration of source code.

Similar to system programming, modelling will be based on literate programming and literate modelling. For MaP, literate modelling becomes essential. Modelling_as_programming is oriented on development of complex systems. model suites Model suites can be developed on the basis of different frameworks as mentioned above,

A number of tools are going to support MaP:

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

- Sophisticated editors and frameworks have to be developed for this approach as extension and generalisation of existing ones, e.g. ADOxx, Kieler, Monti-Core, and Ptolemy II.
- Code generation for the general MaP programme is still a matter of future development. There are already parts and pieces that can be used for generation and compilation: the RADD workbench realisation (Rapid Application and Database Design) [94], database programming by VisualSQL tool [49], performance management and tuning tuning (e.g. [90, 102]), advance high-level workflow specification [14], integrated web information systems design, and co-design.
- The implementation approach to MaP may be inspired by three solutions that are already common for programming languages:
  - Transformation and compilation is based on standardised combinable components. These components can also be reflected by specific models within a model suite.
  - Each specialisation can be enhanced by directives for compilation and by pragmas for pre-elaboration, convention setting, and exception handling like those in C++ and ADA. Model directives configure and pre-prepare a model for compilation. Models can be enhanced by default directives or by adornments detailing the interpretation of model elements. Pragmas are used to convey "pragmatic information" for compiler controllers, adapters, context enhancers. There are language-defined pragmas that give instructions for optimization, listing control, storage, import of routines from other environments, extenders for integration into systems, etc. An implementation may support additional (implementation-defined) pragmas.
  - MaP aims at programming-language independence. In this case, it has to be supported by multi-language compilers or compiler-compiler technology. For instance, database model suites are going to be mapped in a coherent and integrated form to object-relational, network, hypertext, etc. platforms. The association among various structuring of data structure is governed by the association schema of the model suite.

MaP requires a proper education for modellers. They have to master modelling, system thinking, programming techniques, reflection of models in various ways, communication with the application experts, and design of model suites. MaP knowledge and skills will become a central module in Computer Science similar to algorithmic thinking and programming. Model suites will become the mediating device between applications and their needs from one side and the system realisation from the other side. Already programming can be understood as an experimental science and as empirical theories by means of a computing device. Modelling continues this line. MaP thus needs a proper development of a theory and technology of modelling. Continuous model quality management will become a challenging issue.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

5 Towards True Fifth Generation Programming

True fifth generation programming will be based on models which are considered to be programs due to generation of program codes from the models without programming at the level of 4PL or 3PL. Compilers or compiler-compilers are transforming the model suite directly to 3PL or 4PL code that can be compiled in the classical approach. A high-quality model suite can be used as a program of true fifth generation and will be mapped to programs in host languages of fourth or third generation.

A new generation of programming languages has to support a large variety of application areas since computers became an essential element of modern infrastructures and proper program support is necessary for all these areas, disciplines and daily life. We might try to develop a very large number of domain-specific languages. In this case, domain experts in a singleton domain are supported within their thought pattern. However, application are rather cross-domain applications with a wide variety of cultures, habits, and approaches. The literate modelling approach seems to be an alternative. In this case, model suites will thus become high-level programs and thus be the basis for true fifth generation programming in true fifth generation programming (5PL).

One potentially applicable realisation strategy is based on a layered approach discussed below similar to successful approaches such as LaTeX. We shall use this strategy in onion meta-model specification approach.

MaaP should also by partially independent on programmer's postulates, paradigms, and principles. It should also be tolerant and robust against changes in the 3PL and 4PL thus providing a programming language independence. At the same time, MaaP needs to be robust against a deviations from the normal application situation. We currently observe that application development is based on some kind of 'normal operating' in the application without taking into consideration potential but tolerable deviations from the normal case. At the same time, it must be supported by a specific MaaP education.

5.1 Ideas for 5PL on the Basis of MaaP

Models are often developed as normal models that inherit implicit deep models together with corresponding methodologies, techniques, and methods, i.e. its matrices. Normal models directly reflect origins according to the focus used for shaping the scope and non-scope of the model, functions that the model play in application scenarios, and analogy to be used for representing the origins. Justification and the quality sufficiency characteristics are often injected by deep models. Moreover, complex applications remain to be complex also in the case that a model suite is going to be used.

Modern applications are also interdisciplinary and are develop by many professionals which follow very different postulates, paradigms, and principles. They are not entirely harmonisable. They co-exist however in daily life. This co-existence can be expressed on the level of models but not on the level of programming languages.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

Model suites will bond the applications, the domain and the computer support. They will become mediators between the application and the supporting infrastructure. Models are at the same time a means, an intermediary, and the medium for expressing separatable aspects of an application. Model suite will play the role of a "middle-range theory".

A model suite comes with its architecture and meta-model of the model suite. We may also use steering and governing models within a model suite. Some models in a model suite can be considered to be guiding ones since they are refined to more specific ones. A model suite can incorporate also quality-supporting meta-models (called checksum models) in order to provide means for quality control and for coherence support. Model suites incorporate also user-oriented interfacing models. We expect that these models will be developed as external and informative models for issues important for different users. In this case, models can be narrative as long as the association schema supports that. Synergetics approaches allow to develop master and slave models. Master models can be configured and adapted by control models. In this case, we need to build sophisticated editors for model suites.

The editors should be based on the same principles as the compilers for program generation from a given model suite. In the next subsection, we consider the onion meta-model for model suite composition. Editors should also include supporting means for check, control, and debugging.

Separation of concern can be based on application profiles [57]

normal model suites for the given application case

libraries of injectable deep model suites for landscaping and intrinsic strategic consideration steps

generic models and heritage model suites

from 3/4PL to 5PL by model suites representing thinking and reasoning

The MaaP approach will thus result in a complete model suite that becomes the source for the code of the problem solution, and for the system to be built. Figure 4

The delivered model (suite) is then going to be *compiled* to a program. The compiler has to transform the model to platform-dependent and directly executable programs after an optimisation phase. The compiler will typically be a 4-pass compiler (lexical analysis, semantical analysis, transformation and optimisation of intermediate code, target coding).

laymen programming

Von Neumann and Turing style of programming is only the beginning of a new era of development of computerised support mechanisms. This programming style does neither entirely reflect how humans reason and work nor represent the way how cultures, organisations, and societies act and evolve. We often map some kind of understanding to a program system and expect that the world outside computation is going to behave this way. Very soon after installation of such system it is going to be changed. Paradigms like programming-in-the-small and programming-in-the large can not be easily extended to programming-in-the-society or programming-in-the-world. The software and the data crises are

essentially a result of the endeavour to ortho-normalise and to conquer the world by means of computer systems.

The simplest approach to change this situation is Models_as_a_Program that reflects human way of working and thinking as well as the understanding of a society.

5.2 One Potential Solution: The Onion Meta-Model Specification Approach

MaaP can be based on a stereotyped model suite specification. This specification may follow the style of LaTeX document specification onion in Figure 5. One typical solution (however only one of many) for system and also model suite development is vertical layering: (1) specify the surface and foundation; (2) provide an understanding of mechanisms of functions, processes, and operations; (3) develop means that the system functions; (4) develop the basis for functioning; (5) develop variants for a solution within a variation spreadsheet with adaptation facilities. This approach supports development and application of well-structured and composable models suites which are governed by the kind of model.

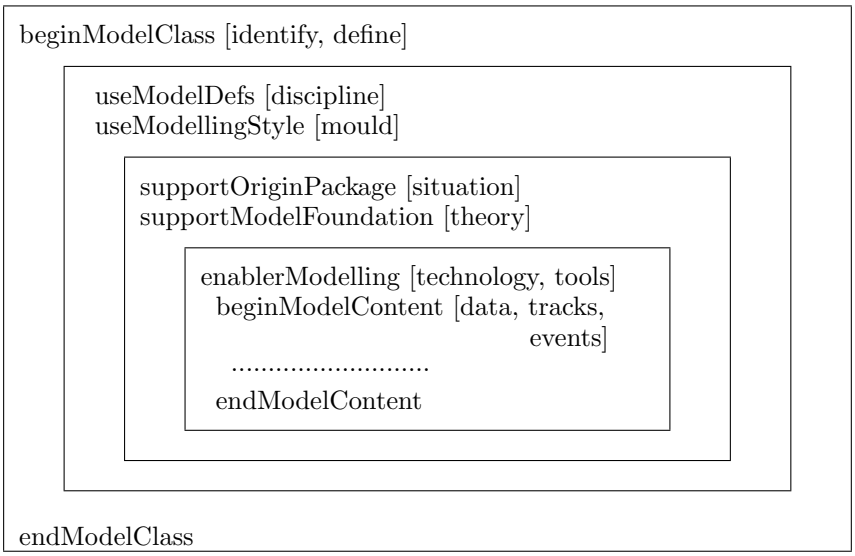


Fig. 5. The onion approach to model suite specification (modified from [99])

Model class: Models are used as instruments and thus depend on the scenario in which the function. Different kinds of models can be stereotyped. This

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

stereotypical categorisation can be used for the definition of the model class. The model class becomes the outer shell of the model specification onion. Model classes are based on the internal shell for model formating and general initialisation.

Model style and pattern: Depending on the model class, we may incorporate various libraries and methodological moulds at the model-style-and-discipline shell of the onion.

Model generics and foundations: Models consist of an internal deep model and of a normal model. The deep model is typically not completely revised. Its essential element is the model background. The grounding is accepted without any revision. The basis can be slightly extended and modified. The background forms the model foundation. The model foundation will be supported by a model package for representation of the specifics of the model situation. Additionally, generic models may be used as a basis for the normal model. These models may be collected in a library of generic models.

Model embedding and tools: The fourth shell is the support shell. Compilers are tools as well as a specification workbench. Technology support is a matter of convenience. A specific support is given for the combination of the given deep model with the normal model.

Normal model specification: The main part that is represents the specifics of the given set of origins is the normal model.

The specification setting follows the LaTeX compilation approach. The generation of the target model suite and the programs will result in a number of auxiliary elements similar to symbol tables for compilers of the first generation. The intermediate result of the transformation is a realisation-independent and infrastructure-independent model suite. The final result is then a realisation-dependent and infrastructure-dependent model suite or a program that may be executed in the given environment.

Pragmas and *model directives* are essential elements that we use for enhancement of conceptual models for system realisation. Pragmas can be considered as a language feature that allows adjusting or fine-tuning the behavior of a program compiled from a model. Model directives might also be used as additional control units for compilation. An essential element of a compiler is the *precompilation* based on a *prefetching* strategy of compiler-compilers.

Model correction is an essential element of prefetching. Already in the case of consistency maintenance for integrity constraints, we realised that models for translation must be correct. Schema-wide correctness is often neglected since most integrity constraints are declared at the local level. Cardinality constraints are a typical example which global correctness is partially based on local and type-based correctness. Since databases have to be finite and schemata are connected then we may derive implications for a given set of integrity constraints. We have to validate whether these implications are correct. Moreover, a set of cardinality constraints may be fulfilled only in infinite databases or in empty databases. Models must have a sufficient quality that is evaluated on the basis

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

of corresponding evaluation procedures. If a model or a model suite is not correct then we have to improve the quality of a model before translation.

This approach is already currently realisable. Let us consider in brief two case studies [50, 65, 74, 99]:

MaaP for Database System Design and Development

Database structure modelling often uses extended entity-relationship models such as HERM [94]. HERM can also be extended to HERM+ with a specific algebra for database functionality and view collections. It is already well-known how to translate an existing entity-relationship schema to a so-called logical (or implementation) schema. ER schemata can be enhanced by directives for this translation.

Rule-Based OR compilation of HERM schemata and models uses the theory of extended entity-relationship models. In the case of extended entity-relationship schemata and of VisualSQL as a query and functionality specification language, we may use a rule system consisting of 12+1 translation phases for transformation. The phases for a compilation are the following ones:

0. Configuration of the HERM compiler, preprocessing, prefetching according to the model directives;
1. Schema and operation lexical analysis;
2. Syntactic analysis for schema and operations;
3. Semantical analysis schema and operations;
4. Generation of intermediate code that is also used as the ground schema for VisualSQL query, view, maintenance specification;
5. Preparation for schema and operation optimisation (or normalisation);
6. Schema tuning (operational optimisation);
7. Introduction and injection of controlled redundancy;
8. Redefinition and revision of generated types and operations (also UDT);
9. Recompilation for quantity matrix (Mengengerüst) (big and huge DB);
10. Toughening for evolution and change in data dictionary;
11. Derivation of support services and view towers;
12. Generation of data dictionary entries.

We can enhance this translation to more specific compilation and embedding into the corresponding platforms. Practitioners use in this case pragmas at least at the intentional level for object-relational technology. The translation includes also translation of view collections. Furthermore, HERM+ can be extended by VisualSQL that allows to declare queries at the level of HERM schemata. This translation results then directly in a performance-oriented structure specification at the level of physical schemato. We may envision that this approach can also be extended to other platforms such as XML or big data platforms. The result

Figure 6 specialises the general Figure 5 for database modelling as sophisticated database programming for a sample application (booking and financing issues in business applications based on the global-as-design specification approach).

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

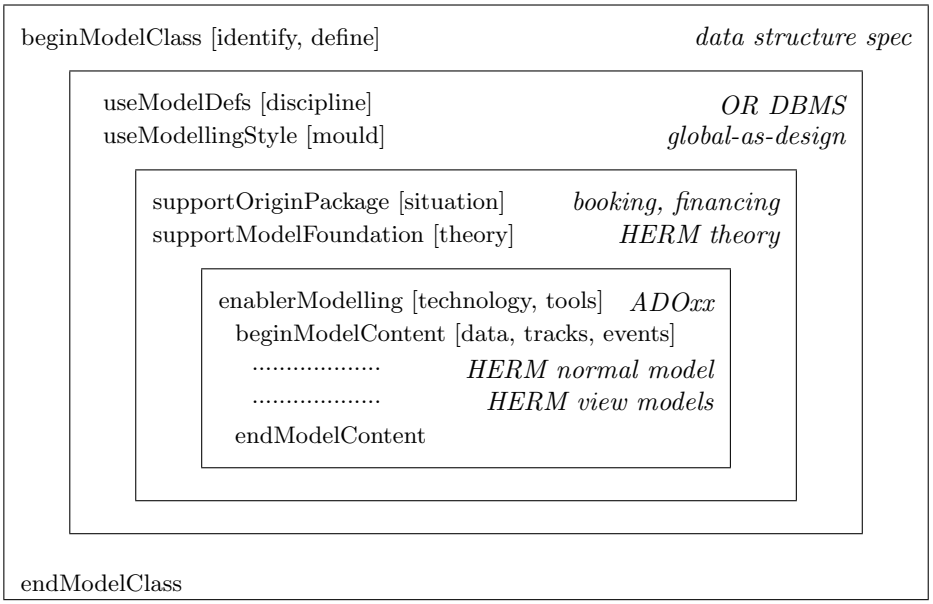


Fig. 6. The model-centric database structure development based on HERM+

MaaP for Workflow Derivation From Storyboards

The second case study is going to discuss solutions if the deep models are not similar. In this case, we either normalise the source models in such a way that they can be transformed to the target models or programs or generate the full variety of potential target models or programs. The last approach is feasible if the generated models or programs are not changed on their own. All changes to them must be changes at the level of the source model suite. Web information system development has already successfully used this approach. Normalisation of source models is driven by well-formedness rules that are applied to the models.

Storyboarding and BPMN-based workflow specification are based on different deep models, i.e. we observe a deep model mismatch. The differences are similar to those observed for impedance mismatches between parallel database processing and sequential program computation. Therefore, the model transformation needs also approaches to overcome this mismatch. BPMN is strictly actor-oriented and based on a strict local-as-design paradigm. Storyboarding is more flexible. We might use global-as-design or partial local-as-design combined with global-as-design techniques. Storyboarding provides some freedom on the flow of activities. BPMN mainly uses a more strict form where diagrams are given with a static flow of activities. Diagrams are not adapted to the changes in user behaviour. Dynamic workflow specification is still based on flexibility at design time and on stability during runtime.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

SiteLang specification allows to define scenes with plots for activities within a scene by different actors. This interaction among actors must be mapped to communication interaction between diagrams based on collaboration diagrams, to choreography diagrams among, or to conversations among actors. Scenes in a SiteLang specification can be visited by all enabled actors and completed by some of them. This freedom of task completion is neither achievable for normal BPMN diagrams nor for dynamic ones. Generic BPMN diagrams can however be used for adaptation to the actual actor behaviour at runtime.

The transformation of a storyboard can be based on language transformation rules. Typical rules are the following ones:

- An actor storyline is directly transferred to a BPMN pool.
- An atomic scene without atomic plots is transferred to a BPMN activity.
- A story sequence is represented by a sequence of BPMN activities. A well-formed story split with its own join (Fitch structure[1]) is transformed to the corresponding BPMN gateway structure. Optional scenes can be transformed to corresponding BPMN gateway structures. Iterations of simple stories can also be directly transferred to BPMN.
- Complex scenes are transformed to either BPMN diagrams or to complex activities or to sub-scenes.
- Story entries and completions are transferred to events in BPMN.
- Communication is based on BPMN communication pattern among diagrams, e.g. a link-scene-link combination among different actors.

The storyboard should also be normalised or transformed in some kind of well-formed storyboard. Parallel links between scenes are normalised either by introduction of intermediate scenes or by merging into a complex scene including plot transformation or link merging. The decision which next scenes are going to be chosen is integrated into the plot of the source scene. The naming of scenes is unified according to a naming scheme. Stories in a storyboard can be encapsulated into units with a singleton task. The storyboard is separated into mini-stories that will become workflows. The stories in a storyboard are decomposed into relatively independent mini-stories with starting and completing points. A mini-story must be representable by a singleton BPMN diagram if the actor in the mini-story does not change. Otherwise, we use superflow diagrams which call subflow diagrams as subprograms.

[74] introduced a small example for a storyboard of a trail system. The normalisation process leads directly to a diagram in Figure 7 that restricts the freedom and reduces the enabled actors to actors with encapsulatable behaviour within a mini-story.

There are several techniques and rules for storyboard conversion including data structure development:

Refinement preprocessing orients on data view mapping for each actor, on strict actor and role separation, on strict start-end flow, on session separation, and on additional communication.

[1] Each split must has its join in the diagram and each join has only its split what is a one-to-one association of splits and joins.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

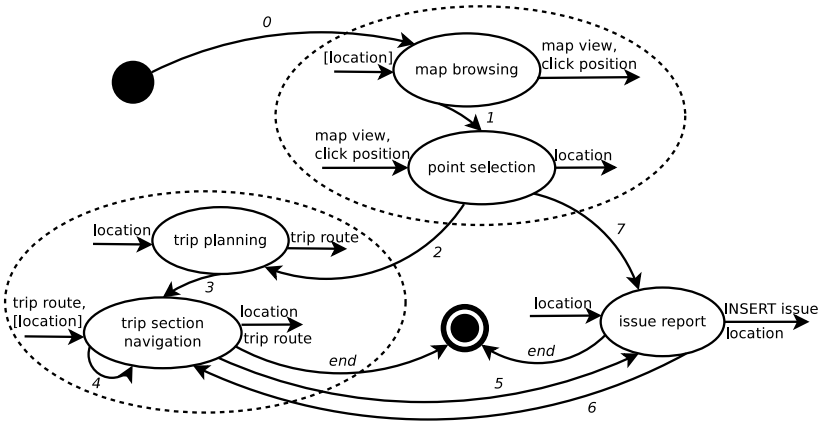


Fig. 7. The storyboard from [74] after refinement and normalization

Restriction of freedom is based on selection of the most essential flows since he storyboard that allows too much freedom. It downsizes the storyboard by restricting the flow of activities to the essential ones and by removing the rest.

Restricted and well-formed parallelism must yield to Fitch structuring.

Normalisation of the storyboard arrives at well-formed BPMN diagrams.

Plot integration for scenes with potential actor and role separation of these scenes into separate scenes with singleton actors.

We apply graph grammar rules to the stories since storyboarding uses a graphical language. The storyboard contains also the data viewpoint. This design information must be supported within a co-design approach to data structuring and workflow specification.

A normalised storyboard is now the basis for a BPMN diagram that displays only the visitors' viewpoint. The flow of activities is restricted to the most essential ones. Nothing else is supported. This transformation is information-loosing. A partial diagram after this transformation is displayed in Figure 8.

5.3 Towards Industrial Development On the Basis of Models

componentisation + standardisation + model suite architecture

6 M2P, MaP, MaaP: Summarising

The M2P, MaP, and MaaP approaches revise, combine, and generalise efforts and projects for model-driven development. We discussed some of those projects while knowing that is became already a major trend in software engineering.

We envision three already existing approaches:

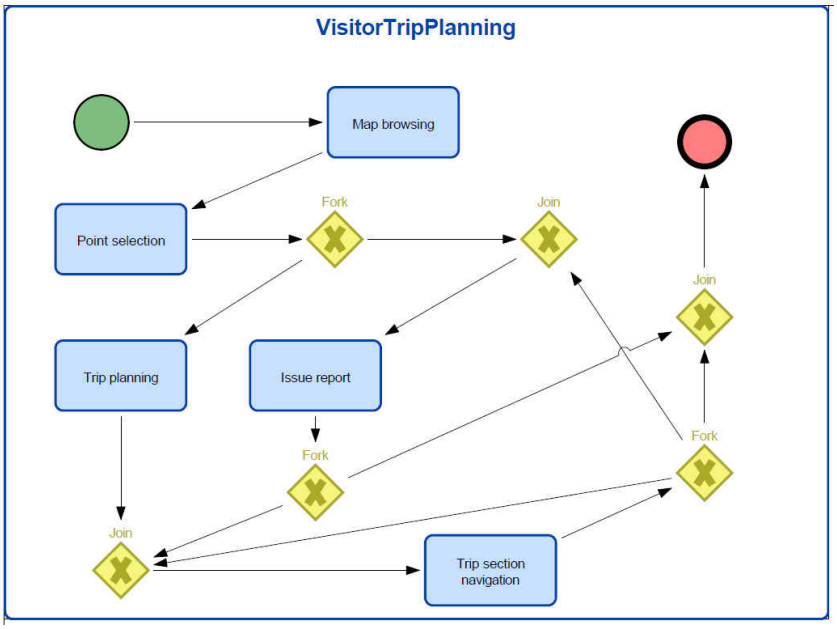


Fig. 8. Translation of the storyboard from [74] to some BPMN process

Finally we ask ourselves why we should target on true fifth generation programming. A number of reasons force us to use next generation programming:

- Computer engineering has not yet reached the level of being a science. It has not got its culture. Education is mainly education for handicraft work.
- Programming is more and more often performed by laymen, outsiders, casual users, non-specialists, self-made programmers, etc. They need a proper support.
- Apps and modern facilities are often developed without CS education. The uncontrolled growth worsens this situation.
- Users have not been the main concern for programmers. Operator and administrator thinking is still THE common sense understanding of the area.
- Migration, integration, maintenance, and modernisation are still a real nightmare. Half-life time of systems and software is far lower than lifespan of software deployment. New systems use new paradigms without downgrading features.
- Documentation is often luxury. Documentations are often generated from code (without inner documentation, without thoughtful architecture, with "monster" classes, only once and never after modification) because of it is otherwise not economic, it seems to be side work, it disturbs delivery in time, and there is no urgent need in better ones. Modernisation and maintenance becomes then a real vexation.

Second International Workshop, M2P, 2020
CCIS 1401, 3--44

We finally complete this paper with a citation to a Genicore poster presented at ER'2017: "Modelware is the new software", i.e. domain-situation models instead of models of software. [51]

References

1. K. Adam, L. Netz, S. Varga, J. Michael, B. Rumpe, P. Heuser, and P. Letmathe. Model-based generation of enterprise information systems. *EMISA Forum*, 38(1):56–60, 2018.
2. H. Aiso. The fifth generation computer systems project. *Future Generation Comp. Syst.*, 4(3):159–175, 1988.
3. H. Aiso. 25 years of MITI and its influence on computing research in Japan. *IEEE Computer*, 24(9):99–100, 1991.
4. H. Aiso. Discussion on achievements, challenges, solutions, and problems of the 5th generation PL project, Nov. 13 2016. Evening with B. Thalheim and colleagues from Keio University, Yokohama, Japan.
5. B. AlBdaiwi, R. Noack, and B. Thalheim. Database structure modelling by stereotypes, pattern and templates. In *Proc. EJC 2014*, volume 2014/4 of *KCSS*, pages 1–19, Kiel, Germany, June 2014. Department of Computer Science, Faculty of Engineering.
6. B. AlBdaiwi, R. Noack, and B. Thalheim. Pattern-based conceptual data modelling. In *Information Modelling and Knowledge Bases*, volume XXVI of *Frontiers in Artificial Intelligence and Applications, 272*, pages 1–20. IOS Press, 2014.
7. A. Albrecht, M. Altus, E. Buchholz, A. Düsterhöft, and B. Thalheim. The rapid application and database development (RADD) workbench - A comfortable database design tool. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *CAiSE*, volume 932 of *Lecture Notes in Computer Science*, pages 327–340. Springer, 1995.
8. L. Baresi, F. Garzotto, and P. Paolini. Extending UML for modeling web applications. In *HICSS*, 2001.
9. J. Becker and P Delfmann, editors. *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*. Springer, 2007.
10. A. Berztiss and B. Thalheim. Exceptions in information systems. In *Digital Libaries: Advanced Methods and Technologies, RCDL 2007*, pages 284–295, 2007.
11. A. Bienemann, K.-D. Schewe, and B. Thalheim. Towards a theory of genericity based on government and binding. In *Proc. ER'06, LNCS 4215*, pages 311–324. Springer, 2006.
12. E. Börger. A practice-oriented course on principles of computation. programming and systems design and analysis. In *CoLogNet / Formal Methods europe Symposium TFM'04*, Gent, 2004.
13. E. Börger and A. Raschke. *Modeling Companion for Software Practitioners*. Springer, 2018.
14. E. Börger and O. Sörensen. BPMN core modeling concepts: Inheritance-based execution semantics. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 9, pages 287–334. Springer, Berlin, 2011.
15. E. Börger and R. Stärk. *Abstract state machines - A method for high-level system design and analysis*. Springer, Berlin, 2003.
16. E. Börger and B. Thalheim. A method for verifiable and validatable business process modeling. In *Software Engineering*, LNCS 5316, pages 59 – 115. Springer, 2008.

17. J. Bormann and J. Lötzsch. *Definition und Realisierung von Fachsprachen mit DEPOT*. PhD thesis, Technische Universität Dresden, Sektion Mathematik, 1974.
18. M. Brambilla, S. Comai, P. Fraternali, and M. Matera. Designing web applications with WebML and WebRatio. pages 221–261, 2008.
19. G. Brassard and P. Bratley. *Algorithmics - Theory and Practice*. Prentice Hall, London, 1988.
20. J.E. Brenner. The logical process of model-based reasoning. In L. Magnani, W. Carnielli, and C. Pizzi, editors, *Model-based reasoning in science and technology*, pages 333–358. Springer, Heidelberg, 2010.
21. J. Conallen. *Building Web Applications with UML*. Addison-Wesley, Boston, 2003.
22. M.T. Cox and A. Raja, editors. *Mwetareasoning - Thinking about Thinking*. MIT Press, Cambridge, 2011.
23. A. Dahanayake. *An environment to support flexible information modelling*. PhD thesis, Delft University of Technology, 1997.
24. A. Dahanayake and B. Thalheim. Co-evolution of (information) system models. In *EMMSAD 2010*, volume 50 of *LNBIP*, pages 314–326. Springer, 2010.
25. T. Daly. Axiom. the scientific computation system. http://axiom-developer.org/axiom-website/, 2018.
26. P Deransart, M Jourdan, and B Lorho. *Attribute Grammars-Definitions, Systems and Bibliography*. LNCS 323. Springer Verlag, 1988.
27. D. Draheim and G. Weber. *Form-Oriented Analysis*. Springer, Berlin, 2005.
28. Eclipse project web site. http://www.eclipse.org.
29. M. Edwards. A brief history of holons. *Unpublished essay. http://www. integralworld. net/edwards13. html. Published*, 2003.
30. H. Ehrig, C. Ermel, U. Golas, and F. Hermann. *Graph and Model Transformation - General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2015.
31. D. W. Embley, S. W. Liddle, and O. Pastor. Conceptual-model programming: A manifesto. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 1, pages 1–15. Springer, Berlin, 2010.
32. D. W. Embley and W.Y. Mok. Mapping conceptual models to database schemas. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 5, pages 123–164. Springer, Berlin, 2010.
33. A. P. Ershov. The transformational machine: Theme and variations. In *Proc. MFCS 1981*, volume 118 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 1981.
34. E. A. Feigenbaum and P. McCorduck. *The fifth generation - artificial intelligence and Japan's computer challenge to the world*. Addison-Wesley, 1983.
35. P. Fettke and P. Loos, editors. *Reference Modeling for Business Systems Analysis*. Hershey, 2007.
36. G. Fiedler, H. Jaakkola, T. Mäkinen, B. Thalheim, and T. Varkoi. Application domain engineering for web information systems supported by SPICE. In *Proc. SPICE'07*, Seoul, Korea, May 2007. IOS Press.
37. R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-driven development using uml 2.0: promises and pitfalls. *Computer*, 39(2):59–66, 2006.
38. U. Frank. Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering*, 6(6):319–337, 2014.
39. F. Garzotto, P. Paolini, and D. Schwabe. Hdm - a model-based approach to hypertext application design. *ACM ToIS*, 11(1):1–26, 1993.

40. D. Q. Goldin, S. Srinivasa, and B. Thalheim. Is = dbs + interaction: Towards principles of information system design. In *Proc. ER 2000*, volume 1920 of *Lecture Notes in Computer Science*, pages 140–153. Springer, 2000.
41. R. Grossmann, J. Hutschenreiter, J. Lampe, J. Lötzsch, and K. Mager. DEPOT 2a Metasystem für die Analyse und Verarbeitung verbundener Fachsprachen. Technical Report 85, Studientexte des WBZ MKR/Informationsverarbeitung der TU Dresden, Dresden, 1985.
42. D. Harel. *Algorithmics: The Spirit of Computing*. Addison-Wesley, Reading, Massachusetts, 1987.
43. D. Harel and R. Marelly. *Come, Let's play: Scenario-based programming using LSCs and the play-engine*. Springer, Berlin, 2003.
44. K. Hölldobler, N. Jansen, B. Rumpe, and A. Wortmann. Komposition Domänenspezifischer Sprachen unter Nutzung der MontiCore Language Workbench, am Beispiel SysML 2. In *Proc. Modellierung 2020*, volume P-302 of *LNI*, pages 189–190. Gesellschaft für Informatik e.V., 2020.
45. G.-J. Houben, K. van der Sluijs, P. Barna, J. Broekstra, S. Casteleyn, Z. Fiala, and F. Frasincar. HERA. pages 263–301, 2008.
46. J. Hutschenreiter. *Zur Pragmatik von Fachsprachen*. PhD thesis, Technische Universität Dresden, Sektion Mathematik, 1986.
47. T. Isakowitz, E.A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, 1995.
48. H. Jaakkola, T. Mäkinen, B. Thalheim, and T. Varkoi. Evolving the database co-design framework by SPICE. In *Informaton Modelling and Knowledge Bases Vol. XVII, Series Frontiers in Arificial Intelligence, volume 136*, pages 268–279. IOS Press, May 2006.
49. H. Jaakkola and B. Thalheim. Visual SQL - high-quality er-based query treatment. In *IWCMQ'2003*, LNCS 2814, pages 129–139. Springer, 2003.
50. H. Jaakkola and B. Thalheim. Model-based fifth generation programming. In *Information Modelling and Knowledge Bases Vol. XXXI*, Frontiers in Artificial Intelligence and Applications, 312, pages 381–400. IOS Press, 2020.
51. T. Jonsson and H. Enquist. Semantic consistency in enterprise models - through seamless modelling and execution support. In *Proc. ER Forum 2017 and ER 2017 Demo Track*, volume 1979 of *CEUR Workshop Proceedings*, pages 343–346. CEUR-WS.org, 2017.
52. E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.
53. D. Karagiannis, H. C. Mayr, and J. Mylopoulos, editors. *Domain-Specific Conceptual Modeling, Concepts, Methods and Tools*. Springer, 2016.
54. S. Kelly, M. Rossi, and J.-P. Tolvanen. What is needed in a metacase environment? *Enterprise Modelling and Information Systems Architectures*, 1(1):25–35, 2005.
55. S. Kelly and K. Smolander. Evolution and issues in MetaCASE. *Information & Software Technology*, 38(4):261–266, 1996.
56. Website Kieler. Kiel Integrated Environment for Layout Eclipse Rich Client. https://www.rtsys.informatik.uni-kiel.de/en/research/kieler, 2018. Accessed July 29, 2018.
57. Y. Kiyoki and B. Thalheim. Analysis-driven data collection, integration and preparation for visualisation. In *Information Modelling and Knowledge Bases*, volume XXIV, pages 142–160. IOS Press, 2013.

58. M. Klettke and B. Thalheim. Evolution and migration of information systems. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 12, pages 381–420. Springer, Berlin, 2011.
59. D. E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986.
60. D. E. Knuth. *Literate Programming*. Number 27 in CSLI Lecture Notes. Center for the Study of Language and Information at Stanford/ California, 1992.
61. N. Koch, A. Knapp, G. Zhang, and H. Baumeister. UML-based web engineering. pages 157–191, 2008.
62. D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige. Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software and System Modeling*, 16(1):229–255, 2017.
63. F. Kramer and B. Thalheim. Holistic conceptual and logical database structure modelling with adoxx. In D. Karagiannis, H.C. Mayr, and J. Mylopoulos, editors, *Domain-specific conceptual model*, pages 269–290, Cham, 2016. Springer.
64. F.F. Kramer. *Ein allgemeiner Ansatz zur Metadaten-Verwaltung*. PhD thesis, Christian-Albrechts University of Kiel, Technical Faculty, Kiel, 2018.
65. Y. Kropp and B. Thalheim. Model-based interface generation. In *Proc. 29'th EJC*, pages 70–87, Lappeenranta, Finland, 2019. LUT, Finland.
66. G. Lakoff. *Women, fire, and dangerous things - What categories reveal about the mind*. The University of Chicago Press, Chicago, 1987.
67. L. Lamport. *LaTeX: A document preparation system*. Addison-Wesley, 1994.
68. J. Lötzsch. *Metasprachlich gestützte Verarbeitung ebener Fachsprachen*. PhD thesis, Dresden University of Technology, Germany, 1982.
69. L. Magnani, W. Carnielli, and C. Pizzi, editors. *Model-Based Reasoning in Science and Technology: Abduction, Logic, and Computational Discovery*. Springer, 2010.
70. B. Mahr. Information science and the logic of models. *Software and Systems Modeling*, 8(3):365–383, 2009.
71. D. Marco and M. Jennings. *Universal meta data models*. Wiley Publ. Inc., 2004.
72. H. C. Mayr, J. Michael, S. Ranasinghe, V. A. Shekhovtsov, and C. Steinberger. Model centered architecture. In *Conceptual Modeling Perspectives.*, pages 85–104. Springer, 2017.
73. S. Meliá and J. Gómez. The WebSA approach: Applying model driven engineering to web applications. *Journal of Web Engineering*, 5(2):121–149, 2006.
74. A. Molnar and B. Thalheim. Usage models mapped to programs. In *Proc. M2P – New Trends in Database and Information Systems*, Bled, 2019. Springer, CCIS 1064.
75. T. Moto-oka, editor. *Fifth generation computer systems.* North-Holland, Amsterdam, 1982.
76. P.-A. Muller, P. Studer, F. Fondement, and J. Bézivin. Platform independent web application modeling and development with Netsilon. *Software and System Modeling*, 4(4):424–442, 2005.
77. N. J. Nersessian. *Creating Scientific Concepts*. MIT Press, 2008.
78. K. Noack. Technologische und methodische Grundlagen von SCOPELAND. White paper, www.scopeland.de, 2009.
79. S. E. Page. *The model thinker – Waht you need to know to make data work for you*. Basic Books, New York, 2018.
80. O. Pastor, S.M. Abrahão, and J. Fons. An object-oriented approach to automate web applications development. In *Electronic Commerce and Web Technologies, Second International Conference, EC-Web 2001 Munich, Germany, September 4-6, 2001, Proceedings*, volume 2115 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2001.

81. Oscar Pastor and Juan Carlos Molina. *Model-driven architecture in practice - a software production environment based on conceptual modeling*. Springer, 2007.
82. T. Pittman and J. Peters. *The Art of Compiler Design: Theory and Practice*. Prentice Hall, Upper Saddle River, 1992.
83. A.S. Podkolsin. *Computer-based modelling of solution processes for mathematical tasks (in Russian)*. ZPI at Mech-Mat MGU, Moscov, 2001.
84. Website PtolemyII. Ptolemy project: heterogeneous modelling and design. http://ptolemy.berkeley.edu/ptolemyII/, 2018. Accessed July 29, 2018.
85. D. Di Ruscio, H. Muccini, and A. Pierantonio. A data-modelling approach to web application synthesis. *Int. J. Web Eng. Technol.*, 1(3):320–337, 2004.
86. K.-D. Schewe and B. Thalheim. Co-design of web information systems. Texts & Monographs in Symbolic Computation, pages 293–332, Wien, 2015. Springer.
87. K.-D. Schewe and B. Thalheim. *Design and development of web information systems*. Springer, Chur, 2019.
88. J. W. Schmidt and F. Matthes. The DBPL project: Advances in modular database programming. *Inf. Syst.*, 19(2):121–140, 1994.
89. D. Schwabe and G. Rossi. The object-oriented hypermedia design model. *Communications of the ACM*, 38(8):45–46, 1995.
90. D. E. Shasha and P. Bonnet. *Database Tuning - Principles, Experiments, and Troubleshooting Techniques*. Elsevier, 2002.
91. L. Silverston. *The data model resource book. Revised edition*, volume 2. Wiley, 2001.
92. M. Steeg. *RADD/raddstar - A rule-based database schema compiler, evaluator, and optimizer*. PhD thesis, BTU Cottbus, Computer Science Institute, Cottbus, October 2000.
93. S. T. Taft, R. A. Duff, R. Brukardt, E. Plödereder, P. Leroy, and E. Schonberg. *Ada 2012 Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/2012 (E)*, volume 8339 of *Lecture Notes in Computer Science*. Springer, 2013.
94. B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.
95. B. Thalheim. Codesign of structuring, functionality, distribution and interactivity. *Australian Computer Science Comm.*, 31(6):3–12, 2004. Proc. APCCM'2004.
96. B. Thalheim. The theory of conceptual models, the theory of conceptual modelling and foundations of conceptual modelling. In *The Handbook of Conceptual Modeling: Its Usage and Its Challenges*, chapter 17, pages 547–580. Springer, Berlin, 2011.
97. B. Thalheim and A. Dahanayake. Comprehending a service by informative models. *T. Large-Scale Data- and Knowledge-Centered Systems*, 30:87–108, 2016.
98. B. Thalheim and I. Nissen, editors. *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston, 2015.
99. B. Thalheim, A. Sotnikov, and I. Fiodorov. Models: The main tool of true fifth generation programming. In *Proc. EEKM 2019 – Enterprise Engineering and Knowledge Management*, pages 161–170, Moscov, 2019. CEUR workshop poceedings, Vol. 2413.
100. B. Thalheim, M. Tropmann-Frick, and T. Ziebermayr. Application of generic workflows for disaster management. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 64–81. IOS Press, 2014.

101. S. Torge, W. Esswein, S. Lehrmann, and B. Thalheim. Categories for description of reference models. In *Information Modelling and Knowledge Bases*, volume XXV of *Frontiers in Artificial Intelligence and Applications, 260*, pages 229–240. IOS Press, 2014.
102. M. Tropmann, B. Thalheim, and R. Korff. Performance forecasting. *21. GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)*, pages 9–13, 2009.
103. M. Tropmann-Frick. *Genericity in Process-Aware Information Systems*. PhD thesis, Christian-Albrechts University of Kiel, Technical Faculty, Kiel, 2016.
104. O. De Troyer, S. Casteleyn, and P. Plessers. WSDM: Web semantics design method. pages 303–351, 2008.
105. K. Ueda. Logic/constraint programming and concurrency: The hard-won lessons of the fifth generation computer project. *Sci. Comput. Program.*, 164:3–17, 2018.
106. R. P. van de Riet. An overview and appraisal of the fifth generation computer system project. *Future Generation Comp. Syst.*, 9(2):83–103, 1993.
107. B. Vela, C.J. Acu na, and E. Marcos. A model driven approach to XML database development. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)*, pages 273–285, Shanghai, China, November 2004.
108. B. F. Webster. *Pitfalls of object-oriented development: a guide for the wary and entusiastic*. M&T books, New York, 1995.
109. P. Wegner and D. Q. Goldin. Computation beyond Turing machines. *Commun. ACM*, 46(4):100–102, 2003.
110. R. Wilhelm, H. Seidl, and S. Hack. *Compiler Design - Syntactic and Semantic Analysis*. Springer, 2013.
111. Li Xu and D. W. Embley. Combining the best of global-as-view and local-as-view for data integration. In *Proc. ISTA'2004*, volume P-48 of *LNI*, pages 123–136. GI, 2004.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

Models versus Model Descriptions

Joachim Fischer[1][0000−0003−2476−3996], Birger Møller-Pedersen[2][0000−0003−2123−3260], Andreas Prinz[3][0000−0002−0646−2877], and Bernhard Thalheim[4][0000−0002−7909−7786]

[1] Department of Computer Science, Humboldt University, Berlin, Germany
fischer@informatik.hu-berlin.de
[2] Department of Informatics, University of Oslo, Oslo, Norway birger@ifi.uio.no
[3] Department of ICT, University of Agder, Grimstad, Norway
andreas.prinz@uia.no
[4] Department of Computer Science, University Kiel, Germany
bernhard.thalheim@email.uni-kiel.de

Abstract. In the development of computer-based systems, modelling is often advocated in addition to programming, in that it helps in reflecting the application domain and that it makes the design and experiment activities of development more efficient. However, there is disagreement about what models are and how they can be used in software systems development. In this paper, we present the Scandinavian approach to modelling, which makes a clear distinction between models and model descriptions. This paper explains the connections between models, descriptions, systems, and executions. Combining the Scandinavian approach with the Kiel notion of model, we establish that both descriptions and executions are closely connected instruments with different roles. The paper argues that (program) executions are the models of dynamic systems, not their descriptions in terms of diagrams and text. So in a general sense programming is about writing descriptions for systems. In particular the paper clarifies when programming is also modelling.

Keywords: model · system · description · execution · semantics.

1 Introduction

The development of computer-based systems brings together different areas of experience, methods and terminology. Therefore, it is not surprising that terms have different meanings in different contexts. Surprisingly, even very basic terms such as system, model, model description, modeling and programming are affected. In the development of computer-based systems, three methodologically different computer-related disciplines meet. First, there are engineers who use computer models to design new technical systems, simulate and test them before actually building them. Second, there are IT experts who design software systems from abstract models by means of extensive transformations. Third, there are other IT experts who design and implement systems directly using specific

Second International Workshop, M2P
2020, CCIS 1401, 67-89

programming languages and techniques. Each of these three groups use programming and modelling, and all of them have slightly different understanding of what a model is. The discrepancy in the conceptual perception remains with modellers and programmers even for development of pure software systems.

As part of system development, *programmers* produce running systems. They do this by programming, that is writing programs. Often, they do not subscribe to the idea of modelling as this typically implies the creation of diagrams that do not contribute to the making of programs and quickly become obsolete.

Modellers handle different kinds of *models* (domain models, requirements and design models) in a mixture of diagrams and text. It can be argued that programmers are to a certain extent also modelling, and modellers are programming. We will look at these two activities and explain how they are similar and how they are different. In this paper, we present the Scandinavian approach to modelling as a shared understanding between modellers and programmers using an understanding of models as illustrated in Figure 1.

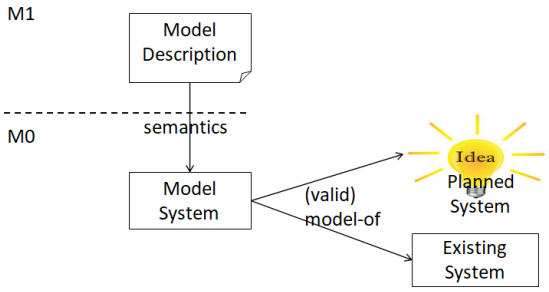


Fig. 1. Overview of the Scandinavian Approach to Modelling

The approach presented in this paper is called the Scandinavian Approach to Modelling. It was started in the design of the language SIMULA [Dahl and Nygaard, 1965] and [Dahl and Nygaard, 1966], and developed further with the languages Delta [Holbæk-Hanssen et al., 1973], Beta [Madsen et al., 1993], and SDL-2000 [Union, 2011]. Moreover, there are several articles describing aspects of the Scandinavian approach, see [Nygaard and Dahl, 1978], [Fischer et al., 2016], [Madsen and Møller-Pedersen, 2018], [Gjøsæter et al., 2016], [Madsen and Møller-Pedersen, 2010], [Prinz et al., 2016], and [Scheidgen and Fischer, 2007].

The paper evolves around the concept of a system, see the bottom of Figure 1. This is quite natural as systems development is about producing systems. Systems can be existing physical systems, or imagined and planned systems. Systems can be made using various kinds of *descriptions*, e.g. *programs*. These descriptions imply executions, i.e. systems. There is a dotted line in Figure 1 between the world of systems, called M0 (below the line) and the world of descriptions, called M1 (above the line).

Second International Workshop, M2P
2020, CCIS 1401, 67-89

The Scandinavian approach considers a *model* to be a (model) system, which is a model of another system called *referent* system. This referent system is an existing system in case of simulation or a planned system in case of system development. The (model) system is created by a (model) description, which is the diagram or code that describes the (model) system.

In this paper, we will focus on dynamic models. This restricted view on models helps to keep the discussion focused. Still, this is the prominent use of models in computer science and in system development. We combine the Scandinavian approach with the Kiel notion of model [Thalheim et al., 2015], which also provides a general definition of the term model.

The distinction between models and model descriptions has been made before in [Madsen and Møller-Pedersen, 2018] and in [Fischer. et al., 2020]. Here we apply the argument to the MOF architecture [Kleppe and Warmer, 2003] (see Figure 2), and we conclude that models are systems at level M0.

M3	meta-language	Example: MOF
M2	language	Example: UML
M1	model description	Example: UML diagrams for library system
M0	model execution	Example: running library system

Fig. 2. OMG four level architecture

The paper is structured as follows. In Section 2, we discuss the notion of systems as a starting point for the discussion. This is extended in Section 3 with the role that descriptions have in creating systems. Then we continue with discussing the notion of models in Section 4. We bring all these parts together in Section 5 and compare with the Kiel notion of model. We discuss the Scandinavian approach in Section 6, and summarize in Section 7.

2 Systems

Modelling and programming are used in system development, so we start by defining the concept 'system'. Interestingly, although UML claims to be a language for the modelling of systems, it does not define what a system is. In general, programming languages do not define the term system, either. However, the UML standard mentions 'running system' when talking about interactions. Thus the UML idea of systems is running systems.

System development is about making dynamic systems, i.e. systems that inherently change over time. Figure 3 shows a sample system: a room with a control system for cooling and heating. Systems are composed of parts, and the interacting parts are changing, thus bringing about the system state changes[5]. We call the system state changes the *behaviour* of the system.

[5] The state changes can be continuous or discrete. We call the systems discrete versus continuous systems, or combined systems, if both kinds of state changes appear.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

Fig. 3. Systems

The parts of a system can be existing entities, like a chair in the room, and they can be planned (imagined) parts, for example the controller of the heating system. We apply the perspective of object-oriented modeling and programming, so the parts of a system are objects. The parts of a system can be systems themselves.

The system structure itself can be static or subject to dynamic changes. Of course, there is the extreme case where a system only has existing parts, for example when we create a model in order to *understand* reality. Another extreme case is a system that only contains planned parts, most often in the process of creating something new. This brings us to our definition of system, compiled mainly from [Fischer et al., 2016] and [Bossel, 2007].

Definition 1 (System). *A system is a purposeful collection of executing objects interacting with each other and with entities in the environment. It is a whole and loses its identity if essential objects are missing. A system has structure (existing and imagined objects having properties) and behaviour (executions by means of object behaviour).*

This way, a system is a set of possible executions, i.e. a set of object configurations that exist at different points in time. It has purpose, identity, structure, behaviour, and interaction with the environment.

When we want to work with a system, it is important to know what we can observe. This is given by the system *state*, which in turn is based upon the system *structure*. The structure of a system is a dynamic collection of interacting objects, each with their properties that contribute to the state of the system. With the state we can observe the system by capturing snapshots of its progression.

Systems are just some part of reality. Typically, they have *interaction* with other parts of reality, which we call *environment*. If they do, we call them open (non-autonomous) systems. If systems can exist on their own, then they are closed systems. In many cases, a system with only planned parts is an open system. In this case, it is possible to close the open system by introducing an abstract part for the environment of the system, which is an existing part. In this paper, we are mostly talking about open systems, where the system reads inputs from the environment and writes outputs to the environment.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

By definition 1, not all parts of reality are systems; we have to consider a given part of reality as a system, and we abstract away unimportant parts and features of reality. There are two abstraction levels. First, we have a discourse, i.e. a way we look at reality in general, see Figure 4. The discourse starts with our understanding how reality is composed in general. For our considerations and for this paper, we consider reality to be composed of objects, following an object-oriented discourse.

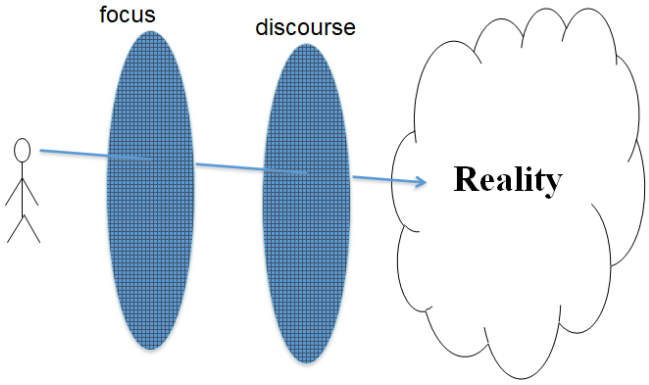


Fig. 4. Viewpoints and Discourse

In addition to the discourse, the *purpose* of the system will add another level of abstraction. This is our focus when looking at the system, and it reduces the aspects to look at. The set of five ODP-RM viewpoints [ITU-T, 1995] is an example of a high-level classification of the purpose of systems for a special class of complex system.

System identification always requires an abstraction of reality, which depends on the purpose of the system. This means that, with a completely identical system purpose, different systems can be created as object configurations. For example, the content of a boiler can be measured by the filled volume or by the water level.

All in all, we conclude that systems are part of reality (really existing or hypothetically assumed), and this means they are placed on OMG level M0.

3 System Descriptions

After we have clarified the notion of system, we will ask ourselves where the systems come from. In many cases, systems are created by making descriptions of them. In this section, we discuss these system descriptions.

2020, CCIS 1401, 67-89

3.1 Systems and Descriptions

Many systems are based on system descriptions, see Figure 5. A typical example

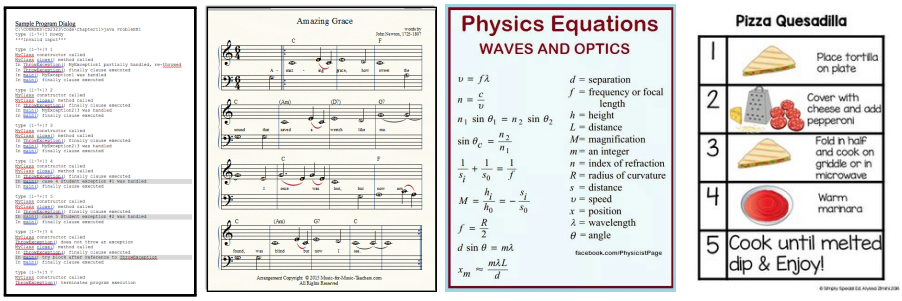


Fig. 5. Different system descriptions

is a system that is based on a program in some programming language. The program is the description, and the description leads to a running system on a real hardware. A similar example is a music sheet that describes, i.e. is a description of, a musical performance.

The system described by the description can also be an imaginary system, as for example given by physics formulae. Their meaning creates a virtual world, i.e. a virtual system. In a similar sense, a recipe (description) creates an imaginary sequence of steps for a dish that can be used to create the real dish. In all these cases, the various kinds of *system descriptions* result in running systems, see Figure 6. As discussed before, the running systems consist of executing objects.

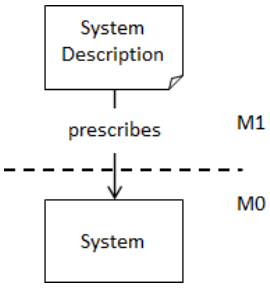


Fig. 6. Prescriptions

All these examples show that the description is not a system in itself: it is not composed of objects and it does not have behaviour. The system related to a description is implied by the semantics of the language of the description and thereby the meaning of the description. It has become customary to call this

Second International Workshop, M2P
2020, CCIS 1401, 67-89

connection between the description and the system for "prescribe", and then the description is rather a prescription, see again Figure 6. A similar approach is used in [Seidewitz, 2003]. This allows us to define programming.

Definition 2 (Programming). *Programming is the activity to produce descriptions (prescriptions) in order to produce systems. In a narrow sense, programming is concerned with prescriptions in the form of computer programs. In the general sense, programming means to construct a description in order to create a system, as designing recipes, composing music, or inventing physics formulas.*

There is a vast body of knowledge as to which steps to take in order to create good descriptions, and computer engineering explain how to do this for computer systems. In this article, we are not concerned how the description comes about, but it is essential to agree that the description is created in order to create a system.

Coming back to the OMG architecture, we notice that a description is placed on level M1, independent of the kind of description. The description then implies a number of possible executions consisting of a changing structure of objects according to the prescription. These objects, i.e. the implied system, will be on M0. The objects behave according to their prescription of behaviour as part of the description on M1.

This approach divides the world into descriptions and objects, as shown in Figure 7. Below the line, on M0, there are objects, there are states of objects,

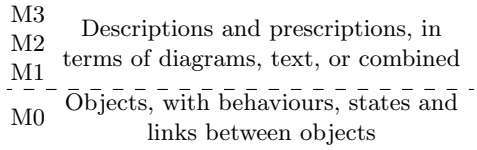


Fig. 7. Descriptions and Objects

and there are state changes resulting from the behaviour of the objects.

Above the line, on M1[6], there are *no* objects, there are *no* states, and there are *no* state changes. However, there are *prescriptions of objects* (e.g. by means of classes), there are *prescriptions of states* (e.g. descriptions of attributes of objects), and there are *prescriptions of state changes* (e.g. assignments).

In the following, we will use the term description for both description and prescription, except in cases where it is essential to distinguish. The main difference is between objects (on M0) and descriptions of objects (on M1-M3).

[6] There are also descriptions and prescriptions on M2 and M3, but this is out of the scope of this paper.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

3.2 Semantics and Meaning

As already stated in [Fischer et al., 2016], it is the semantics of the language of the description defining the system belonging to the system description. Semantics is not about formality in the first place, even though system engineering is most interested in formal description languages and programming languages with a precise semantics.

In the OMG architecture, this is the move from descriptions on level M1 to objects and executions on level M0. This way, semantics is a *vertical* relation (crossing levels). In reality, there might be several transformations (horizontal semantic steps) before the vertical step appears. In the world of computers, the most common way to provide semantics is using a compiler, which is essentially replacing one description by another, more executable description.

Still, at the end we need to come to a description that can be executed, i.e. that can cross the level. We define the idea of semantics as follows.

Definition 3 (Semantics). *Semantics is the relation between a (system) description and its prescribed possible executions (the system)*[7]*. In our context, semantics is the same as the prescribe relation.*

The semantics of a specification is given by the language it is written in; it is not a property of the specification itself. The semantics as given by the language (on level M2) connects the system specification on level M1 to the implied system on level M0.

In this consideration, it is irrelevant whether the semantics is formal or not. The important point is that there is a system implied by the description. A programming language provides more formality than a music sheet, where the conductor can add some interpretation. In both cases, the description has a meaning.

Semantics provides two parts: structural semantics and dynamic semantics. *Structural semantics* details which structure the system implied by the description has, i.e. which objects and which properties it has. Each system state is then characterized by this same structure.

There will be different realizations of these system structures, depending on the underlying reality that is used to run the system. For example, there will be different ways to represent Java objects depending on the underlying machine, but on the abstraction level of the execution, these are the same. This is given by the abstraction property of system as discussed in Figure 4. In a similar sense, a musical sheet will have different realizations depending on the musical instrument it is played on.

[7] The DELTA language report used the neutral term 'generator' that generates a system based upon a system description, i.e. provides the vertical relation. A generator could be a machine or a human being, or a mixture. In MDA, a generator is most often understood as a tool generating a new low-level description out of the original high-level description. This would amount to a horizontal generation and is not what semantics is about here.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

In our object-oriented discourse, we can observe the system behaviour using *snapshots*, i.e. collections of objects with the values of their properties. Such a snapshot is a complete description of the current state of the execution of the system as defined by the execution semantics[8]. A snapshot includes information about all relevant runtime objects.

Experiments with systems, like testing and simulations, are experiments with an execution, not with descriptions. In [Exner et al., 2014], it is well documented that even prototyping is experimenting with systems in varying degrees of completeness, not experimenting with descriptions.

4 Models

After discussing systems and (system) descriptions, we are prepared to look into the definition of models. The UML standard uses the following definition [OMG, 2017]. Note the distinction between existing and planned systems (parts).

> A model is always a model of something. The thing being modeled can generically be considered as a system within some domain of discourse. The model then makes some statements of interest about that system, abstracting from all the details of the system that could possibly be described, from a certain point of view and for a certain purpose. For an existing system, the model may represent an analysis of the properties and behavior of the system. For a planned system, the model may represent a specification of how the system is to be constructed and behave.

4.1 The Model-of Relation

We start with a condensed definition from Webster, Collins, Wikipedia, UML, [Bossel, 2013], and a general understanding of model.

> A model refers to a small or large, abstract or actual, representation of a planned or existing entity or system from a particular viewpoint and with a specific purpose.

Observe that this definition of model always defines a relation between the system acting as the model, and the system being modeled, see Figure 8.

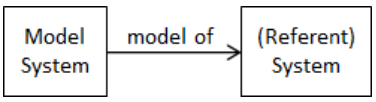


Fig. 8. Models are in a model-of relation with their referent systems.

[8] A debugger is a tool that can show the current state of execution in some notation.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

Keep in mind that a model is related to some other entity or system, and that this means that being a model is always the same as being in a relation with something we call referent system. This relation is usually called *model-of*. Furthermore, a model is something that can represent the referent system, as stated in [Podnieks, 2010]: "a model is anything that is (or could be) used, for some purpose, in place of something else." Given the terms introduced so far, a model is a system itself, which means it can be executed in some sense. This leads to our definition as follows.

Definition 4 (model). *A model is a system, that is in a model-of relationship to a referent system, existing or planned. A model resembles the structure and the behavior of its referent system by a model-of relation. A model might be created using a model description (e.g. a diagram, a formula, or code).*

The description is used for creating the model, but it is not the model itself, see Figure 9, which combines Figure 6 connecting description and model system,

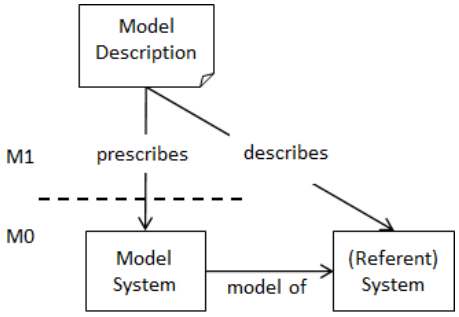


Fig. 9. Model, Description, and Referent System

and Figure 8 connecting model system and referent system. The combination yields a combined relation between the description and the referent system. This relation is most often called a describe relation. It is a bit weaker than the prescribe relation, but it is also relating a description to a system.

A system description always leads to a system (a set of possible executions). The system does not need to be a model system if there is no related referent system.

Note that both physical and mathematical models are systems, because it is their behaviour (their executions) that makes them models. Scale models are also concrete representations, but typically with an object structure that does not change over time.

Let us look at the small example in Figure 10 to see what this definition implies. There are specifications (descriptions) of books in UML and in Java for a library system. The Java description might be derived from the UML description (blue arrow in the figure), but this connection is not important at

Second International Workshop, M2P
2020, CCIS 1401, 67-89

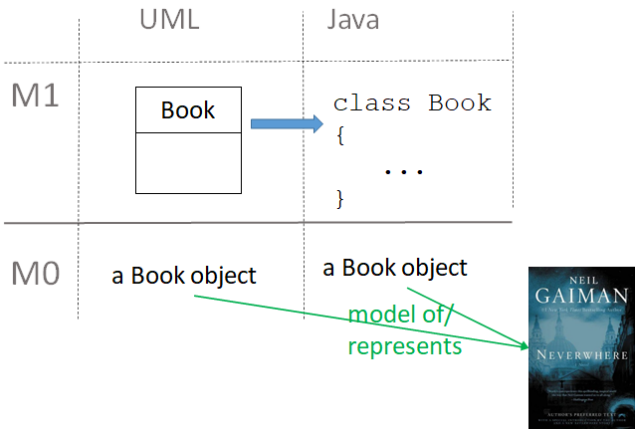


Fig. 10. Models of Books

the moment. The UML class allows a book object to be created at runtime, in the same way as the Java class allows a book object to be created at runtime. These runtime objects contribute to form a system, i.e. a library system. These two objects are typically models of the real book object existing somewhere in a library.

Libraries are systems with changing sets of books and loans. Models of libraries with the purpose to understand libraries or to make computer-based library systems must be systems of objects representing real books and loans. The model, in this case, is an actual representation.

Now that we know what models are, we can define modelling as follows.

Definition 5 (modelling). *Modelling is the activity to create a model based on a purpose. There are two ways to create a model.*

1. *Create the model directly as a system (scale model or physical model).*
2. *Create a description that implies a system which is the model (mathematical, computer, design models).*

The Scandinavian approach fits well with the process of modeling technical or environmental systems. Authors like [Schmidt, 1987], [Pritsker, 1979] and [Hill, 2002] identify three iterative phases of this process. The first phase establishes a model problem resulting from the system purpose, and a model is derived, the validity of which has to be verified experimentally for the purpose of the investigation. The model is described (programmed) in the second phase as an executable simulation model. Instead of experimenting with the original, we are now experimenting with the (executable) model. The third phase is dedicated to targeted experimentation. A distinction is made between experiments to prove the model validity of the simulator and experiments to solve the model problem. The phase is concluded by the intellectual transfer of the model results into the world of the original.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

4.2 Correctness

The model-of relation allows us to discuss correctness of models, see Figure 11. An example for the figure are interactions and use cases of UML, which provide

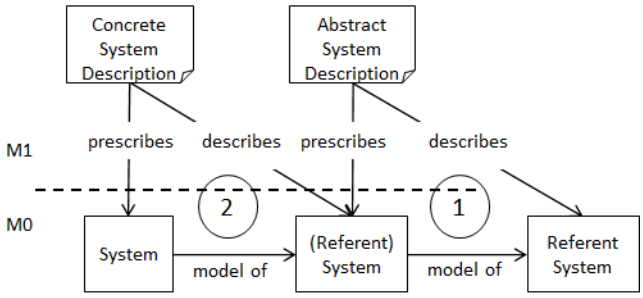


Fig. 11. Correctness

some kind of abstract system description by giving a partial formalization of ideas about the system as shown to the right of Figure 11. Typically, they need to be extended with more formality, for example class diagrams or code, as shown to the left of Figure 11.

The right part of Figure 11 named ① shows the same situation as Figure 9. In this part, the relation between (referent) system and referent system decides whether the model is correct. This is not a formal exercise as the referent system does not exist in a formal way. Validation is the process of finding out whether a system has the right model-of relation to an existing or planned real system, i.e. whether their executions match. In order to prove that the model system can represent the original system for the desired model purpose, i.e. that the model-of relation can be recognized as valid, according to [Bossel, 2013], the validity must be proven with regard to four different aspects: behavioral validity[9], structural validity[10], empirical validity[11], and applicability[12]. Validation by prototyping is the process of finding the system that is the desired model of a planned system.

As the match between model and referent system is already part of the definition of model, an incorrect model would not be a model at all. If the model

[9] It must be shown that for the initial conditions and environmental effects of the original system that are fixed within the scope of the model, the model system produces the (qualitatively) same dynamic behavior.

[10] It must be shown that the effect structure of the model (for the model purpose) corresponds to the essential effect structure of the original.

[11] It must be shown that in the area of the model purpose, the numerical results of the model system correspond to the empirical results of the Originals under the same conditions, or that they are consistent and plausible if there are no observations.

[12] It must be shown that the model and simulation options correspond to the model purpose and the requirements of the user.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

is able to reflect the behaviour of the referent system under certain conditions, it might still be a model for a restricted purpose. Correctness is very important for the existing parts of the system. They start with a physical system and provide the model of it later. For the new parts, the referent system is imagined and the model is the first physically existing system.

In the diagram part named ②, we see again a similar diagram as in Figure 9. This time, in addition to the description for the system, there is also a description for the referent system. This means we can formally compare the semantic implications of the two descriptions and determine whether the two systems match. This approach, called verification, uses the semantics of the two descriptions to compare the syntactic structures without going into the running systems. This way, the question of correctness is lifted to the level M1.

In a system with existing parts, we can simulate the system using the models of the existing parts. This way, we can *experiment* with the models instead of with the real systems and deduce properties. This is the typical development phase, where we use a model for the existing parts and the controller is changing rapidly due to better understanding of the system. Once the controller works well in the simulated environment, we can move to the real environment. The description is still the same, but now the existing parts are exchanged with their referent systems, i.e. the existing parts themselves. This means we do now deploy the new parts onto the real referent system.

5 Relation to the Kiel Notion of Model

In this section, we relate the concepts introduced in the previous sections to the Kiel notion of models [Thalheim et al., 2015]. The Kiel notion of model is a very general notion capturing models from all areas of science. The Scandinavian approach is geared towards dynamic models in computer science. Still, as our approach is quite general as well, the two approaches can be compared. We start with an introduction to the Kiel model, then we summarize our approach, and finally we compare the two.

5.1 The Kiel Notion of Model

The Kiel notion defines a model as follows.

Definition 6 (Kiel Model [Thalheim et al., 2015]). *A model is a well-formed, adequate, and dependable instrument that represents origins. Its criteria of well-formedness, adequacy, and dependability must be commonly accepted by its community of practice within some context and correspond to the functions that a model fulfills in utilisation scenarios and use spectra. As an instrument, a model is grounded in its community's subdiscipline and is based on elements chosen from the sub-discipline.*

In addition, [Thalheim et al., 2015] and [Thalheim and Nissen, 2015] as well as [Thalheim, 2019] give more detail to the criteria as follows.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

- A model combines an intrinsic deep model and an extrinsic normal model. The deep model is based on the community's subdiscipline and has its background, e.g. paradigms, assumptions, postulates, language, thought community. Models are typically only partially developed as normal models which properly reflect the chosen collection of origins.
- The criteria for well-formedness, adequacy, and dependability depend on the function that an instrument plays in the given scenario. Due to the function a model plays, it has a purpose and satisfies a goal. The (p) profile of a model consists of its functions, purposes and goals. A well-formed instrument is adequate for a collection of origins if (a1) it is analogous to the origins to be represented according to some analogy criterion, (a2) it is more focused (e.g. simpler, truncated, more abstract or reduced) than the origins being modelled, and (a3) it sufficient to satisfy its purpose. It is dependable if it is justified and of sufficient quality. Justification can be provided (j1) by empirical corroboration according to its objectives, supported by some argument calculus, (j2) by rational coherence and conformity explicitly stated through formulas, (j3) by falsifiability that can be given by an abductive or inductive logic, and (j4) by stability and plasticity explicitly given through formulas. The instrument is sufficient by (q1) quality characteristics for internal quality, external quality and quality in use. Sufficiency is typically combined with (q2) some assurance evaluation (tolerance, modality, confidence, and restrictions).
- The background consists of (g) an undisputable grounding from one side (paradigms, postulates, restrictions, theories, culture, foundations, conventions, authorities) and of (b) a disputable and adjustable basis from other side (assumptions, concepts, practices, language as carrier, thought community and thought style, methodology, pattern, routines, commonsense).
- A model is used in a context such as discipline, a time, an infrastructure, and an application.
- An instrument is a device that requires skill for proper use in some given scenario. As such it is (i1) a means whereby something is achieved, performed, or furthered, (i2) one used by another as a means or aid, (i3) one designed for precision work, and (i4) the means whereby some act is accomplished for achieving an effect. An instrument can be used in several functions in scenarios.

Models function in scenarios for which they are build. The intrinsic deep model mainly depends on its setting: the function that a model plays in given scenarios, the context, the community of practice, and the background. Scenarios often often stereotyped and follow conventions, customs, exertions, habits. The scenario determines which instruments can be properly used, which usage pattern or styles can be applied, and which quality characteristics are necessary for the instruments used in those activities.

Therefore, we may assume that the deep model underpins any model within the same setting. As long as we only consider models within a given setting

Second International Workshop, M2P
2020, CCIS 1401, 67-89

[Thalheim, 2017] we may use simpler notions of model, as given in [Wenzel, 2000] as follows.

> A model is a simplified reproduction of a planned or real existing system with its processes on the basis of a notational and concrete concept space. According to the represented purpose-governed relevant properties, it deviates from its origin only due to the tolerance frame for the purpose.

This definition already assumes the system background for simulation scenario in the context of production and logistics.

5.2 The Scandinavian Approach to Modelling

As a comparison, we have a look at the Scandinavian approach, see Figure 12. The Scandinavian approach has the following properties.

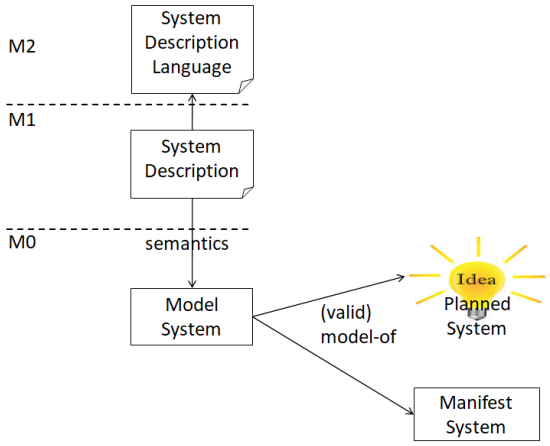


Fig. 12. Scandinavian Model Approach

- A system has a purpose.
- A model is a system.
- A model (system) is in a model-of relationship with another system.
- A system can be created from a system description.
- A system description adheres to the language in which it is made.
- A system is abstracted from the reality in terms of a discourse.
- You probably wants to say: The purpose of a system determines the properties of the system, with other properties 'abstracted away'.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

Scandinavian Approach	Kiel Notion
A system has a purpose.	profile
A model is a system.	specific instrument
A model (system) is in a model-of relationship with another system.	origin-relationship, (a1), (a3), (g), (b)
A system can be created from a system description.	construction scenario
A system description adheres to the language it is written in.	well-formed
A system is abstract in terms of a discourse.	(b), (g)
A system is abstract in terms of a focus coming from the system purpose.	(a2)

Fig. 13. Comparing the two approaches

5.3 Comparing the Approaches

Now we relate the elements of the two definitions, see Figure 13.

Most of the connections are obvious, but some comments are in place.

1. The Scandinavian approach distinguishes between model description and model system, which is not explicitly done in the Kiel notion where a model may consist of several tightly associated models, i.e. a model suite. A bi-model suite may consist of a model and its representation or informative model. The latter is essentially a model description. This leads to the well-formedness being related to the model in Kiel and to the model description in the Scandinavian approach.
2. The Kiel notion allows a description to be a model, which is not the case in the Scandinavian approach. The connection between description and model is further discussed in the next section.
3. The Kiel notion has much focus on purpose, usefulness, and the details of the model-of relation. This is has not been the focus of this paper, even though the Scandinavian approach has some ideas about it. Therefore, some aspects of Kiel do not appear in the table and many are grouped into the purpose and the model-of.
4. The Kiel notion is a generalisation of modelling practices in many scientific and engineering disciplines. It can be adapted to the Scandinavian notion by using specific adequacy and dependability criteria within a system construction setting.

As for the last point in the list above, the Scandinavian approach is based, not upon many scientific and engineering disciplines, but an understanding of application domains (or reality in general) consisting of phenomena (with measurable properties and behavior that changes these properties) and concepts classifying these phenomena. This is well-known from outside computer science, and when applied to modeling and programming, the short version is that objects model

Second International Workshop, M2P
2020, CCIS 1401, 67-89

phenomena in the application domain, classes model concepts, and subclasses model specialized concepts. Composite phenomena are modeled by objects having part objects, the special case being the system containing parts. Objects have their own behaviour (not just methods), reflecting that in some domains there are phenomena (e.g. processes in a process control domain) that are active and exhibit parallel behaviour. The two approaches agree that analogy is a semantic property of the systems (executions). Analogy means in both approaches that elements of a model system represent (model) the corresponding elements in the referent system (origin). It is far more general than the mapping property that is often required for the model-of relation.

The idea of system and the idea of instrument do not match completely. This article defines system, but the notion of instrument is not defined in [Thalheim et al., 2015]. Apart from being a system, an instrument (or tool) also has some property of being useful, as stated in (g). Although system and instrument may not match, the Scandinavian notion of model matches the Kiel notion of instrument in the sense that both are used for the purpose of finding out more about the system to be modeled. Even the model description as an instrument does not collide with the Scandinavian approach: here the model description plays an important role in communication about the model. The name DELTA means 'participate' in Norwegian, and the idea behind the language was that the users of the final system should be able to participate in the 'system description' (the term for model description at that time) prior to its implementation. This later lead to the idea of 'participatory design', but now other means are used, like mock-ups, prototyping. However, a description (e.g a program) still is an important instrument for developers.

6 Discussion

As explained before, the Scandinavian approach makes a clear distinction between a model and its description. This implies that a description is not a model.

We look into modelling and programming first, where we also discuss if programmers model. Then we consider code generation in the context of models and discuss why a description is not a model.

It might seem that the terms introduced and the details considered are not important in general. This is true, but still it is important to have the correct basic understanding in order to sort all the cases that might appear in programming and modelling practice.

6.1 Programming versus Modelling

The Scandinavian notion of model applies to modeling and programming in general and would be the starting point of a combined modeling and programming language as proposed by several authors ([Madsen and Møller-Pedersen, 2010], [Seidewitz, 2016], [Broy et al., 2017], [Cleaveland, 2018]).

Markus Völter has compared programming and modelling in [Völter, 2011] and [Voelter, 2018]. He uses a definition of model-driven as follows.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

> Model-driven refers to a way of developing a software system S where users change a set of prescriptive models M_i representing concerns C_i at an appropriate abstraction level in order to affect the behavior of S. An automatic process constructs a full implementation of S that relies on a platform P.

In this context, his conclusion is that modelling and programming are the same, and also coincide with scripting.

> Programming and modeling, in the sense of model-driven, where models are automatically transformed into the real system, cannot be categorically distinguished. However, the two have traditionally emphasized different aspects differently, making each suitable for different use cases.

He uses a similar definition of programming as we do, but he considers modelling to be creating a system description. This is of course the same as programming. However, a model description is not only a description of a system, but the system has to be related to a referent system using the model-of relation. Then all modelling by creating descriptions is also programming, but not all programming is modelling.

6.2 Do Programmers Model?

The question could be asked if programmers model? Of course, no programmer would want to write useless descriptions, so this kind of modelling they would avoid.

However, programmers *do* model, in the sense that they create systems that relate to reality, i.e. have a model-of relationship. They identify domain concepts and represent appropriate classes in the programs in line with the purpose of the system being developed. Objects of these classes are then model-of the corresponding phenomena. This is already discussed in Figure 10.

This kind of modelling is not alien to programmers and can help programmers use modelling. Of course, a system might also contain platform- or implementation specific elements that are not models of anything.

In fact, on a more basic level, each programmer is following an idea of what the program is supposed to do and tries to write a description of a system that does the same. This is in the very core of modelling.

6.3 Code Generation and Models

Let us look into code generation, either manually or automatically, see Figure 14. What is the relation between the high-level code (UML specification) and the low-level code (Java)? It is often claimed that the high-level code is a model of the low-level code. The same argument as before applies, in that the code itself is not a system. The connection between the two codes is given by the semantics of each of them. They both imply a system each via their respective semantics, and these systems can be related via the model-of.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

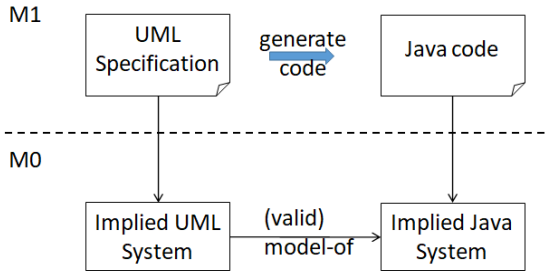


Fig. 14. Code generation and model-of

This indirect connection between the two kinds of code can be used to allow automatic code generation from the higher to the lower level. Please note that there might be different ways to create code from the higher level, and all of them can be correct as long as there is a match between the implied systems, i.e. they are semantically correct as discussed in Figure 11.

6.4 Why is a Description not a Model?

According to the arguments given before, a description is not a model, but implies a system which can be a model-of reality. But maybe it is possible to have also descriptions that are models?

Let's look at Figure 15. It is often claimed that the UML specification or the database specification is the model of the system produced later on. This is not completely wrong, but the relation between the UML specification and the referent system is indirectly composed of two relations as shown in Figure 15. There is a relation from the specification to the implied system, which is given by the semantics. The result of the semantics is then in the model-of relation to the system that is created. We have seen this combination already in the define relation, see Figure 9. So the specification (which is a description) is not the

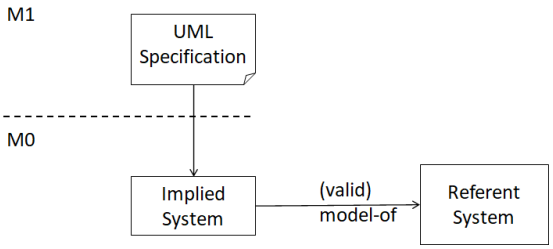


Fig. 15. Description to Model

Second International Workshop, M2P
2020, CCIS 1401, 67-89

model, but the implied system is the model of the referent system. This is in contrast to the following text from [OMG, 2017] on what a model is.

> A Model is a description of a system, where 'system' is meant in the broadest sense and may include not only software and hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given level of abstraction and viewpoint) are represented in the Model.

It is important to be precise that the description is not the model itself, but implies a system which is the model. This is also true for all model descriptions of database systems: A relationship diagram is not the model of the database, but it implies a system (mathematically) that is the model. In fact, the relationship diagram is then translated to code which again is a description. Running the code provides the system that is the model.

A similar situation relates to a model of the Mini (left in Figure 16). Two alternatives are shown in Figure 16, as presented at [Madsen and Møller-Pedersen, 2018]. The same question arises: Is the description the model of the mini, or the implied

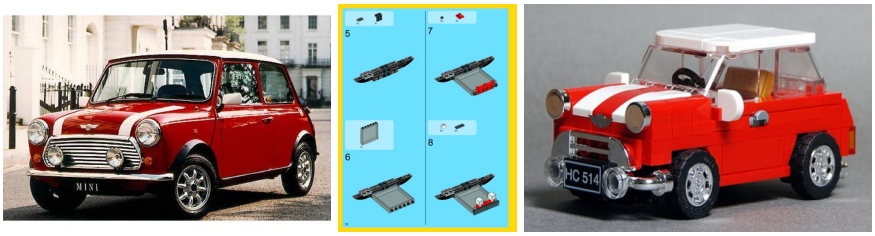


Fig. 16. Original, description, and model

result of the construction, i.e. the small Lego Mini? The answer is obvious - it is the small Mini - the system, not the description.

The connection between model system and real system is obvious when we look at experiments with the model. The Lego car allows to move forward and backward, and to turn. The situation is different with the description, which only helps to generate the Lego car.

Another example was presented in an invited talk by James Gosling, Oslo, 2017. Sea robots were developed by testing them out in a simulation of the sea with waves, currents and obstacles. The simulation, that is the program execution (system) with the sea robots as objects, is the model of how it will be for real. It is not the simulation program and the programs of the sea robots that are models of the sea and of the real sea robots. The example also shows that the programs of the sea robots became part of the real sea robots, i.e. development

Second International Workshop, M2P
2020, CCIS 1401, 67-89

by help of simulation. Note that these sea robots were simply programmed in Java without using a modeling language. Still, the simulation is a model of the sea with robots, and the sea robot objects are models of the real sea robots. Our definition of model as a system that is a model of another system is independent of which kind of language is used for making the model system.

7 Summary

This paper has discussed the relationship between models, systems, and descriptions. These terms were then compared with the Kiel notion of model. In this context, an executable model is the key instrument in the communication process about models and system development. It is also key in aligning the Scandinavian approach with the general Kiel notation of model. Other aspects of the Kiel modelling concept world that relate to the purpose, usefulness and details of the model-of relation are only partially discussed and should be deepened in further investigations. As the main result the paper has clarified the differences and similarities between modelling and programming. The paper has concluded as follows.

Systems belong to the modelling level M0. The paper has argued that systems can be real or imagined, but that they exist on their own. Therefore, they are to be placed on OMG level M0.

Executing a system description leads to a system. System descriptions are descriptions of systems and lead to systems when their semantics is applied. Here it is irrelevant whether the semantics is formal or not. The description itself is not the system, but leads to it.

A model is a system being model-of a referent system. The model-of relation exists between two systems, which may or may not have a description. The description is the model, it is not involved in the model-of. Instead, the implied system of the description is the model.

Modelling is a kind of programming leading to a model system. Programming is about writing descriptions for systems. When the system produced is a model of a referent system, then the programming is also modelling.

Two descriptions can describe the same system. When there are two descriptions of the same system, one high-level and one low-level, then both descriptions are related via their semantics, which may describe the same system on different levels of detail. The descriptions are not models of one another, because their syntax does not fit together, but the systems can be models.

With these clarifications it is easy to combine modelling and programming. There are far more aspects of models that are worthwhile to consider, which have been out of scope for this paper.

2020, CCIS 1401, 67-89

References

[Bossel, 2007] Bossel, H. (2007). *Systems and Models: Complexity, Dynamics, Evolution, Sustainability*. BBooks on Demand GmbH.

[Bossel, 2013] Bossel, H. (2013). *Modeling and Simulation*. Vieweg+Teubner Verlag.

[Broy et al., 2017] Broy, M., Havelund, K., and Kumar, R. (2017). Towards a unified view of modeling and programming. In *Proceedings of ISoLA 2017*.

[Cleaveland, 2018] Cleaveland, R. (2018). Programming is modeling. In *Proceedings of ISoLA 2018*.

[Dahl and Nygaard, 1965] Dahl, O.-J. and Nygaard, K. (1965). Simula—a language for programming and description of discrete event systems. Technical report, Oslo: Norwegian Computing Center.

[Dahl and Nygaard, 1966] Dahl, O.-J. and Nygaard, K. (1966). Simula: An algol-based simulation language. *Commun. ACM*, 9(9):671–678.

[Exner et al., 2014] Exner, K., Lindowa, K., Buchholz, C., and Stark, R. (2014). Validation of product-service systems – a prototyping approach. In *Proceedings of 6th CIRP Conference on Industrial Product-Service Systems*.

[Fischer et al., 2016] Fischer, J., Møller-Pedersen, B., and Prinz, A. (2016). Modelling of systems for real. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, pages 427–434.

[Fischer. et al., 2020] Fischer., J., Møller-Pedersen., B., and Prinz., A. (2020). Real models are really on m0 - or how to make programmers use modeling. In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, pages 307–318. INSTICC, SciTePress.

[Gjøsæter et al., 2016] Gjøsæter, T., Prinz, A., and Nytun, J. P. (2016). MOF-VM: Instantiation revisited. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, pages 137–144.

[Hill, 2002] Hill, D. R. C. (2002). Theory of modelling and simulation: Integrating discrete event and continuous complex dynamic systems: Second edition by b. p. zeigler, h. praehofer, t. g. kim, academic press, san diego, ca, 2000. *International Journal of Robust and Nonlinear Control*, 12(1):91–92.

[Holbæk-Hanssen et al., 1973] Holbæk-Hanssen, E., Håndlykken, P., and Nygaard, K. (1973). System description and the delta language. Technical report, Oslo: Norwegian Computing Center.

[ITU-T, 1995] ITU-T (1995). *Basic Reference Model of Open Distributed Processing*. ITU-T X.900 series and ISO/IEC 10746 series. International Organization for Standardization.

[Kleppe and Warmer, 2003] Kleppe, A. and Warmer, J. (2003). *MDA Explained*. Addison–Wesley.

[Madsen and Møller-Pedersen, 2010] Madsen, O. L. and Møller-Pedersen, B. (2010). A unified approach to modeling and programming. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 1–15, Berlin, Heidelberg. Springer-Verlag.

[Madsen and Møller-Pedersen, 2018] Madsen, O. L. and Møller-Pedersen, B. (2018). This is not a model : On development of a common terminology for modeling and programming. In *Proceedings of the 8th International Symposium, ISoLA 2018: Leveraging Applications of Formal Methods, Verification and Validation - Modeling, Lecture Notes in Computer Science 2018 ;Volume 11244 LNCS*, pages 206–224.

[Madsen et al., 1993] Madsen, O. L., Møller-Pedersen, B., and Nygaard, K. (1993). *Object-oriented Programming in the BETA Programming Language*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Second International Workshop, M2P
2020, CCIS 1401, 67-89

[Nygaard and Dahl, 1978] Nygaard, K. and Dahl, O.-J. (1978). *The Development of the SIMULA Languages*, page 439–480. Association for Computing Machinery, New York, NY, USA.

[OMG, 2017] OMG (2017). *Unified Modeling Language 2.5.1 (OMG Document formal/2017-12-05)*. OMG Document. Published by Object Management Group, http://www.omg.org.

[Podnieks, 2010] Podnieks, K. (2010). Towards a general definition of modeling. available at https://philpapers.org/rec/PODTAG.

[Prinz et al., 2016] Prinz, A., Møller-Pedersen, B., and Fischer, J. (2016). Object-oriented operational semantics. In *Proceedings of SAM 2016, LNCS 9959*, Berlin, Heidelberg. Springer-Verlag.

[Pritsker, 1979] Pritsker, A. A. B. (1979). Compilation of definitions of simulation. *SIMULATION*, 33(2):61–63.

[Scheidgen and Fischer, 2007] Scheidgen, M. and Fischer, J. (2007). *Human Comprehensible and Machine Processable Specifications of Operational Semantics*, pages 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schmidt, 1987] Schmidt, B. (1987). What does simulation do? simulation's place in the scientific method of investigation. *Systems Analysis Modelling Simulation*, 4(3):193–211.

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Software*.

[Seidewitz, 2016] Seidewitz, E. (2016). On a unified view of modeling and programming, position paper. In *Proceedings of ISoLA 2016*.

[Thalheim, 2017] Thalheim, B. (2017). General and specific model notions. In *Proc. ADBIS'17*, LNCS 10509, pages 13–27, Cham. Springer.

[Thalheim, 2019] Thalheim, B. (2019). Conceptual modeling foundations: The notion of a model in conceptual modeling. In *Encyclopedia of Database Systems*. Springer US.

[Thalheim and Nissen, 2015] Thalheim, B. and Nissen, I., editors (2015). *Wissenschaft und Kunst der Modellierung: Modelle, Modellieren, Modellierung*. De Gruyter, Boston.

[Thalheim et al., 2015] Thalheim, B., Nissen, I., Allert, H., Berghammer, R., Blättler, C., Börm, S., Brückner, J.-P., Bruss, G., Burkard, T., Feja, S., Hinz, M., Höher, P., Illenseer, T., Kopp, A., Kretschmer, J., Latif, M., Lattmann, C., Leibrich, J., Mayerle, R., and Wolkenhauer, O. (2015). *Wissenschaft und Kunst der Modellierung (Science and Art of Modelling) - Kieler Zugang zur Definition, Nutzung und Zukunft*. De Gruyter, Berlin, Boston.

[Union, 2011] Union, I. T. (2011). Z.100 series, specification and description language sdl. Technical report, International Telecommunication Union.

[Voelter, 2018] Voelter, M. (2018). Fusing modeling and programming into language-oriented programming. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Modeling*, pages 309–339, Cham. Springer International Publishing.

[Völter, 2011] Völter, M. (2011). From programming to modeling - and back again. *IEEE Software*, 28:20–25.

[Wenzel, 2000] Wenzel, S. (2000). Referenzmodell für die Simulation in Produktion und Logistik. *ASIM Nachrichten*, 4(3):13–17.

Trends and Future of Data Modelling

Hannu JAAKKOLA[a,1] and Bernhard THALHEIM[b,2]
[a] *Tampere University, P.O.Box 300, FI-28101 Pori, Finland*
[b] *Christian-Albrechts-University Kiel, Computer Science Institute, 24098 Kiel, Germany*

Abstract. Data (conceptual, data, information, knowledge) modelling is still the work of an artisan, i.e. an art in the best case, made by humans, because of the need for human intelligence. Data modelling is an essential part of Information System (IS) design, specifying how data is implemented as part of an IS. The principles of data modelling follow the evolution of IS development paradigms, and these in turn follow the progress of technological changes in computing. Although technology has changed a lot during the decades of commercial use of computers – since the early 1950s to now, close to 70 years – data modelling is still based on the same basic principles as decades ago. Or is it really so? Finding the answer to this question was the main motivation to start writing this paper. Since the future is more interesting than the past, we set our research problem to be "What are the challenges for data modelling in the future?". The reason for this is that we see some significant changes in the future in the data modelling sector which we wanted to examine. However, the future is a continuum of the past. The future cannot be fully understood without understanding the past. Humans also tend to forget the details of the past. Even the most remarkable innovations from the past have become part of the new normal. Consequently, at the beginning of our paper we look shortly at the progress of data modelling during the era of commercial computing. Our focus is on the recent past and we look at the technological changes that have been of key importance in data modelling in the role of triggers and enablers. To find the answer to our research question, we retrieved some recent studies handling the future of data modelling and analyse the challenges found in these sources. The paper is concluded by some future paradigms. In general, the big changes seem to be the growing importance of Artificial Intelligence (AI) and machine learning (ML) as its fuel. AI not only conducts algorithmic rule-based routines, it has learning capability, which makes it more intelligent and adaptable, and able to compete with human intelligence, even in data management tasks.

Keywords. data modelling, information modelling, information system modelling, information system design, data base

1. Introduction

Data Modelling defines the principles of how to create data models (physical, logical, conceptual, etc.) related to an information system (IS). It covers defining and determining the data needs of the target organization, and the goals of using the data. It is an essential part of IS design and creates the foundation of the whole development life cycle. Since the 1950s – the early phase of the commercial use of computers – the paradigms and

[1] hannu.jaakkola@iki.fi
[2] thalheim@is.informatik.uni-kiel.de

technologies of IS development have changed radically. The same goes for the principles of data modelling.

A variety of terms - conceptual modelling, data modelling, information modelling, etc. – are used to describe the same basic element of IS design from different points of view and abstraction levels. In our paper we have adopted the term "*data modelling*" to cover the use of modelling techniques in different parts of the IS development lifecycle; thus it is an umbrella term covering conceptual modelling as well as data-oriented aspects in IS design at all levels of abstraction. The development of the model reflects the principles of the development paradigm. The final manifestation of the conceptual model (after several modifications during the development life cycle) can be seen in the data structure (database, software data structure) and functionality of the application.

The aim of this paper is to consider the changes in data (conceptual, information, …) modelling over the decades, with the main aim to look at the current situation and the future. We look backwards to a paper from the late 1990s, which the second author of this paper co-wrote (Chen, Thalheim & Wong in 1999) [3]. If we examine the era of databases and the conceptual modelling related to it, that paper was written in the *growth era* (technology was maturing towards the growing use of relational databases) of database management systems (DBMS). It is reviewed briefly in Section 2 of this paper to provide an opportunity to compare "how the future was seen in 1999", "what really happened," and how we see the future today.

What has been the progress of data modelling over the decades? Foote's paper in 2017 [7] answers this question; it is slightly modified in the following according to the authors' subjective interpretations. It separates four (overlapping) phases covering the era of systematic data modelling from the 1960s to 2020. These are briefly introduced in the following paragraphs.

The development of DataBase Management Systems - DBMS (from roughly the 1960s to 1999). In this period a variety of implementation architectures existed: hierarchical, inverted list, network, and at the end of the period the idea of object-oriented database management systems - OODBMS. The last mentioned emerged in parallel with the coming of the object-oriented programming paradigm and Smalltalk. The first commercial database system, the integrated data store (IDS), was introduced in 1964 by Charles Bachman. Most of the "OODBMSs" were implemented on top of an existing architecture without having the pure properties of an object model; these extensions were also typical in the following era (RDBMS).

The birth of Relational DBMS – RDBMS (started at the end of the 1980s/early 90s). Edgard F. Codd introduced his data model based on relational calculus in 1969/1970[3]. At the beginning, it was the focus of a wide researcher community, and the first experimental implementations occurred quite quickly. Although the first "commercial" implementations (Multics) was available in the late 1970s, RDBMSs were not adopted for real commercial use until the 1990s. In addition to the logical structure of databases it brought a user-friendly way to define database contents and to implement queries in it

[3] The starting point of relational model and RDBMS is based on the following two articles:

Codd, E.F (1969). Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks, Research Report, IBM.

Codd, E.F (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. Classics. 13 (6): 377–87. doi:10.1145/362384.362685. Archived from the original on 2007-06-12.

the Structured English QUery Language (SEQUEL), which was later abbreviated to the more commonly known SQL. This also gave rise to the commonly used name for this era: SQL Databases. In IS industry practices this era is continuing, but to an increasing extent it is mixed with approaches providing additional functionalities for data handling.

Specialized DBMSs and their use in data analysis (started in 1990s and continuing in its varying forms). Foote refers to this era as the support for online analytical processing (OLAP) but in practice it covers a variety of forms to apply managed data for business intelligence applications, data mining and a variety of (big) data analytics. OLAP itself is an approach that allows multi-dimensional handling of data. To allow powerful processing the data must be organized to support this (multidimensional) method of data handling. In practice these are different applications of relational and NoSQL? databases. Currently the frontier technologies support effective handling of mixed structured and non-structured data in (big) data analytics. One of the frontier technologies today is *Apache Hadoop*[4], which consists of a data storage part (distributed file system (HDFS)) and a processing part (MapReduce programming model) supporting parallel data processing. We see this as a continuum of the progress that began with more simple "intelligent" data analysis, despite it being considered as the start of a new era of (specialized) data processing.

NoSQL (started in late 2000). The progress of the NoSQL data model was accelerated by the growing need for big data analytics. NoSQL provides a mechanism for handling data that no longer follows the structure of relational databases. The source of such data may be in large masses of documents and in the data streams (pipelines) of real-time WWW applications. In this kind of context, the role of data modelling faces new challenges; traditionally the data model is based on a structure – with components and the relations between them presented in a selected framework. In NoSQL data modelling, with such data stores, the model is embedded in the code, instead of traditional data models.

From this, we continue in the following sections of this paper by providing a look at the current situation and the future. The purpose and the role of data modelling have varied over the eras presented above. There is a clear link between the *models*, the *IS development paradigms* and the *tools used* in different life cycle phases. The evolution of programming languages has had a strong influence on shaping the practices of using these languages, as well as on the tools used to build the IS models. An additional aspect worth noticing is the capabilities of computing environments. Lack of processing power, computer memory capacity, the structure and capacity of mass memory are typical reasons for delays in adopting new technologies. What is the reason for RDBMS becoming a commercial success some 20 years after its theoretical foundations were introduced? According to Jaakkola et al. in 2017 [15], computer performance (memory, processing), mass memory capacity and networking (technology, speed) are triggers for adopting new paradigms and technologies. Progress in storing and handling data, as well as in data modelling, has had the same change triggers.

The aim of this paper is to consider the changes in data modelling over the decades, with the focus on the future. The history and path of the progress were dealt with above. We continue in Section 2 by mirroring the current situation with a view from 1999. Section 3 reports our findings related to the future of data modelling based on a "literature mapping" study. In Section 4 two future "paradigms" are discussed. Section 5 concludes the paper.

[4] Apache Hadoop – see https://en.wikipedia.org/wiki/Apache_Hadoop.

The *research problem* of this paper can be crystallized as follows:

What are the challenges for data modelling in the future?

2. How the future was seen in 1999

2.1. View of the future from the past

The work of Chen, Thalheim & Wong in 1999) [3] summarizes and abstracts the papers of a 1997 workshop published in a post-proceedings form. The papers in this volume were written by several researchers, mainly from the conceptual modelling community and especially by contributors to ER conferences. The paper we reconsider summarizes the entire post-proceedings, which were the basis for subsequent compilations by various authors, e.g. Embley & Thalheim in 2011 [6] and Thalheim in 2000 [25]. Here, however, we will use the summary from 1999 and then discuss the differences with the current state of the art.

The future and important directions are divided into the following in this paper:

- *Active modelling*: Modelling should become a continuous process. Once a model has been implemented within a system, the improvement of the system and the database code should be accompanied by a revision of the model. Models need continuous improvement due to changes in the applications, changes in the technology, integration of systems, extension to areas in which databases can also be successfully used, the quality of data in use, modernization of DBMS, and size of code and models.
- *Relationship between natural languages and conceptual modelling*: The semiotics of natural languages has great potential for improvement of modelling languages. At the same time, a sophisticated methodology similar to literal would improve modelling. Modelling languages should become as powerful as natural languages.
- *Conceptual framework for sharable information services*: Technology may enable the development of global information services that can be called in a variety of exchange facilities and protocols. The information explosion has to be met by the provision of small and effective services for the needs and demands of users.
- *Relationships between the real world and the software world*: From the viewpoint of normal users, computer systems are mainly software systems since hardware has become encapsulated and interfaced by software. Models of the software world and its activities allow us to handle, understand and evolve it properly according to the needs of the real world. These models should also support the cross-infusion of the software world and the information systems world.
- *Conceptual model as the basis for interoperability*: Systems should be transitioned into harmonized systems that run in a consistent and seamless manner instead of using a "let's-glue-pieces-together" approach. Models and conceptual models might be a top-down solution.
- *Conceptual modelling as the first step for application engineering*: Systems are continuously evolving. New users need their own interfaces which should be based on proper understanding of the entire metadata. Modelling should

become a continuous activity which includes management of change, extension of scope, and utilization as well as integration into existing infrastructures.

- *Global communication*: While global data sharing is going to take place, sharing knowledge and high-quality information remains a key concern. Aspects of information and knowledge management have to be included.
- *Human knowledge integration*: Human knowledge can be understood from three perspectives: normative, situational and existential. Its integration is the ultimate goal for information technology development. Such systems must support inference, justification, question proliferation and answer generation according to the profile of human beings.
- *Pragmatics of conceptual modelling*: Modelling languages should be extended in such a way that they allow the development of a conceptual data model at a high-quality level. At the same time, they should correspond to improved object-relational database management systems.

The list of trends remained incomplete at that time. It would never be possible to provide an exhaustive list. However, it is a survey of most of the major trends as seen in 1999.

2.2. Continuing - Some trends since 1999

The architecture of information systems became separated into a frontend system and a backend system. The frontend system can use several data model paradigms. The backend system is typically built as an object-relational system. The systems around 1999 followed a global-as-design approach with a global data model for the backend and derivable views defined on this global model for the backend. With the sophistication of the frontend, this decision has been revised, especially for web architectures and distributed databases. Data models then became *model suites* (by Thalheim in 2008) [26]. This model suite approach also allows the co-designing of data models with business process models.

The experience gained with the maturation of classical applications has been summarized, generalized and abstracted in reference models. These reference models can be used in new projects as a starting point. Universal applications use such generalized data models for the generation of a specific model

Information systems became the kernel of data-intensive applications with the advent of web infrastructure. These systems are distributed and integrate a number of viewpoints. They use a variety of data models and are continuously revised and modernized. The two main branches of programming (in the small and in the large) have been extended to programming in the web by partners that only partially follow a common strategy. Typical programming in the web is not governed by any guru. Projects co-evolve without tight coordination, integration or harmonization.

Data modelling has become an issue for almost all disciplines of science and engineering. While in the past databases were developed by specialists, laymen have now become systems developers. At the same time development and research have become more interdisciplinary. Collaborations that use common data are not yet supported by the co-development of the corresponding databases.

Databases have become huge in some cases. At the same time, data models have become *more complex*. Big databases are highly distributed and run in parallel. Data replication has enabled controlled redundancy.

2.3. Analysis and the state of the art compared to 1999

Let us now structure our trend observations and experience gained in many projects into three groups: directions that have resulted in almost completed research, directions that are still research tasks but achievable, and directions that proved to be unachievable:

(1) Successful research[5]:
- The linguistic foundation of modelling has become a common framework in data modelling.
- Modelling methodologies have reached maturity at ISO 33001 level 2 or 3.
- Distributed and federated database systems have become well-supported by modelling.
- Ontologies can be used for harmonization of viewpoints for business users.
- Service architectures and provision have become the standard.
- Question-answer forms combined with input-output forms provide an initial solution to the global communication problem.

(2) Issues for continuing research:
- Data modelling is still an art and has not become a culture. Each research and development group follow their own approach. The integration and harmonization of the variety of approaches hinder data integration in such collaboration projects.
- Model transformation and active modelling has not yet become state-of-the-art.
- Co-evolution of database systems and models is nowadays tackled by models-as-programs.
- Continuous modelling in dependence on changes is still a major lacuna.

(3) Tasks that have not been continued:
- Modelling languages will not achieve the expressivity of natural languages.
- Pragmatics of modelling did not receive common treatment in dependence of users.
- Research on object-oriented database systems became the source for object-relational system technology.
- Interoperability of data-intensive systems and applications can only be provided for greenfield application development and cannot be supported in migration and evolution scenarios.
- Knowledge and information management inside a database system is not supported.

We will return to these topics to some extent in the following sections.

3. How the future is seen in 2020

3.1. Challenges to data modelling in its current form

We know of more than two dozen data modelling languages. For instance, the handbook of conceptual modelling by Embley & Thalheim in 2011 [6] discusses almost a dozen of them. Half a dozen of these languages have found their way into practice with the most

[5] Obtained from a large reference list for this body of knowledge and research.

prominent UML class diagrams and various extensions of entity-relationship modelling languages. Data modelling based on such languages faces several challenges:

- The challenge of neglected foundations: Most languages are purely syntactically defined. Some of them exist only as graphical languages. UML class diagrams have several interpretations. Data modelling languages mainly provide notions with one lexical meaning of a word in a given culture and organization, e.g. the term "Person_Name" has many different meanings. Part of the foundation is thus to enhance a data model with concepts from a conception space. The syntax orientation of constructs limits semantics to local semantics for each construct. Full semantics of a data model are not derivable.
- The challenge of model usage beyond the description purpose: So far, data modelling has eroded to descriptive modelling. Data models are generally considered to be inspiration models for database system developers. The model will not be used as a prescription for development since many essential elements that are needed for development can be interpreted in various ways or are missing. The model will become an historic document for system documentation that cannot be and will not be maintained or modernized with respect to changes.
- The challenge of non-agreement in the community of practice: The (conceptual) data modelling communities have not agreed on a common notion of the (conceptual) data model in 45 years of its investigation in conferences, international bodies and projects. There are more than threescore different notions (Thalheim in 2018) [27]. Three ER conferences (2017, 2018 and 2019) incorporated workshops for the central issue of developing a commonly accepted notion of the conceptual model.
- The challenge of language simplicity: The entity-relationship model in its current widely used form uses rather simple language constructs. The main sentence constructor follows the SPO paradigm, i.e. two or more kinds of object are associated by a relationship type. English language has 25 sentence patterns and is thus far richer. The ER language is not cognitively complete. It allows only representation of three out of six cognition dimensions. Advanced extensions of the ER modelling language struggle with the sixth dimension.
- The challenge of the logical separation of syntax and semantics: Natural languages use wording that have a syntax, semantics and pragmatics. A word field combines all three semiotic dimensions. Computer science uses rigid layering and omission: syntax first, semantics later on the basis of the syntax, and pragmatics is not of interest. This separation leads to flat models, i.e. models whose constructs do not have an inner structure and an inner semantics, e.g. "Person_Name" with its specific nature.
- The challenge of concentration on one abstraction level for data: Data often come from various sources with their quality, i.e. micro-data, cleansed meso-data, aggregated macro-data. OLAP cube applications suggest that it might be essential to drill down from more abstract data to less abstract or to roll up from fine granular data to less granular. Moving though data abstraction layers requires understanding of the meaning of the data and of the attributes at their abstraction layers (Molnar & Thalheim in 2007) [22].
- The challenge of structure-behaviour separation: A common understanding of a data model is that a data model defines the "data structures, data operations, and so forth, that together make up the abstract machine with which the user interacts" (Date in 2005) [5]. Data models thus concentrate on the data structure, constraints defined on

those structures and, to a certain extent, operations which can be compiled from higher-order functions in order to support computing on these structures. Structures and constraints may however be dynamic (Thalheim in 2000) [25]. Furthermore, performance support requires auxiliary structures, application-governed management of the data and structuring. The theory of normal form which is today outdated has been developed for data restructuring according to performance and behaviour.

- The challenge of M0-M1-M2 layering: Natural languages allow definitions and construction of complex items inside the same language. They are somehow "flat". OMG proposes to separate things at the M0 layer from their representation at the M1 layer (termed the model layer), to separate representation from the language layer (M2, called the "meta"model layer), and to separate the language layer from the foundations layer (M3, called the "meta""meta" layer). This leads to a difficult treatment of IsA hierarchies together with class hierarchies (`a student is a person', `the class of student is a sub-set of the class of persons'). It is often inappropriate, e.g. the difference of various variants Is-A and of Is-Instance associations is not clear. Multilevel data modelling (Frank in 2014) [8] crosses these layers.
- The challenge of global-as-design: Data modelling assumes that a global schema is specified at the conceptual and implementation layers (which can be separated into a logical and a physical layer). The external layer concentrates on defining views which represent the viewpoints of business users. The three-layer architecture was appropriate for mainframe systems. Local viewpoints have to be represented by views. Co-design of data and business processes becomes very difficult since the business processes are based on a local-as-design paradigm. It results essentially in an impedance mismatch between the two extreme modelling strategies.

Despite these challenges there are many other lacunas, unexplored scientific issues and hidden culture problems.

3.2. First steps from the past to the present

At the end of section 1 we defined the research problem of this paper: *What are the challenges for data modelling in the future*". We also concluded that data modelling principles cannot be distinguished from the *general trends* in IS development, which in turn influences the IS development environment, including modelling and implementation tools. We also emphasized that progress in tools and environments depends significantly on progress in the triggering technologies. What are the leading technologies defining the trends in IS development? Some of these are listed here, referring to a variety of sources: virtualization and cloud, XaaS (something as a service), opportunity to use parallel (cluster based) processing, growing importance of system interoperability and commonly used open interfacing solutions (APIs, e.g. REST), integrating (big) data analytics to legacy ISs, the transition from plan-driven development towards agile development processes, distributed development and DevOps, the growing importance of application development for affordable devices (mobile terminals) using development platforms for them, the scalability of IS to be used in varying platforms, the use of the internet as a delivery mechanism, new kinds of revenue logics (transfer from licenses to pay-as-you-go), replacement of the customer by a faceless user (loose, if any, supplier-customer relationship), the importance of openly available components and (sub)solutions, growing value of the (external) data, "every product is a software product" – the era of IoT and embedded software. This progress

was examined from the point of view of the software business in a paper by Yrjönkoski et al. in 2019 [29].

Going back to history, the basic principles of conceptual modelling (e.g. ER models) were developed in the period of very traditional IS development based on the waterfall type of life cycle, a strong client-customer interface and the use of IS in a customer-managed operating environment. The progress in modelling has gradually followed the progress in the characteristics of ISs and their development process by adapting to the changes. One remarkable step towards the real needs of the ICT sector was the unification of modelling practices (languages) by introducing the unified modelling language (UML)[6] in the middle of the 1990s. Closely related to this, an iterative software development process – the rational unified process (RUP)[7] - was introduced to replace the traditional waterfall model. An important step in the adoption of UML was introduced by Kruchten in his 4+1 view software architecture model [8]. It integrates five views of IS models (based on UML) and highlights the principle of *multiple (consistent) models* in the modelling of IS during its development life cycle. Where traditional ER modelling focused on the *static structure* of the IS (concepts and their relations) leaving modelling of its dynamics (behaviour) to be done separately, UML highlighted and supported the need to model IS's dynamics as an integrated part of the whole. UML replaced the scattered set of models with an integrated set of modelling languages and transferred IS development from the era of structured development to the object-oriented (OO) era.

When analysing ER conference proceedings since 2010, we observe an increasing number of papers that solve some of the modern application problems by developing enhancements for the current technology. The data models proposed are essentially extensions of classical data models. Typical data modelling examples are proposals oriented to application challenges such as the following: (big) data analytics, data warehouses and data marts, data exchange among distributed systems, heterogeneous data ensembles, weakly structured massive data, evolving and migrating systems, clients such as search clients, web-based systems, support of the entire development cycle starting with requirements, graph-based computation, support for advanced systems such as geographic information systems, XML representation, challenging applications such as genomics and life sciences, enterprise data integration, service systems, research collaboration, support of model-driven approaches, privacy, not-only-SQL (NoSQL) systems, and the integration and collaboration of systems. At the same time, ontologies have become widely accepted as a modelling technique.

---

[6] OMG (1997). UML Specification version 1.1 (OMG document ad/97-08-11)". Retrieved from https://www.omg.org/cgi-bin/doc?ad/97-08-11 on January 29th, 2020.

[7] Wikipedia (2020). Rational Unified Process. Retrieved from https://en.wikipedia.org/wiki/Rational_Unified_Process on January 29, 2020.

[8] Kruchten, Philippe (1995). ViewModel of Software Architecture. IEEE Software 12 (6), November 1995, pp. 42-50. Openly available and retrieved from https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf on January 29th, 2020.

3.3. The current trends – being prepared for future needs

We approached this topic by conducting a simple literature study, implemented as an Internet search, applying (freely) the principles of the mapping study approach by Pettersen in 2008 [23]. From the results we selected a dozen articles for further investigation. The study gave evidence of our expectations: this topic has not been dealt with widely by researchers in scientific writings, but more by well-known analyst companies (Gartner, Forbes, McKinsey etc.) in their studies. In turn, these have been analysed in numerous articles, which were finally selected as the sources of our summarizing analyses. This was done in a critical manner augmented with interpretations / concluding remarks / subjective experience-based opinions.

The article by Ghosh in 2019 [13] names *Artificial Intelligence* (AI) and *Machine Learning* (ML) as the key factors of the current dynamic in data modelling. She lists some remarkable changes that are discussed in the following paragraphs - interpreted and integrated by the authors.

Data Science. Significant changes will occur in the division of jobs and duties. Ghosh refers to the study of Gartner in 2019 [12], which predicts 40 per cent *automation* in data science tasks (augmented analytics, with augmented data analytics as part of it). *Model analytics* decreases the opportunity for human error and increases the quality of models, whereas *automatic model generation* decreases human work. Data analysis tasks and traditional practices in master data management (MDM), metadata management and data governance can be automated by AI/ML. More power in (business) data modelling will be offered to *citizen data scientists;* analytics will be driven by automated ML models. *Two types of data models* (views of the same data) are needed – one for professionals and one for citizen users on a plug-and-play basis to execute quick solution type tasks.

Non-structured data and pipeline-driven data integration. The growing importance of non-structured data is a reality. New database technologies - NoSQL/non-relational databases, data lakes – are becoming part of ISs. In applications, instead of on the traditional ETL (extract, transform, load) basis, data is transferred in a pipeline-driven manner as a continuous stream from a variety of sources (IoT, social network data streams, etc.), which is not common in traditional ISs.

Traditional DB technologies are not disappearing – instead there is a hybrid mix of technologies. In spite of the rise of non-relational databases, the hybrid mix of database technologies maintains the importance of traditional data modelling - relational systems are not disappearing and will have a more important role than ever as part of the data infrastructure. Although "algorithmic intelligence, self-describing data formats and standardized models" have taken some of the labour out of data modelling, the newer database technologies have initiated a whole new set of challenges for data modellers. These *hybrid systems* are challenging for data modellers, who must be familiar with the beneficial use of opportunities that AI and ML provide for their traditional tasks.

Robotic Process Automation (RPA): The rise of RPA gives an increasing role to *business process modelling*. Combined with the cloud trend - SaaS (software as a service) solutions and ML (MLaaS – machine learning as a service) -, this creates challenges in data governance issues, which may remain *external* to the organization (externalization).

Transition? from problem specific to problem area specific solutions. In data modelling there is a transition from problem specific to problem area specific instruments. This is analogous to the progress of ISs, which are no longer single systems but complicated *systems of systems*; see the conference Paper of Boehm in 2006 [1]. At

the conceptual level we can also create an analogy as the transition towards *concept systems* instead of concepts per se – a kind of hierarchy. Interoperability-related requirements will of necessity move towards *commonly used standardized data structures and models*. Modern systems are to an increasing extent built with the guidance of existing frameworks – *platforms and ecosystems*, which leads to the *higher abstraction* of the models. The role of the ecosystem owner may create links and dependences that are not fully under the control of an individual organization. The performance expectations of modelling are expanding to new areas – to the ability to manage complex IS network structures, their interfaces and interactions, for example. Data governance is becoming more problematic – there are an increasing number of *unknown* data governance issues, due to cloud-based (XaaS) solutions. Cloud dominance affects data structures and needs support from a complicated data management ecosystem.

In summary, it is easy to see the growth of modelling complexity, the transfer of data (modelling) related tasks from professionals to end users and AI to support human work. At a basic level, data modelling remains as it has always been, but in practice a lot of new challenges are appearing.

The paper by Knight in 2020 [21] emphasizes that in the future (and now): "Modeling will need to do more, faster, with less". *Faster* means speed, *more* refers to the growth of the amount and complexity of data, and *less* indicates the need to lower the costs. Information must be processed faster, the volumes handled are greater, having their source in a *multitude of systems*, and a certain dynamic (readiness to react quickly to changing needs) is expected. AI will have a growing role in *automating business* tasks, even in the data management sector. The work of data modellers will transition to solving a decreasing amount of *complex business problems* in a short timeframe. The aim of companies is to optimize the data modelling activities to be able to manage them at lower cost; this includes the adoption of AI and the ML within it. Knight emphasizes four important trends in data modelling in 2020: just-in-time data modelling, better automation and machine learning, more widespread use of data modelling, and focused data modelling. These are briefly explained in the following paragraphs.

Just-in-time data modelling. This refers to changes in the IS development process. In agile development, even *agile data modelling* is expected. Because the agile philosophy is based on minimal (sufficient) design, even modelling activities are based on the *on-the-fly* principle. Just-in-time data modelling deals well with varying *mixed (hybrid) data structure*s – relational, NoSQL, dimensional[9] (data warehouse), master data[10], graph database[11], etc. The basic work of a data modeller remains the same – documenting relational (and other) structures. The work will happen to an increasing extent in the *cloud* because of its easy scalability, availability and low cost. This will also encourage use of *larger (and more complicated) applications*. Thanks to accelerating data transmission, data modelling is becoming "portable" – higher volumes of data from a wider range of systems and easy access to data models from anywhere by the relevant stakeholders.

Better automation and machine learning. Knight follows the same theme as Ghosh (discussed above) a year earlier. A lot of tasks will be automated based on the AI and ML built in the tools and development environments. The tasks following predefined

[9] https://en.wikipedia.org/wiki/Dimension_(data_warehouse)
[10] https://en.wikipedia.org/wiki/Master_data
[11] https://en.wikipedia.org/wiki/Graph_database

processes will be replaced / supported by RPA-type solutions, which transfer human routine work to machines. Therefore, the duties of data scientists / data modellers will be changed to more complex (non-routine, non-algorithmic) tasks.

More widespread use of data modelling. This confirms the trend already discussed above. Put simply – the role of *citizen data scientists* is growing because of the higher intelligence of the plug-and-play type (flexible) analysis and modelling tools.

Focused data modelling. Data modelling will become more focused; this is because less comprehensive general-purpose data models are needed for use with hybrids of DB architectures and data streams. The scope of the data models will *cover larger* amounts of enterprise knowledge and technical details. According to the IDC study and prediction[12], *unstructured data* will account for 80% of enterprise knowledge by 2025. The focus of data management will move to a *higher level of abstraction*, the management of knowledge. The models will be used by an *expanding audience of stakeholders* and must be customized to cover the needs of specific purposes. Data models must be organized by *multi-level concerns* covering business level, solution level (logical level concerns) and implementation, in some cases providing additional viewpoints.

Related to the other papers selected for our analysis, the findings seem to repeat the topics handled above. This is evidence of penetration (as a stop rule in the literature study) in the coverage of the topics handled. Therefore we will satisfy ourselves with considering these papers and articles briefly in the following paragraphs, by highlighting some new findings.

Frisendal (in 2020) [10]. This paper points out the importance of semantic *graph-based modelling* techniques and graph databases as an important category of NoSQL databases. Semantic graphs allow the modelling of semantic relationships between concepts. The ISO/IEC JTC1 SC32 WG3 database languages committee (SQL committee) has decided to add support for property graph queries within SQL. Graph databases represent data and its relations as a set of nodes; this "*semantic network*" supports data analysis type tasks. Frisendal also refers to traditional "*semantic web*" technologies (RDF, OWL etc.), which support the knowledge graph idea and have been adopted by some vendors.

Cagle (in 2018) [2]. This article focuses on large-scale enterprise models with large vocabularies to manage. He provides RDF[13] (resource description framework; semantic web) as a tool for such enterprise-level modelling.

Roussopoulos & Karagiannis (in 2009) [24]. This paper provides an interesting historical perspective on the progress of conceptual modelling. The last section of the paper focuses on the future. Conceptual modelling is seen as a continuous process. The web environment is seen as a constantly changing environment. The user must have the opportunity to *adapt the conceptual schema* and to track his interactions with the Web and SOA (service oriented architecture). This approach is called "*schema-during (SD)"*, in which the process of conceptual modelling is on a continuum with the operations in the database. Maintenance of the model needs a *user-friendly click-language* tool for *end-user conceptual modelling*. This approach is convergent with the growing

[12] https://www.datanami.com/2019/08/23/enterprise-search-in-2020-and-beyond-5-trends-to-watch/; See also https://solutionsreview.com/data-management/80-percent-of-your-data-will-be-unstructured-in-five-years/.

[13] https://www.w3.org/RDF/

importance of citizen data analysts – more responsibility is being transferred from professionals to end users.

Harper (in 2019) [14]. This article emphasizes the change from relational database towards *heterogeneous data sources*, which are purpose-built for real-time use (IoT, data from a variety of sources based on pipelining). *Adaptable data modelling* practices accelerate the modelling process, especially when including diverse sources. It is based on the use of tools supporting *codeless models* (model-driven techniques) and *visual model building*, which can be used for *collaborative data management*, especially when the data comes from a wide variety of different sources.

Wells (in 2016) [28]. This article provides an interesting view of data pipelines, which lead to *reverse (backward) data modelling*. The traditional model is built on a top-down basis – conceptual modelling -> logical design -> physical model - to get the structure to store the data. In pipeline-based data sources, the data already exists and is stored without the opportunity to change its physical structure. The logical model is then deduced in a *reverse modelling* process by starting with fields, then trying to deduce the items that they describe and finally the relationships between those items. ER models are not able to handle many constructs that are typical of *NoSQL databases*. Wells lists the following: many-to-many relationships, multi-valued attributes, embedded arrays, associations implemented without foreign key relationships.

Guess (in 2020) [11]. This article handles the role of *low-code programming*. Although this phenomenon is a little beyond the scope of the debate on data modelling, it confirms the transfer of work from professionals to end users, as in the "citizen data analyst" and "no-code" data modelling discussed above. Low-code development is supported by a *platform* that provides an environment for the programmer to use when developing an application using a graphical user interface. The platform typically supports application area-oriented development and may need additional coding for special situations. Typical applications suitable for low-code development cover *databases*, *automation of business processes*, user interfaces and *web applications*.

David (in 2017) [4]. This paper largely follows the same approach as the one we followed at the beginning of this paper. It lists milestones that have had / will have high importance in IS development, including aspects related to data modelling. All the aspects included in the paper have already been covered in the discussion above.

Kiyoki et al. handle in their paper [20] semantic modelling. It is based on the *Mathematical Model of Meaning* (MMM), in which an orthogonal multidimensional semantic space is created, and used for semantic associative search. Retrieval candidates and queries are modelled - mapped - onto the semantic space, and the semantic associative search is performed by calculating the correlation of the retrieval semantic space. In [20] the method is applied in the multi-spectrum images used for environmental analysis, covering spatial, temporal and semantic axis in the analysis and modelling.

We have analysed a set of articles / papers dealing with data modelling, and a slightly wider scope of IS modelling. The concluding message is that the future of modelling work – already visible in the present – faces a lot of challenges caused by the breakthrough of new types of data sources. On the one hand automation helps the work, but on the other hand the remaining work needs new updated skills. The role of the end user will grow in IS development and in the use of data sources. The essence of data is changing towards built-in semantics and characteristics of knowledge.

3.4. Summarizing the findings – discussion

We apply the categorization of technological changes introduced by Freeman & Perez in 1988 [9] in the classification of the changes in data modelling; the changes are divided into four categories (discussed in detail by Jaakkola et al. in 2017 [16; 17]):

- *Incremental changes* appear continuously in existing products and services (continuing the existing trend); these accelerate the existing change in the existing trend.
- *Radical changes* appear when new research findings are applied in products to transfer their properties or performance to a new step or cycle; at a certain moment there is an upward shift (break in the trend) caused by the innovation.
- *Changes in technological systems* are caused by combinations of several incremental and radical innovations in societal and organizational systems; these changes provide the means for increasing competitiveness and fast growth of productivity for early adopters.
- *Changes in paradigms* are revolutionary and lead to pervasive effects throughout the target system under discussion. They indicate permanent changes in societal systems, daily ways of operating and business models that replace the old ones.

Although the ideas of Freeman & Perez were developed to analyse technological changes in society, the principle can be applied in the case of a single technology, i.e. data modelling (as one of the technologies in IS design). We have included one additional category in our analysis below - *triggering and enabling technologies*. These are changes that do not have a direct connection to data modelling itself, but act as an enabler to make the changes necessary or possible.

In technology analysis it is worth noting that the situation does not remain stable. New change-driven innovative technologies tend to become the *new normal* over the course of time. This has happened for instance to UML and the object-oriented approach. These were revolutionary (radical, system change, paradigm change) at the time of their appearance, but no longer cause any significant changes.

In sub-section 3.1 we discussed the challenges related to the practices of data modelling and the languages used. The source of the challenges lies in the expression power of the modelling tools (languages), which typically are *semi-formal* by nature. They have high (but limited) expression power, but lack precision, which leaves space for interpretation. The beneficial aspect is high (visual, graphical) readability from the point of view of a heterogenous set of interest groups. The challenges have solutions, which are *incremental*, and are at most *radical* in character.

Sub-section 3.2 focused on the near-past phenomena that mainly have characteristics of *triggering and enabling technologies* and *radical* changes. ISs are transitioning quickly and powerfully towards cloud-based implementation and service architectures (XaaS, SOA). This has caused remarkable changes in data modelling. Modelling tasks are "*outsourced*" to the responsibility of the cloud data management and to an increasing extent an individual organization has to adopt the dominance of *"external" data models* and adapt their own solutions within these. We listed the following trends, which belong to the category of changes in technological systems:

- the growing importance of system interoperability and interfaces (API),
- transfer from plan-driven to agile development processes,

- the use of distributed collaborative (concurrent) development environments (Devops),
- transition towards platform and ecosystem dominance, and
- the growing value of external (non-modellable, non-structured) data produced by IoT and a variety of other external sources.

We are also losing the power of direct supplier-customer communication, which is being replaced by the development paradigm "faceless supplier to faceless client" caused by the growing demand for service-based systems, the use of web-based and ecosystem-dominated applications, and openly available components. This makes requirements engineering and elicitation of requirements of an IS more demanding.

Sub-section 3.3 focused on the "future" of data modelling. The future is partially already here as a continuum of the current movement towards the more mature solution models and technologies of the future. These are long-term changes in data modelling. Consequently, we have included these in the *system changes* category. Almost all the papers we analysed underlined *AI and ML* as the key change factors in data modelling. Undoubtedly, these have characteristics of triggering / enabling technologies. We do not want to separate these from their application context: on their own they are not important but, when connected to a variety of application contexts, they are. AI supports the work of data analysts, which is partially *automated*. Part of the work is also *transferred* to the stakeholders in their other roles – as *citizen data scientists*, *low code programmers*, *end users* etc. Data modelling must support the heterogenous needs of all these groups – the transition to *multilevel concerns* covering a *variety of abstraction levels* is a reality. Adaptable schemas, adaptable data modelling and visual model building are needed to support the needs of end users. In addition, the agile development process needs an on-the-fly type of modelling. *IS complexity* is growing. Consequently, models are growing, and focus must be placed on a multitude of systems instead of only one. Data management solves complex business problems rather than less complex IS problems. In data modelling the role of *external data* (pipeline-based, having the source in IoT nodes, sensors and a variety of data streams) is growing. Challenges in managing mixed (hybrid) data formats in one application make data modelling complex. External pipeline data requires *reverse modelling* without the opportunity to affect its structure. *Platform* and *ecosystem* based IS development supports the transition from specific data models to less comprehensive, problem area specific and general-purpose data models that support interoperability between independent systems. *RPA* transfers the focus in modelling from IS level to business process modelling.

In section 4, we will build a view of the future by presenting ideas about paradigm changes in data modelling.

4. Future paradigms in data modelling

4.1. Data modelling will become database programming

Data models are widely used in most data-intensive applications. Data models could be reduced to a database systems construction. There are, however, also other scenarios for the usage of a data model: communication and negotiation, description and conceptualization, documentation, explanation and discovery for applications,

knowledge discovery and experience propagation, and explanation and discovery for systems.

For system construction, we can see the *first generation of data modelling*: Models are used as a mediator and blueprint for database realization, for the description of the data in an application, for negotiation etc. They allow reasoning on database systems, i.e. *model-based development* and reasoning. This generation can be characterized as *data-modelling-enabling-data-programming*.

Jaakkola and Thalheim in 2020 [18] have also envisioned a *second generation of data modelling*: Model development and the modelling infrastructure have reached a maturity that allows the derivation of database structuring and supporting programs (or their essential parts) from data models or data model suites. Models will thus be the source code for executable programs. We characterize this generation as *data-modelling-as-programming.*

The next generation can be characterized as *data-models-as-database-programs*. Data models can be directly translated into database dictionaries and database programs, i.e. database programming can be performed almost entirely through modelling. The third generation of data modelling is then the *true fifth generation database programming,* which essentially frees the database developer from writing database programs and host programs with third or fourth generation languages. Everybody, including database laymen, who can specify data models will become a database programmer. Data models will then be model suites.

4.2. Data modelling must change along with application changes and technology evolution

Modern database applications are becoming more voluminous, have higher velocity, use a variety of models, are often hampered by data quality and veracity, are becoming more viable, have to cope with variations, use highly heterogeneous data, are challenged by viability, and must provide tangible added value. An appropriate technology has still not been found. To some extent, big data machines can solve some of the problems. It seems, however, that conventional technology must also be revised. Already very large data sets frighten us and result in the avoidance of complex operations beyond $n \cdot \log(n)$ complexity. Big data should be supported by technology with operations far below data size (i.e. n) complexity. Proper data modelling approaches that could meet these challenges are under investigation. So far, modelling research has not led to any convincing approaches.

The classical database systems have been based on OLTP-web_frontend, distributed database, OLTP-OLAP, OLTP-data_warehouse-data_mart, or the micro_data-macro_data-analysis_data layered database architecture. They result in an increase in data size and computational problems. We already know solutions for modelling through *model suites*, i.e. a collection of coherent and sufficiently tightly associated models where each of the data models reflects some of the aspects in these architectures. Kiyoki & Thalheim in 2013 [19] revised the layered architecture for a real big data application to a highly *distributed data network* with some data collaboration styles and patterns, e.g. publish-subscribe or master-slave. Each node in this network could use its data model. In the given case, each node used an internal data model, an import data model and an export data model where the last two were essentially views of the first model.

The application world will result in a number of rapid changes to data modelling as well as the capability of modern systems. All these changes will initiate revision modernization of data modelling. Let us consider some of them:

- *Flexible model transformation without information loss*: Models represent a specific application viewpoint that coexists with a larger variety of viewpoints in other application fields. At the same time, partners essentially operate using the same data. Therefore, we need data models that can be transferred to each other without loss of data and structure information. The notion of infomorphism is a starting point.
- *Large data models*: Already legacy (or rather heritage) applications use models that have been developed by generations of developers and have become "mannered and stilted cathedrals". With the wide spread of database technology, data models become larger and larger. Their management is becoming an obstacle for applications.
- *Weakly structured data*: Data might be weakly structured and can be enhanced at a later stage with refined structuring. Data models must be sufficiently robust for this kind of evolution or complete modernization.
- *Data models reflecting provenance and quality*: Data is seldom 100% correct. This observation was already true in 1999. Nowadays, such data is widely used. We use data that is partially corrected and cleansed together with data that needs improvement. Data might also be changed especially for biased usage with decreasing quality. Engineering has learned how to cope with errors inside its systems without changing the system. Such kind of inner quality correcting data models? allow to become robust.
- *Toughening the data model*: A data model might be a starting point that is continuously improved during exploitation. The model has a higher quality within the quality portfolio of the application. The classic 'code-that-smells' repair approaches will be replaced by the co-evolution of models and their data.
- *Adaptable and self-adapting data models*: Data structures and applications continuously evolve. This evolution can be mastered by a real artisan. It is far better to integrate the evolution strategy and the adaptation tactics into the data model.
- *Heritage extraction models*: Many applications have journeyed through data models that have evolved over time. This journey is badly documented. It is also hampered by very special implementation issues. We will fall into the Excel macro trap if we are unable to recover the models in such a form that the current status of data is captured.
- *Just-in-time data models*: Data models will be developed wherever data occurs in an application. The initial data model will often be proprietary and not harmonized with existing ones. A typical case is large research clusters (e.g. within the German Excellence Programme) without controlled data integration.
- *Data models reflecting storage are separated from data models supporting computation*: It is not necessary to keep all the data together with the data processing facilities. Instead, we might use a universal data store with all the data together with local data stores that neatly correspond to the local application. The binding mechanism could be a sophisticated export/import view management.
- *Special data models for special tasks*: Data mining and analysis also target the detection of patterns within the data. These patterns are the kernel of explanation models. The data used is empirical data and is thus often of problematic quality. We might use data model pipelines that reflect this mining and analysis framework.

Similarly, many applications need their specific domain-oriented modelling language which is well embedded into the application domain. Enterprise data models must thus be different from social data models.

- *Layman data models*: Data models are often developed by users that do not have sufficient experience and knowledge of modelling. Models reflect their disciplinary, education and business context without explicitly stating it. Models are local. Their integration is only possible if the full context is known.
- *Harmonization of data modelling languages with computation features*: Big data challenges current technology and will force us to redevelop current database technology. We envision that only those operations that perform well in a given architecture and in a given application world are going to be realized in systems. Similar to the past when we co-developed algorithms with their own data structure, we need data modelling languages that support high-performance computation for challenging data volumes.

This list is at its best only a starting point. It should be enhanced by the trends discussed above in order to meet the challenges discussed above. One solution path to better data models is *standardization*. We start with high-quality models as components of the models. Model composition follows approved composition approaches. Mechanical engineering and industrialization have widely used this approach. Another solution is development through *generic models*. These models are supported by sophisticated refinement strategies and tactics for their improvement. Generic models are also widely used in engineering for production moulds or forms.

5. Conclusions

At the beginning of this paper we set our research problem: "What are the challenges for data modelling in the future?". We did not dissect it in the form of research questions but followed a systematic path that started with an analysis of the progress path of data modelling, covering the decades from the early phases of commercial computing until the present day. After that we made a "forecast of the past" by benchmarking a future forecast from 1999 – over twenty years ago. It provided us with an opportunity to compare the state of the art of today to the expected progress. Understanding the future is not possible without understanding the past – everything is a kind of continuum from the past to today and on into the future. The future analysis of the paper is based on the "literature" mapping study. We collected expert opinions from several sources about future visions in data modelling. These findings were then classified and analysed. No clear future path can be defined, but a set of dominant factors can be gathered – the growing complexity of models, increasing tool intelligence, growing importance of external (non-modellable) data, heterogeneity of stakeholders in modelling and technology as a change driver – just to mention some of the findings. The paper concludes with two future paradigms.

References

[1] Boehm, B. (2006). A view of 20th and 21st Century Software Engineering. Paper presented at the Proceedings of the 28th International Conference on Software Engineering (ICSE), Shanghai, China.

[2] Cagle, Kurt (2018). Why Data Modelling is Important (and Why It's Not). Retrieved from https://www.forbes.com/sites/cognitiveworld/2018/11/05/why-data-modeling-is-important-and-why-its-not/#1e9cc98e50cb on January 31st, 2020.

[3] Chen, P. P., Thalheim, B., & Wong, L. Y. (1999). Future Directions of Conceptual Modeling. In G. Goos, J. Hartmanis, J. van Leeuwen, P. P. Chen, J. Akoka, H. Kangassalu, & B. Thalheim (Eds.), Conceptual Modeling: Current Issues and Future Directions (pp. 287-301). Berlin, Heidelberg: Springer Berlin Heidelberg. Available also from https://www.researchgate.net/publication/2847577_Future_Directions_of_Conceptual_Modeling on January 31st, 2020.

[4] David, Amos (2017). Current trend in data modeling and information systems. International Conference on Applied Information and Communication Technology. Lead City University, Ibadan, Nigeria. Retrieved from https://www.researchgate.net/publication/327120249_Current_trend_in_data_modeling_and_information_systems on January 31st , 2020.

[5] Date C.J. (2005). Database in depth: Relational theory for practitioners. O'Reilly, Sebastopol.

[6] Embley D. and B. Thalheim, editors (2011). The Handbook of Conceptual Modeling: Its Usage and Its Challenges. Springer.

[7] Foote, Keith D. (2017). A Brief History of Data Modeling. Retrieved from https://www.dataversity.net/brief-history-data-modeling/ on January 29th, 2020.

[8] Frank U. (2014). Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design. Business & Information Systems Engineering, 6(6):319–337, 2014

[9] Freeman C., Perez C. (1988). Structural Crises of Adjustment, Business Cycles and Investment Behavior. In Dodi G., Freeman C., Nelson R., Silverberg G. and L. Soete L. (Eds), Technical Change and Economic Theory. Pinter Publishers, London.

[10] Frisendal, Thomas (2020), Ten 2020 Visions for Data Modelers. Retrieved from https://www.dataversity.net/ten-2020-visions-for-data-modelers/ on January 31st, 2020.

[11] Guess A.R. (2020). Low-Code 20/20: A Clear Vision for Solving the Software Crisis. Retrieved from

[12] Gartner (2019). To 10 Strategic Technology Trends for 2019. Retrieved from https://datavizblog.com/2018/11/18/gartner-top-10-strategic-technology-trends-for-2019/ on January 29th, 2020.

[13] Ghosh, Paramita. (2019). Data Modeling Trends in 2019. Retrieved from https://www.dataversity.net/data-modeling-trends-in-2019/ on January 29th, 2020.

[14] Harper, Jelani (2019). 2019 Trends in Data Modeling: Real-Time Integration for Cognitive Computing. Retrieved from https://aibusiness.com/2019-trends-data-modeling/ on January 31st, 2020.

[15] Jaakkola, H., Henno, J., & Mäkelä, J. (2017). Technology and the Reincarnation Cycles of Software. In Z. Budimac (Ed.), SQAMIA 2017 - Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications. Belgrade, Serbia, September 11-13, 2017. (Vol. Vol-1938, pp. 5:1-10). Belgrade, Serbia: CEUR Workshop Proceedings.

[16] Jaakkola, H., Henno, J., Mäkelä, J., & Thalheim, B. (2017a). Today is the Future of Yesterday, What is the Future of Today? In P. Biljanović (Ed.), MIPRO 2017 - Proceedings of the 40th Jubilee International Convention. May 22-26, 2017, Opatija, Croatia. (pp. 741-749). Opatija, Croatia: Mipro and IEEE.

[17] Jaakkola, H., Henno, J., Thalheim, B., & Mäkelä, J. (2017b). The educators' telescope to the future of technology. In P. Biljanović (Ed.), MIPRO 2017 - Proceedings of the 40th Jubilee International Convention. May 22-26, 2017, Opatija, Croatia. (pp. 766-771). Opatija, Croatia: Mipro and IEEE.

[18] Jaakkola H. and Thalheim, B. (2020). Model-based fifth generation programming. In Information Modelling and Knowledge Bases Vol. XXXI, Frontiers in Artificial Intelligence and Applications, 312, pp. 377–396. IOS Press.

[19] Kiyoki Y. and Thalheim, B. (2013). Analysis-driven data collection, integration and preparation for visualisation. In Information Modelling and Knowledge Bases, volume XXIV, pp. 142–160. IOS Press.

[20] Kiyoki, Y., Chen, X., Sasaki, S. and Koppåipat, C., Multi-Dimensional Semantic Computing with Spatial-Temporal and Semantic Axes for Multi-spectrum Images in Environment Analysis. In Welzer, T., Jaakkola, H., Thalheim, B., Kiyoki, Y., Yoshida, N. (Eds.), Information Modelling and Knowledge Bases XXVII. IOS Press, 2016, pp. 14-31.

[21] Knight, Michelle (2020). Data Modeling Trends in 2020: A Year of Optimization. Retrieved from https://www.dataversity.net/data-modeling-trends-in-2020-a-year-of-optimization/ on January 31, 2020.

[22] Molnar A. and B. Thalheim (2007). Conceptual development of OLAP applications. In Business Intelligence: Methods and Applications, pp. 27 – 38. Klöden-Verlag, 2007.

[23] Pettersson F., Ivarsson M., Gorschek T., Öhman P. (2008). A practitioner's guide to lightweight software process assessment and improvement planning. Journal of Systems and Software 81, 6 (June 2008), 972–995. DOI:https://doi.org/10.1016/j.jss.2007.08.032.

[24] Roussopoulos N., Karagiannis D. (2009). Conceptual Modeling: Past, Present and the Continuum of the Future. In: Borgida A.T., Chaudhri V.K., Giorgini P., Yu E.S. (eds) Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science, vol 5600. Springer, Berlin, Heidelberg

[25] Thalheim B. (2000) Entity-relationship modeling – Foundations of database technology. Springer, Berlin.

[26] Thalheim, B. (2008). Model suites. In H. Jaakkola, editor, Selected Topics on Distributed Disaster Management: Towards Collaborative Knowledge Clusters., pp. 108 – 128. Tampere University Press, Pori unit.
[27] Thalheim B. (2018) Conceptual model notions - a matter of controversy; conceptual modelling and its lacunas. EMISA International Journal on Conceptual Modeling, February, pp. 9–27.
[28] Wells, Dave (2016). Big Changes in the World of Data Modeling. Retrieved from https://tdan.com/big-changes-in-the-world-of-data-modeling/19578 on January 31st, 2020.
[29] Yrjönkoski, K., Jaakkola, H., Mikkonen, T., Systa, K., & Henno, J. (2019). SQAMIA: Software business: a short history and trends for the future. In Z. Budimac & B. Koteska (Eds.), 8th Workshop on Software Quality Analysis, Monitoring, Improvement and Applications, SQAMIA 2019 Proceedings (Vol. Vol-2508, pp. 18:11-18:18): CEUR Workshop Proceedings.