Linguistic Structures in the Light of the Digital Transformation: Addressing the Conflict Between Reference and Change

Ulrich Frank

1 Introduction

We are in the middle of a gigantic transformation that is unprecedented in the history of the humankind. While we do not know what the exact outcome of this transformation will be, it seems obvious that it will result in fundamental changes. It will not only change the way we work and live, but, also, how we learn, speak—and think. Change of this dimension leaves many puzzled, is perceived as a threat by not only a few, and regarded as a fabulous opportunity by some. As researchers in Business Information Systems we have the privilege to not only study the digital transformation and the manifold phenomena it creates, but also to be among those who are asked for advice with respect to shaping the transformation for the good of society. Therefore, the digital transformation does not only create fascinating research questions, but also a serious responsibility for researchers in our field. This situation demands for asking essential questions regarding the subject and objective of research in Business Information Systems. First, the prevalent model of research in Information Systems focusses on studying the actual use of information technology and its development. Following the impressively successful neo-positivist model of research in the natural sciences, it aims at describing and eventually explaining observable phenomena. For this purpose, hypotheses that were derived from theories are tested against empirical data. While this approach is convincing at first sight, because it corresponds to a wide-spread understanding of science, it is accompanied by serious doubts about its suitability. They comprise principal differences between social systems and nature, the peculiarities of human cognition and thought, and philosophical concerns about the applicability of the correspondence theory of truth. Against the background of the digital transformation, there is a further concern that might be even more convincing than epistemological or methodological objections: is it sufficient to study the

U. Frank (✉)
University of Duisburg-Essen, Essen, Germany
e-mail: ulrich.frank@uni-due.de

K. Bergener et al. (eds.), *The Art of Structuring*,
https://doi.org/10.1007/978-3-030-06234-7_5

present to develop attractive and feasible orientations for change? Can the structures that are identified with current patterns of developing and using information systems be applied to future systems and their use, too? Are they sufficient to develop images of a future world that could inspire the transformation? I am skeptical that this will be the case. Since there are no theories that would allow for predicting the future, it is a daring and careless, if not irresponsible assumption that the future will be an extrapolation of the past. A second, more specific question relates to the design of business information systems. The development of methods and tools for supporting analysis and design is at the core of research in Business Information Systems. Is it appropriate to use comprehensive analysis and design methods in times of disruptive change, where yesterday's requirements may be outdated today already? With respect to the development of business information systems, conceptual models have been for long the undisputed instrument of choice. In particular, reference models that serve as a foundation of a wide range of software systems are especially promising (Becker, Delfmann, & Knackstedt, 2007, Becker, Algermissen, Niehaves, & Delfmann, 2005, Fettke & Loos, 2007). They are not only an attractive research topic. Furthermore, they offer substantial benefits for organizations, too. The reuse of thoroughly developed conceptual models enables better software systems at lower costs. However, in recent years, this assumption has been challenged, especially by the proponents of so-called "agile" approaches. This brings us to a further essential question. Is it still appropriate to focus on conceptual models or are code-oriented approaches better suited to cope with the peculiarities of an ever-changing world?

To analyze these questions, I will at first look at a principal need of all systems, technical and social, to work properly and to satisfy economic constraints. Without reference, systems cannot survive. They would not allow for communication and integration, would not enable the evolution of economies of scale, and without reference, software systems would literally make no sense. Second, I will look at the need for designing information systems that are facilitators of change rather than inhibitors. At first sight, the need for reference, and the need for supporting change are in conflict. However, as I will try to show, there are ways to substantially relax this conflict. They do not only depend on the development of powerful abstractions, but also on education programs that emphasize the prospects of abstraction.

2 The Need for Reference

There is no communication without reference. The words or signs we use to express a thought need to refer to concepts that we share (or assume to share) with the recipients of our messages. Software systems depend on reference, too. This applies in two respects. First, the words used in program code refer to (virtual) memory locations, types, instructions, etc. Second, to make sense of software and, hence, to use it properly, it needs to include references to concepts its users are familiar with. This is typically achieved through words or signs used in the domain where a program is deployed. For communication, and for software to work properly, references must

be reliable. References must not be corrupted, e.g. by redirecting them or by deleting the referenced concept or artefact. In other words, references should be stable.

2.1 The Pivotal Role of Conceptual Models

Software systems are linguistic artefacts. Their development requires some kind of implementation language, the constructs of which are mapped to the instruction set of a computer. In an ideal case, there is a formal implementation or specification language that allows taking advantage of formal methods to ensure that the software satisfies its requirements and is free of contradictions. However, apart from pure formal semantics, a formal representation does not include any meaning. In particular, it does not help with analyzing requirements and mapping them appropriately to a software architecture. For understanding a domain, to communicate about it, and for eventually re-organizing it, it is essential to use concepts that make sense to us and that are suited to guide us with appropriately structuring it. To serve this purpose, conceptual modelling makes use of modelling languages that allow referring simultaneously to concepts of implementation languages and to concepts of the language used in the targeted domain. Concepts of the domain language need to be reconstructed with the concepts offered by the modelling language. Figure 1 illustrates the role of conceptual models that are specified with a general-purpose modelling language (GMPL).

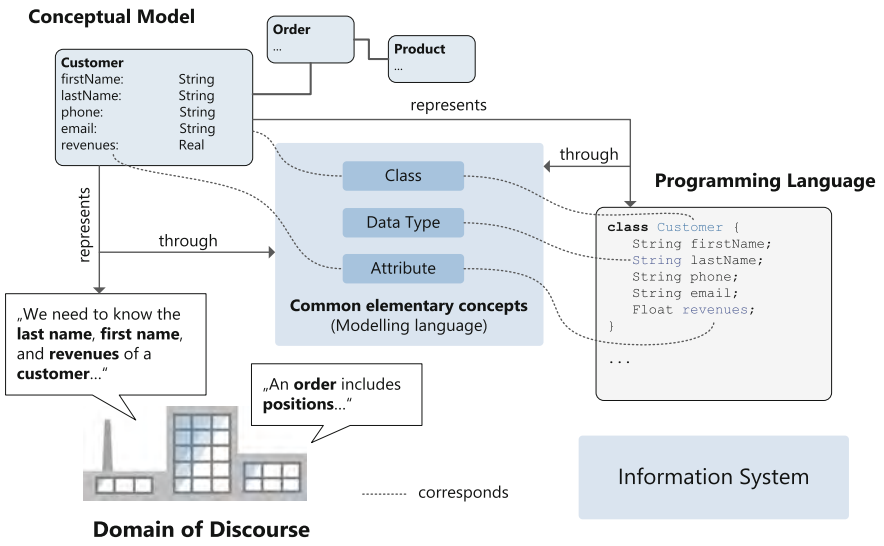

Fig. 1 Illustration of conceptual modelling

There have always been students and even professional programmers who claim that they would not need conceptual models. They are wrong. While it is conceivable to write code without drawing a diagram that represents a conceptual model, writing code without a model of the domain is not possible. Doing without conceptual models would mean doing without thinking: "But besides intuition there is no other kind of cognition than through concepts." (Kant, 1998, B93, A 68). It would also mean to ignore the pivotal relevance of codified knowledge: "Models are proffered truths. To proffer truth is the human means of acquiring knowledge. In this sense, cognitive acquisition, human learning is essentially mediated by representation." (Wartofsky, 1979).

A conceptual model serves as a reference in various ways. First, it provides a common representation for developers and prospective users they can refer to in order to reduce the chance of misunderstanding. Second, it can be used as a common reference by various development teams in order to foster cross-application reuse and integration. The benefits of conceptual modelling are accompanied by serious challenges. From an economic perspective, it is the question how much effort is justifiable for the development of conceptual models, and how this effort can be reduced. From an epistemological perspective, it is a challenge to assess the quality of a model, or, in other words, the quality of the knowledge represented by a model. From both perspectives, it is relevant to know how conceptual models need to be designed to enable flexible information systems that allow the convenient adaptation to changing requirements.

2.2 The Ambivalence of Semantics

Reuse is the most promising approach to reduce development costs. There are two principal approaches to promote reuse in conceptual models. Both approaches can be combined. Reference models cover a range of possible applications. At best, a reference model can be applied directly without adaptations. Domain-specific modelling languages (DSML) provide modellers with domain concepts (see Fig. 2). Hence, they narrow the gap between a domain and the representation of a corresponding software system. Thus, they foster modelling productivity and model quality. While GPML allow for designing any kind of model, even the most absurd ones, a DSML excludes models that violate the embedded domain-specific constraints.

Developing reference models with a DSML is especially promising, because it facilitates safe and convenient customization. However, designing a reference model or a DSML for reuse is confronted with a serious challenge. It is related to principal conflicts. On the one hand, there is a conflict between range of reuse and productivity of reuse. On the other hand, there is a conflict between stability of references and flexibility. The more specific a model artefact, or the concept of a modelling language, is, in other words, the more domain-specific the semantics it includes, the better it is suited to foster productivity of reuse. However, at the same time, specific concepts are limited to a certain context, that is, their range of reuse, and, hence, the

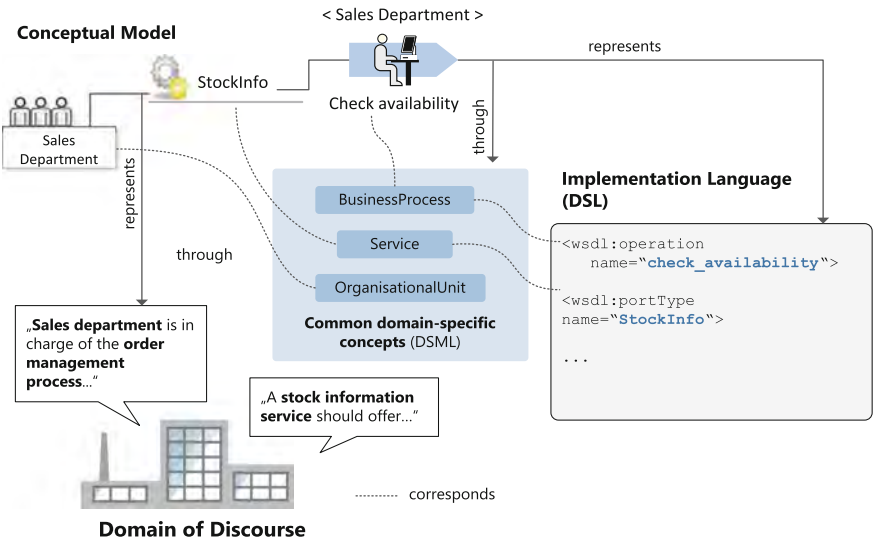

Fig. 2 Illustration of DSML

economies of scale they enable are restricted. To justify investments, reference models and modelling languages should be stable. At best, they should be standardized. However, at the same time, a standard may be an inhibitor of change, because the costs to abandon a standard may be prohibitively high. The more specific a model or a modelling language is, the more likely it is that it will not last for too long in an ever changing world.

The ambivalent nature of semantics also affects another pivotal aspect of information system design. Integration of system parts requires common concepts that all parts can refer to in order to communicate. The more specific these concepts are, the more effective communication will be, or the higher the level of integration they enable. Take, for example, two systems that exchange data, which represent products. If a product is represented by a string only, it can be mistaken for anything. The more semantics is incorporated in the representation, the better are the chances for the receiver to apply an appropriate interpretation. At the same time, the range of integration will decrease, since only components can be integrated that include references to a specific concept.

Against the background of this conflict, it seems reasonable to follow a widespread rationale of systems design, which is reflected by the slogan of "loose coupling" and by technologies such as service-oriented architectures. Components that are coupled via interfaces with little semantics only, e.g., XML documents, can be replaced by a wider range of other components that can cope with those interfaces, too. At the same time, such an approach is not satisfactory. On the one hand, it leads to redundancy across loosely coupled (badly integrated) components, which would compromise maintainability. On the other hand, it seems strange that the efficiency

of communication between components is reduced. Who wanted to communicate in a language consisting of a few primitive concepts such as string, integer, etc. only?

3 Coping with Change

Conceptual models are sometimes seen as inhibitors of change. With respect to the effort and time it takes to create a comprehensive model, it is not beside the point to suppose they are already outdated at the time when they are released. Against this background, it does not come as a surprise that conceptual modelling has been discredited during the last years. At the same time, other approaches to cope with change have gained remarkable attention. They are based on the assumption that top-down approaches that demand for a comprehensive conceptualization before the implementation starts are not feasible or that having them done by humans is too expensive and too slow.

3.1 Agile Approaches to the Rescue?

About 20 years ago, a book on "extreme programming" (Beck, 2000) became a driver of a movement that had a remarkable impact on the practice of software development. Based on a manifesto of 12 principles, the proponents of so called agile approaches promised to uncover "better ways of developing software by doing it and helping others do it." (Agile Manifesto) They addressed various critical issues that the academic field of software engineering had widely ignored. Software development is not a mere engineering task. Instead, its success depends on communicating with customers, on collaboration, on budgets, management support and—on good software developers. While stressing the importance of these factors can be seen as an enrichment of traditional approaches to software development, a further aspect of the manifesto represents a radical criticism of software engineering. Software engineering is based on the assumption that the comprehensive design of a system before its implementation is a prerequisite of avoiding mistakes. Taken to the extreme, it means that programming is the task of proving an implementation against a given specification (Dijkstra, 1972). The proponents of agile approaches favour testing over proving, and, more important, the early realization of working software is given top priority. They do not deny that starting early with partial implementations bears the risk of missing requirements. However, they assume that requirements will change anyway. Therefore, they recommend turning a necessity into a virtue: "Welcome changing requirements, even late in development." (Agile Manifesto) To cope with changing requirements, they propose refactoring patterns (Fowler & Beck, 2010), which address certain change requests.

The agile movement is an enigmatic and ambivalent phenomenon. There is no doubt that it has been very successful. Many companies regard it as a necessity to

implement agile processes—or at least to pay lip service to them. Who wants to blame developers that they prefer working in agile teams rather than in bureaucratic organisations? Focussing on customer needs, on communication, on organisational culture, or on testing, is certainly important. However, it seems bizarre that a software development approach, which is supposed to guide intelligent people, has facets of a religion, or, as Meyer ironically remarks: "With agile methods you are asked to kneel down and start praying." (Meyer, 2014, p. viii) Apart from that, it is the question whether agile approaches are the preferable way to cope with change. Analysing this question recommends to ask what it means to be agile. While the literature on agile approaches lacks a deep reflection of this question, it seems that it mainly emphasizes two aspects: agility as the ability to develop working solutions in time and to successfully cope with change. It is, however, not clear, what it takes to be agile. Preaching the slogan "embrace change" (Agile Manifesto) like a mantra is certainly not sufficient. No reasonable software developer will embrace changing requirements at a late stage in development (Meyer, 2014, p. 140). It is also daring to assume that "best architectures, requirements, and designs emerge from self-organizing teams." (Agile Manifesto) "Responding to change over following a plan" (Agile Manifesto) is a strange advice, too. On the one hand, it is worthless as long as it remains unclear how to respond adequately to change. On the other hand, it represents a radical criticism of one of our most valuable cultural assets, that is, the idea of rationality. However, it seems that the principles of the manifesto are not to be taken as too literal. The refusal of plans and documentation is often seen as an advice against models. But the manifesto does not explicitly offer such an advice. To the contrary, the memorandum includes a somewhat unmasking principle, too: "Continuous attention to technical excellence and good design enhances agility." Hence, the proponents of agile methods are not so insane as to recommend the abandonment of conceptual models. Furthermore, they are smart enough to identify a key success factor of agility: qualified and reflective developers. To attract those was probably one of the key drivers of the marketing campaign for agile approaches. Our brief analysis shows that apart from certain undisputed virtues, agile approaches do not represent a silver bullet to cope successfully with change. While they may supplement conceptual modelling, they are definitely not suited to replace it.

3.2 Prospects and Limitations of Induction

When it comes to flexibility, the largest information system of all times is probably the undisputed champion. Since its emergence in the early nineties of last century, it has not only grown in volume at a breath-taking pace, but also with respect to the spectrum of information and knowledge it represents. This kind of flexibility was enabled by an obvious violation of a fundamental principal of information system design. The qualification of data through types (or a schema) is of pivotal relevance for system integrity. Types serve to define semantics of data, that is, the range of possible values and a set of operations. The early versions of HTML, however,

did not support data types apart from strings. Therefore, when I first came across HTML, it appeared to me like a fall-back into the stone-age of data processing, hence, like a big mistake. However, my judgement was inappropriate, because I did not realize that the simplicity of HTML and renouncing integrity were key enablers of flexibility and growth. Virtually every piece of information can be represented as string. Setting up a simple HTML document can be done quickly (and does not require an elaborate conceptual model), and changing it is not confronted with serious integrity constraints. Of course, this kind of flexibility does not come without a price. The lack of semantics makes it impossible to use HTML for serious data processing. It is also an obstacle to reliable information retrieval. Despite the impressive power of search engines, they do not enable the specification of elaborate queries that would allow to clearly identify the intended results. On the one hand, this is caused by the poor formal semantics of the representation. If it consists of strings only, a query that aims at finding all sales prices of a certain product below a specific value cannot be expressed, because that would require relational operators on numbers. On the other hand, data types alone would not solve the problem as long as there is a huge variety of conceptualizations (e.g., of products) and names referring to these concepts.

There are various approaches to address this limitation of HTML. Semantics of web pages can be enriched by annotating web pages with words of a standardized vocabulary that refer to entries of an ontology, such as, e.g., *schema.org*, that is supposed to represent a comprehensive net of relevant concepts. In addition, formal languages such as RDF or OWL were suggested to represent content within web pages. They do not only allow for the specification of advanced concepts, but enable deduction, too. Thus, more elaborate queries and machine analysis is possible. However, as long as only a fraction of web contents is enhanced by semantic web technologies, corresponding queries are likely to produce incomplete results. At the same time, updating existing web pages, the number of which is growing every second, ex post, is a Sisyphean task that is likely to create frustration. To cope with this challenge, statistical approaches to infer schemata inductively from data have been proposed in the semantic web community (e.g. Hellmann, Lehmann, & Auer, 2008; Völker & Niepert, 2011). While these approaches did not take off so far, the revival of artificial intelligence in general, of machine learning in particular, led to an optimistic, if not enthusiastic appraisal of inductive approaches, especially those enabled by various kinds of neural networks. According to its proponents, machine learning is suited to revolutionize the process of scientific inquiry, namely the creation of theories and hypotheses (Evans & Rzhetsky, 2010; King et al., 2009). Pentland even predicts the end of the social sciences (Pentland, 2015). With the increasing availability of data on social behaviour, machine learning could be used to discover invariant patterns that would enhance the body of scientific knowledge. According to Domingos, the demanding act of knowledge acquisition and conceptualization, which is at the core of conceptual modelling will be widely automated soon: “In industry, there’s no sign that knowledge engineering will ever be able to compete with machine learning outside of a few niche areas.” (Domingos, 2017, p. 25). As a consequence, the ability to change information systems quickly would be substantially advanced.

Many will probably feel uncomfortable with this vision of replacing scientists and system analysts by machine learning algorithms. But how realistic is it? There are indeed some impressive results of machine learning approaches such as the ex-post discovery of physical laws (King et al., 2009) or machine translation. However, there are serious arguments against the exuberant optimism shown by proponents like Domingos. Conceptual modelling does not just aim at finding some structure. Instead, the linguistic structure, that is, the concepts that are required depend on the purpose the targeted program should serve. This purpose is intentional and can hardly be accounted for by inductive procedures. Furthermore, for software to be usable, it needs to be represented through concepts its prospective users are familiar with. That requires accounting for the language they speak, instead of generating artificial concepts that reflect some kind of commonalities shared by large amounts of data. The strongest argument against the automation of conceptual modelling is directed at the core of inductive reasoning. Induction depends on existing data and concepts. The future, however, may be clearly different from an extrapolation of the past, especially in times of disruptive change. This does not only relate to data, but to the concepts that will emerge to enable, and to cope with future technologies and patterns of using them. In other words, the future is a (linguistic) world that is different from the world we live in. If we assume that change is contingent, we cannot predict the future. Instead, we could develop ideas of possible future worlds that could serve as an orientation for change. Domingos fades this challenge out by emphasizing a naïve realist worldview: "We're only interested in knowledge about our world, not about worlds that don't exist." (Domingos, 2017, p. 25).

4 Prospects of Abstraction

We cannot think without concepts, and we cannot develop software without conceptual models. At the same time, the design of conceptual models is confronted with a serious conflict. On the one hand, a conceptual model should serve as a stable and reliable reference. This aspect is in favor of "freezing" concepts (Hoppenbrouwers, 2003). On the other hand, models that cannot keep up with an ever changing world may impede progress. There is no recipe to eliminate this conflict. However, there are ways to clearly relax it.

4.1 Higher Level Models

Although conceptual models may compromise the adaptability of information systems, they are also mandatory for preparing us for change. Representations in general, conceptual models in particular serve us to develop not only an understanding of the world we live in, but also of possible future worlds: „Mit Modellen machen wir uns die Wirklichkeit des Vergangenen und die Möglichkeiten des Zukünftigen zur

Gegenwart.“[1] (Mahr, 2015, p. 329). However, not every model is suited to support us with developing ideas of possible future worlds. At the same time, there is need for models that represent the present world in a sensible way to support the realization of software systems that fit today's requirements. This challenge is similar to the conflict we identified for the design of DSMLs, that is, the conflict between range of reuse and productivity of reuse. To promote range of reuse, a DSML should not be too specific. To promote productivity, it should be designed to specific purposes. How could models and modelling languages respectively look like to serve both purposes? There is, of course, no definite answer to this question. However, the structure of natural and technical languages as it has evolved in advanced societies may serve as an orientation. There are layers of concepts that built on each other. More specific languages reuse concepts defined in more general languages. At the top of this hierarchy are the concepts used in scientific disciplines followed by concepts that represent textbook knowledge. Those concepts are then refined step by step to suit more specific purposes. At the top level the range of reuse, that is, the range of possible applications, is large. At the bottom level, the range of reuse is small, but the productivity gain enabled through reuse, is high. Thus, such a hierarchical architecture of languages would not only enable to relax the conflict between range of reuse and productivity of reuse, but would also allow building information systems that satisfy current requirements and being open to change at the same time. The exemplary hierarchy of concepts in Fig. 3 illustrates this idea. The more abstract a language is, the wider is the range of possible instantiations it allows for, since it cannot only be instantiated directly into a more specific language, but also indirectly into further instances of its instances, etc. With respect to preparing for change that creates clearly more flexibility. If only small changes of a model (or a language) are required, they could be achieved on the very level of that model, e.g. by adding or modifying properties. If more comprehensive changes are needed, one could think of creating a different instance from the corresponding meta-model, that is, one would stay within the scope of the same language, but the particular concepts specified with this language would change. For more radical types of change, one could go further up the classification level and create new higher level models/languages.

Prevalent architectures of information systems are restricted to one or maybe two classification levels only. Hence, they do not allow for multiple levels of languages/models that are integrated in one model. A different paradigm is required to enable that. Multi-level modelling (Atkinson & Kühne, 2001, Frank, 2014) which extends object-oriented languages, is one approach that does not only enable modelling an arbitrary number of classification levels, but that is also supplemented with corresponding (meta-) programming languages (Clark, Sammut, & Willans, 2008). It is based on a recursive, self-reflective language architecture. Among other peculiarities, it introduces a new kind of abstraction that combines generalization and classification. The concept *Saddle SR1* in Fig. 3, for example, would inherit an attribute like *weight* from the corresponding meta-concept *Saddle*. At the same time, it would instantiate the attribute *color* into “black”.

[1] “Through models we take the reality of the past and the possibilities of the future into the present.”

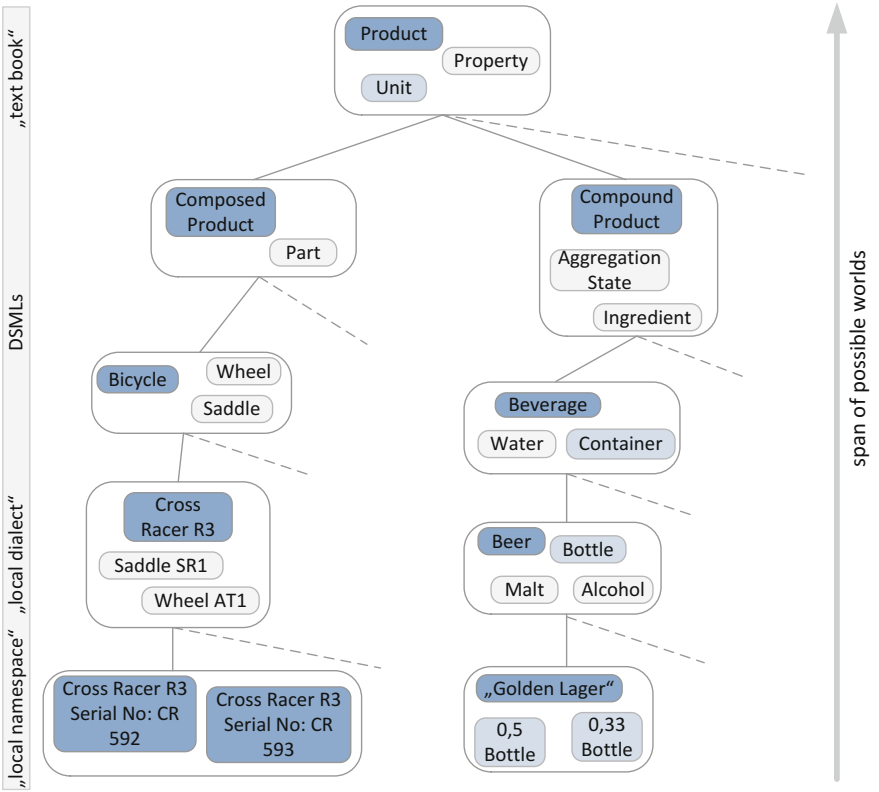

Fig. 3 Illustration of a multi-level model/language hierarchy

4.2 "Higher" Education

While increasing the flexibility of software systems is a major prerequisite of coping with change, it is not sufficient. "Technology creates possibilities and potential, but ultimately, the future we get will depend on the choices we make." (Brynjolfsson & McAfee, 2014, p. 256). It is widely undisputed that being able to make the right choices, hence, being able to benefit from the digital transformation instead of suffering from it depends chiefly on education. It is also acknowledged by many that there is need for new, more efficient ways of teaching and learning (Davidson, Goldberg, & Jones, 2010, Brynjolfsson & McAfee, 2014). It is, however, not clear what education should aim at to improve the ability to cope with change. In his conception of change, Bateson takes on the concept of motion in physics: "Change denotes process. But processes are themselves subject to "change"." (Bateson, 1972, p. 283) Hence, it makes sense to distinguish different orders of change. Based on these considerations, Bateson develops a theory of learning that suggests five different levels of learning, which are characterized by challenging previous knowledge and the pro-

cesses used to acquire it, hence, by systematically raising the level of abstraction. This kind of learning through abstraction is indeed suited to increase flexibility. If, e.g., our knowledge is restricted to conceptualize our surroundings in a specific way, we will be in trouble, if we travel other countries with different cultures. If, however, we are conscious of the fact that the world we live in, that is, the technology, the social norms, the language we are used to, is one of many possible worlds, we will be less surprised, because we know that different conceptualizations are conceivable. Abstraction of this kind will also foster communication and integration, because it tells us that beyond the differences between specific cultural peculiarities, there are commonalities shared by all cultures. If our perception of the world is not restricted to actual experience, but to an open horizon of the possible, we will be able to make sense of a yet unknown future instead of feeling lost.

This, however, is a major challenge. While we need concepts to think a possible future, we need to be aware of the fact that the future will be constituted by a language different from the one we speak. Therefore, taking the creation of possible futures to the extreme is a frightening endeavor: "The future can only be anticipated in the form of an absolute danger. It is that which breaks absolutely with constituted normality and can only be proclaimed, presented, as a sort of monstrosity." (Derrida, 1976, p. 5) There is certainly no recipe for a perfect curriculum to address this challenge, but I would hope that a university that does not only appreciate critical thinking, freedom, and originality, but provides an environment that allows taking them to an extreme, will be able to handle it.

5 Conclusions

As much as living and acting successfully in today's world requires structure, there is need for structure to cope with the digital transformation, too. Ideas of possible future worlds need to be structured, as well as processes of change. While we do not know exactly how the future will look like, we can be pretty sure that it will be penetrated by software systems. Therefore, conceptual models are of pivotal relevance, since they are not only required to build software systems, but also to make sense of them and the environment they operate in. In the end, the development of conceptual models to prepare for the digital transformation implies to challenge and eventually reform the language we use—and the way we think: "We want to establish an order in our knowledge of the use of language: an order with a particular end in view; one out of many possible orders; not the order. To this end we shall constantly be giving prominence to distinctions which our ordinary forms of language easily make us overlook. This may make it look as if we saw it as our task to reform language." (Wittgenstein, 1973, p. 132).

From a rational perspective, it does not seem satisfactory to create a possible future without some kind of evaluation—and hope. At least, it should not be worse than the presence, e.g., by destroying sense without offering alternative options for sense-making that are at least functionally equivalent (Luhmann, 1967, p. 101). At

best, we would change the concepts we use "so as to make them serve our purposes better" (Rorty, 2000, p. 25), which recommends to reflect on our purposes, maybe by following an advice Wittgenstein gave to philosophers: "Laß Dir Zeit" (take your time).

References

Atkinson, C., & Kühne, T. (2001). The Essence of multilevel metamodeling. In *«UML» 2001—the unified modeling language. Modeling languages, concepts, and tools* (pp. 19–33). Berlin, London, New York: Springer.

Bateson, G. (1972). *Steps to an ecology of mind*. Chicago: University of Chicago Press.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Becker, J., Algermissen, L., Niehaves, B., & Delfmann, P. (2005). Business process reference models for reorganizing public administrations-a case study. *Schriftenreihe Informatik, Electronic Government*, 134–142.

Becker, J., Delfmann, P., & Knackstedt, R. (2007). Adaptive reference modeling: Integrating configurative and generic adaptation techniques for information models. In *Reference modeling: Efficient information systems design through reuse of information models* (pp. 27–58).

Brynjolfsson, E., & McAfee, A. (2014). *The second machine age: Work, progress, and prosperity in a time of brilliant technologies*. New York, London: W.W. Norton & Company.

Clark, T., Sammut, P., & Willans, J. (2008). *Superlanguages: Developing languages and applications with XMF*. London: Middlesex University London.

Davidson, C., Goldberg, D., & Jones, Z. (2010). *The future of thinking: Learning institutions in a digital age*. Cambridge, MA: MIT Press.

Derrida, J. (1976). *Of grammatology*. Baltimore: Johns Hopkins University Press.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM, 15*(10), 859–866.

Domingos, P. (2017). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Penguin Books Ltd.

Evans, J., & Rzhetsky, A. (2010). Machine science. *Science, 329*(5990), 399–400.

Fettke, P., & Loos, P. (2007). *Reference modeling for business systems analysis*. Hershey: Idea Group.

Fowler, M., & Beck, K. (2010). *Refactoring: Improving the design of existing code*. Boston: Addison-Wesley.

Frank, U. (2014). Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering, 6*(6), 319–337.

Hellmann, S., Lehmann, J., & Auer, S. (2008). Learning of OWL class descriptions on very large knowledge bases. In *Proceedings of the 2007 International Conference on Posters and Demonstrations* (Vol. 401, pp. 102–103).

Hoppenbrouwers, S. J. B. A. (2003). *Freezing language: Conceptualisation processes across ICT-supported organisations*.

Kant, I. (1998). Critique of pure reason. In P. Guyer & A. Wood (Eds.), *The Cambridge Edition of the Works of Immanuel Kant*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511804649.

King, R. D., Rowland, J., Oliver, S. G., Young, M., Aubrey, W., Byrne, E., et al. (2009). The automation of science. *Science, 324*(5923), 85–89.

Luhmann, N. (1967). Soziologische Aufklärung. *Soziale Welt: Zeitschrift Für Sozialwissenschaftliche Forschung, 18*(2/3), 97–123.

Mahr, B. (2015). Modelle und ihre Befragbarkeit: Grundlagen einer allgemeinen Modelltheorie. *Erwägen Wissen Ethik, 26*(3), 329–342.

Meyer, B. (2014). *Agile!: The good, the hype and the ugly*. Springer.

Pentland, A. (2015). *Social Physics - how social networks can make us smarter*. Penguin LCC US.
Rorty, R. (2000). Universality and truth. *Rorty and his critics* (pp. 1–30). Oxford: Blackwell.
Völker, J., & Niepert, M. (2011). Statistical schema induction. In *Proceedings of the 8th Extended Semantic Web Conference on the Semantic Web: Research and Applications-Volume Part I* (pp. 124–138). Heidelberg: Springer Verlag.
Wartofsky, M. (1979). *Models: Representation and the scientific understanding* (R. Cohen, Ed.). Sprinter Netherlands.
Wittgenstein, L. (1973). *Philosophical investigations.* Wiley-Blackwell.

Ulrich Frank holds the chair of Information Systems and Enterprise Modelling at the Institute of Computer Science and Business Information Systems at the University of Duisburg-Essen. His main research topic is enterprise modelling, i.e. the development and evaluation of modelling languages, methods and corresponding tools. He is in general interested in philosophy, the art/science of computer programming, cognitive psychology, linguistics, logic and sociology.

In recent years, he focused especially on multi-level domain-specific modelling languages (DSMLs), meta-programming languages, and corresponding tools. Further areas of research include method engineering, models at run time, methods for IT management and research methods.

Ulrich Frank is on the editorial board of the journals "Enterprise Modelling and Information Systems Architectures", "Business and Information Systems Engineering", "Software and Systems Modeling", "Information Systems and E-Business Management", and the "Journal of Information System Modeling and Design". Ulrich Frank is a review board member of the Deutsche Forschungsgemeinschaft (German National Science Foundation) and the founding director of the international student exchange network IS:link.