Towards a Pragmatic Perspective on Requirements for Conceptual Modeling Methods

Boris Wyssusek[1], Johannes Maria Zaha[2]

[1]Queensland University of Technology
School of Information Systems
Brisbane, Australia
b.wyssusek@qut.edu.au

[2]University of Duisburg-Essen
Software Systems Engineering
Essen, Germany
johannes.zaha@sse.uni-due.de

Abstract. The mere unmanageable amount of techniques for conceptual modeling of information systems poses a challenge both for users of existing techniques and for developers of new techniques. However, interpreting the emergence of new conceptual modeling techniques as a result of changing architectural paradigms for information systems seems to be inappropriate. Through inspection of prevalent architectural paradigms, we show that the derivation of requirements for respective modeling techniques has been neglected frequently and an appropriate connection between architectural paradigms and conceptual modeling techniques is lacking. Motivated by these findings we argue that primarily pragmatic considerations can and should guide the development of new conceptual modeling techniques.

Keywords: architectural paradigms, modeling methods, evaluation.

1 Introduction

Conceptual modeling is generally regarded as one of the most fundamental technologies in information systems development. The importance of this technology is mirrored not only in the vast amount of pertinent academic and professional publications but also in the substantial effort that has been invested in the development of conceptual modeling methods. However, the latter phenomenon has been met by a number of criticisms, the most prominent of which can be seen in the characterization of the proliferation of conceptual modeling methods as the YAMA syndrome—Yet Another Modeling Approach [1]. The rationale behind this characterization addresses the lack of any recognizable theory driving the proliferation of modeling methods, making it difficult for the modeling practitioner to arrive at a theoretically informed assessment of the merits of old and new modeling methods and to choose a modeling method appropriate for the task at hand. The academic community has responded to this unfavorable and challenging situation with two distinct types of suggestions. On the one

hand it has called for the identification of extant theories or the development of new ones that would not only guide the development of modeling methods but also serve as a point of reference for their subsequent evaluation [2]. On the other hand academics have proposed comparative analyses of methods based on feature lists [3] or meta models [4]. However, so far none of those proposals has been proven to successfully address the YAMA syndrome.

Our work is only partially motivated by the YAMA syndrome, since we believe that conceptual modeling practitioners do not face the plethora of extant modeling methods completely unprepared. For those modelers, modeling methods are tools which help them to accomplish a task—the creation and representation of conceptual models. Thus, modeling methods are technologies, i.e., they are means to an end. However, conceptual modeling is not an end in itself, meaning that the conceptual models eventually created are not a terminal but an intermediate goal since those models—once created—serve as means for further ends. In means–ends relationships it is instrumental that the means are appropriate for the achievement of the respective ends. Thus the ends not only justify the means but ultimately define the requirements the means have to fulfill. This way, the hierarchy of goals in which lower-level goals contribute to the achievement of higher-level goals is complemented by a hierarchy of requirements that have to be met by the corresponding ends. Consequentially we believe that conceptual modeling practitioners are likely to perform means–ends analyses [5] and to choose their means in accordance with the ends to be achieved.

With our focus on means–ends relationships we adopt a pragmatic perspective which, in the given context, implies that predominantly pragmatic criteria guide the selection of conceptual modeling methods and should thus also guide their development. Yet in our research presented in this paper we do not put the selection of modeling methods center stage. Rather we are interested in learning if and how the pragmatic perspective can contribute to gaining an understanding of the genealogy of the proliferation of conceptual modeling methods and derivation of general principles for the development of modeling methods. Eventually, such principles could be used for the evaluation of the respective methods. As starting point for our pragmatic considerations we take paradigms of systems decomposition or architecture respectively (e.g., [6]). We postulate that the criteria determining systems decomposition or architecture have—from a pragmatic point of view—the most impact on the requirements conceptual modeling methods should meet. We substantiate this postulate with an analysis of the development of paradigms of systems decomposition and architecture as well as of the corresponding conceptual modeling methods. We also include some criticism, highlighting constellations where the conceptual modeling methods obviously do not meet pragmatic requirements.

With the present paper we do not intent to present a final assessment of the merits of the pragmatic perspective. Rather our goal lies in demonstrating the validity of a perspective on the development of conceptual modeling methods that has largely been neglected in previous research. Thus we contribute to the extant body of research by broadening the horizon from which the development of conceptual modeling methods has been theorized so far.

The remainder of the paper is structured as follows: In section two we introduce the pragmatic perspective on the selection and development of conceptual modeling methods in the context of architectural paradigms. Section three provides an exem-

plary analysis of four architectural paradigms and their pragmatic implications for conceptual modeling methods. We further substantiate the validity of the pragmatic perspective in section four, by showing how changes in architectural paradigms underlying information systems development entail changes in the requirements for conceptual modeling methods. We conclude our paper in section five with a short summary and an outlook on further research.

2 A Pragmatic Perspective on the Relationship between Architectural Paradigms and Conceptual Modeling Methods

Information systems are complex systems and information systems development is a complex task. In order to deal with those complexities, researchers and practitioners alike have proposed various strategies such as decomposing systems into parts (e.g., [7]) and using stage or process models for the analysis and design of systems (e.g., [8]). The strategy of system decomposition has its analogue in a general principle of problem solving: structuring larger problems in terms of a hierarchy of ever smaller problems till a level of elementary, i.e., not further subdividable problems is reached. Following this principle, the overall problem is solved by decomposing it into sub-problems in a top-down fashion and solving all the sub-problems in a bottom-up fashion.

While such a strategy appears to be quite natural and straight forward, it allows for considerable leeway posing significant problems itself. The most prevalent problem of decomposition rests with the required selection of a criterion or principle that eventually determines the structure of the decomposition. This problem has frequently been addressed in the information systems literature (e.g., [9]), but it is Parnas' [10] analysis which showed that the application of principles of decomposition has consequences beyond the structuring of the decomposition. We focus on implications for requirements that have to be met by conceptual modeling methods used in systems analysis and design, i.e., in systems decomposition and systems composition.

The number of principles that could potentially guide systems decomposition is basically infinite, and so is the number of implications that those principles may have for the requirements on conceptual modeling methods. But when we look at the history of information systems development, we can identify the occasional emergence of paradigms of systems decomposition and composition. Those paradigms are now generally referred to as architectures or architectural paradigms. According to [11], architecture is the "fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." As such, architectural paradigms not only encompass principles of composition and decomposition but also the patterns that eventually make up the structure or organization of the overall system. Thus, by taking extant architectural paradigms as starting point, we limit the number of principles of system decomposition that have to be taken into consideration. While such a constraint certainly limits the generalizability of any propositions derived from our investigation, it is in line with the pragmatic considerations that have led to the emergence of architectural paradigms in the first place, as evidenced by our analysis of prevalent architec-

tural paradigms in the next section of this paper. The emergence and development of those paradigms was usually not informed by theory, but largely driven by pragmatic considerations. It is thus consequential if we extend this pragmatism to the theorizing of the link between the principles of system decomposition underlying architectural paradigms and requirements for conceptual modeling methods used in systems architecture.

The emergence and evolution of architectural paradigms is a manifestation of analysis and design patterns that have proven successful in their practical application. As such, architectural paradigms are expressions of practical wisdom that has been generalized for a class of recurring problems in the analysis and design of information systems. The practical value of such pragmatic generalizations has already been postulated in 1968 at the famous NATO conference on Software Engineering [12], when McIlroy [13] proposed an approach to systems architecture which later became known as the component-oriented architectural paradigm. He based his argument solely on pragmatic considerations such as structuring, reuse, and productivity. Similarly, in the domain of programming languages such considerations led to the development of languages that explicitly support analogue variants of structuring programs and program development (e.g., [14]).

Understanding conceptual modeling as a technology, i.e., as a means to an end, we conceptualize the development of conceptual modeling methods from a pragmatic perspective as follows: The starting point for the development of a technology is a goal that needs to be achieved. In the case of conceptual modeling, the goal (i.e., the end) is the creation and representation of conceptual models. To achieve this goal in a systematic fashion with a predictable outcome an appropriate method is required. In the case of conceptual modeling such a method would be a conceptual modeling method. Ideally, the practical use of a method should be supported by appropriate tools. In the case of conceptual modeling such a tool would be an implementation of the respective conceptual modeling method in a conceptual modeling tool. Thus, both the modeling method and its implementation in some tool are technologies, i.e., the means to the end of conceptual modeling. This way we arrive at a conceptualization linking the task (conceptual modeling) with a method (conceptual modeling method) and a tool (conceptual modeling tool) in a hierarchical structure of means–ends relationships along which pragmatic criteria in the form of requirements are propagated from top to bottom. So far, this conceptualization does not take into consideration that conceptual modeling is not an end in itself. However, it seems that theory-driven approaches towards the development of conceptual modeling methods adopt this perspective. Yet following our pragmatic orientation we extend the hierarchical structure of means–ends relationships by including not only goals conceptual modeling serves but also requirements for conceptual modeling methods that derive from the respective goals.

Having taken our starting point in architectural paradigms, our conceptualization of the development of conceptual modeling methods in terms of an extended hierarchy of means–ends relationships enables us to establish a conceptual as well as pragmatic link between architectural paradigms and conceptual modeling methods used in systems architecture. This link finds its material expression in requirements for conceptual modeling methods that derive from the architectural paradigms. The obviously most prominent requirement for conceptual modeling methods is to be seen in the ex-

plicit methodical support of the principles of systems architecture of the respective architectural paradigm—acknowledging that architectural paradigms can be based on more than one principle, and that multiple architectural paradigms can be utilized in the creation of a single concrete architecture.

The pragmatic perspective on the development and selection of conceptual modeling methods would be incomplete if it would limit the sources of pragmatic implications. While it is consequential in the context of systems architecture to focus on the pragmatic implications that derive from architectural paradigms used in the creation of a system's architecture, other significant sources of implications should not be overlooked without prior consideration. Adopting the pragmatic perspective will help in the differentiation between pragmatic and other types of implications, thus ensuring that only pragmatic implications, i.e., requirements that are instrumental for the achievement of the respective goals are brought to bear in either the selection or the development of conceptual modeling methods; non-instrumental requirements as well as fads and fashions will be excluded. Decisions regarding the inclusion of pragmatic implications should be based on the concept of the hierarchy of means–ends relationships outlined above, thus ensuring that only pragmatic implications are considered that can be meaningfully integrated in such a hierarchy, i.e., which are instrumental with respect to the overall goal of the respective endeavor.

We wish to emphasize that we do not claim that theoretical considerations should be abandoned. As already mentioned in the introduction, our concern is the broadening of the horizon from which the development of conceptual modeling methods has been theorized so far. Pragmatic considerations are no substitute for theoretical reflections. Nevertheless, our explicit pragmatic orientation does not come without a criticism of the predominantly theory-oriented research on the development of conceptual modeling methods. As evidenced by pertinent conceptual modeling methods, theory did not play any significant role in their development. Thus it does not come as a surprise if none of the contemporary theoretical considerations, for example informed by cognitive psychology (e.g., [15]) or ontology (e.g., [16]) are able to explain the proliferation of conceptual modeling methods. Hence shifting the focus of research from theory-orientation to pragmatic considerations may prove to be a viable strategy for research policy in this area, since the theory-oriented research does not appear to appropriately address the issues pertinent to the development of conceptual modeling methods.

3 Architectural Paradigms and Modeling Methods

In this section we describe four prevalent, implementation-oriented architectural paradigms and the motivations for their emergence. The architectural paradigms entail specific views that have to be captured by conceptual modeling methods used in information systems development based on the respective paradigm. The views entailed differ with respect to the level of abstraction chosen during conceptual modeling. However, there are generic requirements for conceptual modeling methods which are independent from the level of abstraction. In the following, we will the views that are

associated with a concrete paradigm and the resulting generic requirements for conceptual modeling methods necessary to describe these views.

From a historical point of view, monolithic information systems were the first to emerge. The architectural paradigm of the *monolithic architecture* (e.g., [17]) implies that the individual parts of the system are highly interwoven and it requires strong efforts to separate the constituting parts, e.g., to change or to replace them. All parts of a software system that are necessary for the execution of a given task are included in one entity, no matter if they are directly related to the task to be performed or if they represent functionality that is independent of the task. This integration of all task-related parts allows for a centralized maintenance of the information system. Nonetheless, the maintenance can be costly since during the implementation of these systems basic principles of software engineering like separation of concerns or information hiding have been ignored frequently. Another drawback of monolithic information systems is the lack of integration between the systems for related task-areas, e.g. separate systems for procurement, production and sales.

The necessity to integrate data and processes of related task-areas was the motivation for an architectural paradigm called *three-tier architecture* (e.g., [18]). This paradigm implies the separation of the system parts in three layers: presentation layer, application layer and data layer. The presentation layer includes functionality to perform the interaction with users. Application layer and data layer comprise functionality to actually support a task and to store and retrieve data respectively. This separation is hierarchical, since there is a specific order of precedence: the presentation layer is only interacting with the application layer, which in turn is only interacting with the data layer. The separation allows for distribution as well as for centralization of functionality. An example for simultaneous realization of both principles would be the implementation of application servers for different tasks, combined with one central database within an organization and with presentation systems that exclusively support the interaction with end-users. Assuming that the implemented application servers comprise less functionality than an application server for a whole task-area, the complexity to be tackled while implementing and maintaining the information system is reduced. Moreover, the architecture fosters the flexibility when adapting the information system to changing requirements.

The three tiers of this architectural paradigm correspond to three views on the information system. For describing these views, specific modeling techniques are needed that have to fulfill certain requirements. For describing the view on the information system from the perspective of an end-user, techniques for depicting user-interfaces or techniques to capture different scenarios of usage are needed. The view on application tier and database tier requires techniques like process description languages and languages to describe the operations to be performed by the database management system.

The partition of an information system into functional elements is the main characteristic of the paradigm called *component-based architecture* (e.g., [19]). In contrast to the three-tier architecture, the criteria for separation of one component from another can be diverse. One could imagine a component that provides functionality for realizing a specific function in a user interface or a component that additionally writes the input of this interface into a database. Thus, the component-based paradigm is not dependant on the three-tier architecture. However, typical component-based architec-

tures combine these two paradigms while the application layer is divided into components that realize specific business functionality. While defining the borders of a component, the principle of modularization is applied: the components should be homogenous inside, heterogeneous to each other, and their functionality should be accessible via well-defined interfaces. They are integrated via a mediator that ensures communication between the components and between the components and their environment (e.g., with the two remaining tiers). The application of this paradigm fosters the reduction of complexity during development and maintenance, and it increases the flexibility of the resulting information system since components can be easily updated, replaced and added. Moreover it allows for reusing existing components in different scenarios, which reduces development time and costs.

The architectural paradigm of component-based architecture leads to several views on an information system. Assuming that this paradigm is combined with a three-tier architecture, there are already three extant views requiring adequate modeling techniques. Since these tiers can be further divided into components, there have to be two additional views on these components that capture the functionality provided and how to utilize this functionality. Therefore, modeling techniques to describe the component's interfaces and their functional capabilities are needed. Additionally, there is a process view on succession relationships between the executions of services offered by the components. Examples for modeling techniques to describe this view would be process description languages or temporal logics.

Modularization is also the principle behind the architectural paradigm of *service-oriented architecture* (e.g., [20]). This paradigm implies that functionality is encapsulated in services and that these services are orchestrated to realize specific business functionality, requiring a mediator that ensures communication between the services and between the services and the remaining system parts. As in component-based information systems, this concept can be combined, e.g., with a three-tier architecture, resulting in similar views and the respective requirements for modeling techniques. However, the fact that a service can be remote and only the interface and the description of its functional capabilities are used while integrating it in the information system, leads to a higher level of flexibility. As long as the interface is not changing, the implementation of the service can be changed without changing the implementation of the information system. This offers, for example, the possibility to update service implementations or replacing the information system that provides the service. The ad-hoc integration of services to fulfill certain requirements is another scenario that comes along with the service-oriented paradigm. Thus, modeling techniques for the description of the behavior of services in different scenarios and environments that allow for the generation of a view on the information system from an orchestration perspective that is dependent on the concrete usage scenario are needed.

4 Implications of Architectural Paradigms for Requirements on Conceptual Modeling Methods

In this section we use an example to demonstrate how the objective of conceptual modeling changes by altering the architectural paradigm of an information system.

Based on the changes of the objective of conceptual modeling, we have the ability to derive the changes of the requirements for modeling techniques that are needed to describe the views on an information system (defined by an architectural paradigm). These new requirements for conceptual modeling techniques are subsequently compared with the properties of modeling techniques that are associated with the respective architectural paradigm. Finally, the differences between claimed and actually existing properties are examined with respect to the extent to which pragmatic considerations are the reasons for these differences and how their influence can be used while developing new modeling techniques.

To compare two architectural paradigms, we will consult the paradigms three-tier architecture and service-oriented architecture. Since the architectural paradigm service-oriented architecture allows for a considerable leeway when designing a concrete architecture, we narrow the leeway in the following by consulting a concretization of this architectural paradigm. The service-oriented architecture considered in the following is combined with a three-tier architecture and the services in this architecture are only encapsulating functionality on the application layer. As already mentioned above, the goals for conceptual modeling of information systems can be derived from the properties of the underlying architectural paradigm. When analyzing the architectural paradigm three-tier architecture, these properties can be separated in one area for each of the three layers presentation layer, application layer, and data layer. These three views on an information system shall be described with respect to the requirements determined by the potential recipient of the respective view. In the three-tier architecture, these recipients are: the end-users for the presentation layer, the presentation layer for the application layer, and the application layer for the data layer. The requirements of these three recipients for the respective views influence the requirements for the modeling techniques to be used. Generally speaking, these requirements include the description of different usage scenarios, the description of provided functionality including process descriptions, and the description of operations to be performed by the database management system. These requirements are fulfilled by numerous modeling techniques, e.g., by the OMG standards UML communication diagram and UML activity diagram, and by SQL, to name one example for every layer.

Since we assume that our service-oriented architecture is combined with a three-tier architecture, the above mentioned basic views and respective requirements for modeling techniques also hold for this architecture. Additionally, the architectural paradigm service-oriented architecture involves new views that imply corresponding requirements for modeling techniques. In particular, there are three new views to be considered: two views describing the functionality of a service and how to utilize this functionality, and one view describing the behavior of services in different scenarios and environments. However, available modeling techniques are partly neglecting the requirements associated with these views. While there exist numerous languages for describing the functionality of a service and how to utilize it—like the Web Service Description Language (WSDL) [21], the Business Process Execution Language (BPEL) [22], the Web Services Choreography Description Language (WS-CDL) [23] or Let's Dance [24]—, these languages provide no sufficient support for the description of the behavior of services in different scenarios and environments. Thus, the change of the architectural paradigm from three-tier architecture to service-oriented architecture has a one-sided influence on the development of new modeling tech-

niques. Modeling techniques fostering the technical integration of services are far more dominant than techniques supporting the description of the usage context, which is highly relevant during the orchestration of services to form service-oriented architectures.

5 Summary and Conclusion

The proliferation of conceptual modeling methods has generally been acknowledged to be problematic, both from practical and theoretical points of view. Proposals regarding our dealings with this proliferation either call for the development of theoretical foundations or for comparative analyses of conceptual modeling methods. However, so far none of those proposals proved to address the phenomenon adequately. In response to this unsatisfactory situation we propose a pragmatic perspective, based on the conceptualization of the development and selection of conceptual modeling methods in terms of hierarchies of means–ends relationships. Taking architectural paradigms in information systems development as context, we have illustrated how the pragmatic perspective not only helps in gaining an understanding of the proliferation of conceptual modeling methods but also how this perspective can be used to derive requirements for conceptual modeling methods. However, due to space limitations we did not take non-hierarchical constellations of means–ends relationship into consideration. This is clearly a major limitation of this paper, since strictly hierarchical constellations of means–ends relationships cannot account for conflicting goals which are likely to be present in complex information systems development projects. We will address this issue in another paper.

References

1. J. L. H. Oei, L. J. G. T. van Hemmen, E. D. Falkenberg, S. Brinkkemper: The Meta Model Hierarchy: A Framework for Information System Concepts and Techniques. Technical Report 92-17, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands (1992).
2. Y. Wand, R. Weber: Research Commentary: Information Systems and Conceptual Modeling—A Research Agenda. Information Systems Research, 13, 4 (2002) 363–376.
3. N. Jayaratna: Understanding and Evaluating Methodologies. NIMSAD: A Systemic Framework. McGraw-Hill, Maidenhead (1994).
4. B. List, B. Koherr: An Evaluation of Conceptual Business Process Modelling Languages. In: Proceedings of the ACM Symposium on Applied Computing. ACM Press, New York (2006) 1532–1539.
5. A. Newell, H. A. Simon: GPS, A Program that Simulates Human Thought. In: Billings, H. (eds.): Lernende Automaten. Oldenbourg, München (1961) 109–124.
6. P. C. Clements, L. M. Northrop: Software Architecture: An Executive Overview. In: A. W. Brown (ed.): Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. IEEE Computer Society Press (1996) 55–68.
7. P. J. Courtois: On Time and Space Decomposition of Complex Structures. Communications of the ACM, 28, 6 (1985) 590–604.

8. B. W. Boehm: A Spiral Model of Software Development and Enhancement. IEEE Computer, 21, 5 (1988) 61–72.
9. N. Wirth: Program Development by Stepwise Refinement. Communications of the ACM, 14, 4 (1971) 221–227.
10. D. L. Parnas: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15, 12 (1972) 1053–1058.
11. IEEE: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Standards 1471-2000, ANSI/IEEE (2000).
12. P. Naur, B. Randell: Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee. 7.–11. October 1968, Garmisch, NATO Scientific Affairs Division, Brussels (1969).
13. M. D. McIlroy: Mass Produced Software Components. In: P. Naur, B. Randell: Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee. 7.–11. October 1968, Garmisch, NATO Scientific Affairs Division, Brussels (1969) 138–155.
14. N. Wirth: Modula: A Language for Modular Multiprogramming. Software – Practice and Experience, 7, 1 (1977) 3–35.
15. K. Siau, Y. Wand, I. Benbasat: Evaluating Information Modeling Methods – A Cognitive Perspective. In: Proceedings of the Workshop on Evaluation of Modeling Methods in Systems Analysis and Design. (1996) M1–M13.
16. Y. Wand: Ontology as a Foundation for Meta-Modelling and Method Engineering. Information and Software Technology, 38, 4 (1996) 281–287.
17. B. W. Boehm: A View of 20th and 21st Century Software Engineering. In: L. J. Osterweil, H. D. Rombach, M. L. Soffa (eds.). 28th International Conference on Software Engineering (ICSE 2006), Shanghai, China (2006) 12–29.
18. J. Edwards: 3-Tier Client/Server at Work. Wiley, New York (1999).
19. C. Szyperski: Component Software – Beyond Object-Oriented Programming. Addison-Wesley, Boston (2002).
20. M. P. Papazoglou: Service-Oriented Computing: Concepts, Characteristics and Directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (2003) 3–12.
21. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language (WSDL), version 1.1, March 2001. Available at http://www.w3.org/TR/wsdl
22. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana: Business Process Execution Language for Web Services, version 1.1, May 2003. Available at: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel
23. N. Kavantzas, D. Burdett, G. Ritzinger, Y. Lafon: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, November 2005. http://www.w3.org/TR/ws-cdl-10
24. J. M. Zaha, A. Barros, M. Dumas, A. ter Hofstede: Let's Dance: A Language for Service Behavior Modeling. In: Proceedings of the Fourteenth International Conference on Cooperative Information Systems (CoopIS), Montpellier, France, 2006, LNCS 4275.

Acknowledgements

We are thankful to the four anonymous reviewers for their many helpful comments and suggestions that have resulted in improvements of this paper.