Towards supporting malleable architecture models

Robbert Jongeling
Mälardalen University
Västerås, Sweden
robbert.jongeling@mdu.se

Federico Ciccozzi
Mälardalen University
Västerås, Sweden
federico.ciccozzi@mdu.se

***Abstract*—Engineers commonly use informal diagrams for sketching, brainstorming, and communicating initial system designs. Diagramming is accessible, new concepts can be added freely, and diagrams can be specifically adjusted to communicate at the exact right level of abstraction depending on the audience. However, the information carried by informal diagrams is most often not precise enough for automation purposes, e.g. analysis. Consequently, there is a risk that these diagrams and the knowledge carried by them are lost in slide decks or buried somewhere in a documentation space. Diagrammatic models, which, unlike informal diagrams, have well-defined syntax and semantics, can be used for automatic analysis of consistency, architectural conformance, change impact, and others. Given the preference for informal diagramming, complete adoption of modeling for architectures seems unlikely in industrial settings that we have studied so far. Therefore, we propose to support engineers to benefit from both these seemingly opposites, as well as to ensure that diagrams convey the right amount of information for the particular task at hand. We envision more malleable models, in which engineers can freely navigate along two orthogonal spectra: (1) horizontal, from informal diagramming to modeling and (2) vertical, from less to more details exposed by a diagram. This paper describes our vision for working with such malleable models, use cases for it, and research needed to make it possible.**

I. Introduction

Since in Model-Driven Engineering (MDE) 'everything' is to be considered model [1], any sketches or informal diagrams of software architectures may be seen as models of the software system. However, for the purposes of this work, it is necessary to distinguish between different types of models, depending on the way by which they are captured. Specifically, we differentiate between models that have a defined syntax and semantics and conform to modeling languages and informal diagrams that may have an agreed syntax and semantics in the minds of the engineers creating them, but not explicitly captured nor enforced. Models conform to metamodels and can be automatically parsed and transformed, whereas informal diagrams cannot be automatically processed in the same way.

A. Benefits of modeling and informal diagramming

Modeling provides unambiguous communication by enforcing conformance to a (possibly domain-specific) modeling language with agreed syntax and semantics. In addition, modeling tools may provide support for model validation and model management. Models can be transformed or used as input in automated tasks for the purpose of analyzing aspects like consistency, architectural conformance, or change impact.

Informal diagramming is trivial to adopt, users are not bound to particular tools, nor any special knowledge is required. It is free-form; that is, concepts can be added at will and it is not relevant whether or not these are formalized in a language. This freedom makes it possible to include concepts in a diagram that are not yet clear, or not so easy to model when having to strictly conform to a modeling language. Moreover, the scope of diagrams can be adjusted at will and on a need basis to the target audience and use case for the particular diagram. For these reasons, informal diagramming is a popular means of communication in industry.

B. How much modeling is enough?

The title of this paragraph is taken from one of the "grand challenges" for MDE, as described in a recent overview paper [4]. The question reflects the desire to minimize the amount of modeling and simultaneously maximize its use, for instance, for formal analysis of a design.

We observe that modeling and diagramming serve multiple purposes in architecting software: communication, design, and documentation. In other MDE applications, models may also be used for implementation but that is not our focus in this work. To effectively reuse captured knowledge for all these purposes, engineers shall be able (1) to include both informal aspects and formal modeling aspects in artifacts and (2) to adjust the amount of information and the level of detail by which information is shown in a particular artifact.

Despite the advantages of informal diagramming, we expect these diagrams to be of limited use in later development phases because there is limited ability to automatically parse and transform these diagrams. Moreover, the lack of formalized syntax and semantics of these diagrams may lead to different interpretations by different architects. Nevertheless, informal diagramming may be desired to be kept, for example, to support a common way of communicating within a domain or a company. In addition, informal diagrams are not easily replaced by more formal models since engineers may not be willing to give up the freedom informal diagramming provides.

Diagrams (be them informal or concrete model representations) are created both for early communication and for documentation to be referenced during system development and maintenance. Therefore, it should be possible to create diagrams that capture 'exactly' the right amount of information at the right level of abstraction for the target audience. The same diagram can then be "filtered" to show the appropriate amount

of detail for a specific stakeholder or task at hand. When we want to consider the created diagrams as real development artifacts, that is, we want to trust that the implementation reflects them or that they accurately document what is implemented, we need to keep the two synchronized. This requires formalized models rather than informal diagrams. Conversely, enforcing strict conformance of models to modeling languages limits engineers in expressing informal aspects, such as parts of the system that are not easily expressed in the modeling language. In addition, it may be difficult to define how a model may contain exactly the right amount of information needed for a particular communication task.

C. Malleable models

Hence, both extremes of diagramming and modeling are, in their way, too strict to adequately capture architecture knowledge and design throughout the development process; what is required for models is to be more *malleable*, like a metal that can be bent without breaking. Therefore, we propose malleable models that allow engineers to benefit both from the advantages of informal diagramming and from the benefits of strict modeling, while also supporting the adjustment of the level of detail by which models are visualized for specific uses.

Ideally, a malleable combination of informal diagramming and modeling would allow communication, design, and documentation at the right levels of abstraction, strictness, and granularity. "Flexible modeling", proposed by Guerra et al., shares some of the arguments for benefiting from both modeling and informal diagramming [6]. We describe in Section II how we can build on it. Moreover, in this paper we aim to show a vision that extends our previously published work [7] on flexible and blended [5] modeling.

II. Malleable architecture models

In this section, we detail the two orthogonal spectra comprising malleable models. An overview is shown in Figure 1, which includes examples of artifacts at different extremes along the horizontal and the vertical spectra.

A. The two spectra

The horizontal spectrum goes from informal diagramming to modeling. We purposefully do not include sketching on pen and paper or whiteboards in this spectrum, since these sketches are clearly only used for communication and are less likely to be reused in later development stages to serve as design or documentation. On the diagramming end, we can imagine tools such as Microsoft Visio, DrawIO, and others that may support a modeling syntax, e.g. for UML models, but do not enforce it. On the other end, we can imagine tools such as Enterprise Architect, IBM Rational Rhapsody, and others. Both types of tools are useful for specific architecting and engineering tasks, but, as we will argue, neither of them provides sufficient support for malleable models.

The orthogonal vertical spectrum shows the level of abstraction of the diagrams, ranging from showing less to more detail. For different communication, design, or documentation tasks, different levels of detail may be needed. To communicate effectively and take the most out of the knowledge that is captured in diagrams, we envision supporting engineers to move freely along this spectrum and thereby adjust a diagram to include exactly the type and amount of information needed for a particular use. To realise this vision, we aim to study how to bring model slicing and semantic zooming approaches such as [2] to malleable modeling.

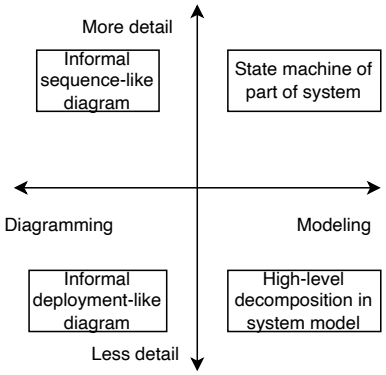

Fig. 1. Spectra of malleable modeling with example artifacts along them.

The horizontal spectrum can be considered from both directions. When starting from diagramming, we can consider settings in which informal diagrams are not used optimally for the reuse of information. By modeling parts of the information contained in the informal diagrams, this information becomes accessible for automation. The amount of information captured in models can vary between none (as in a completely informal diagram) to all (as in a model). Our horizontal spectrum represents any amount of modeling in between these extremes would contribute to malleable models. Conversely, when starting from modeling, we can consider settings in which models can be enhanced with informal information to strengthen their understandability and flexible use.

B. Envisioned way of working

To communicate adequately the right amount of information in models, while still allowing engineers to interact with them as if they were informal diagrams, we propose malleable modeling, which allows them both as well as fluid movement along and across the two spectra.

a) Fluidly navigating the spectra: We need to support fluid navigation along both axes of malleable modeling. Fluid navigation implies that models can, without hindrance, be supplemented with informal artifacts, and conversely, that parts of informal diagrams can be modeled while allowing engineers to continue informal diagramming, too. Moreover, we envision a slider-like approach to decreasing or increasing the level of detail shown in a model.

b) Partial modeling: In malleable modeling, we also envision the need to support partial and domain-specific modeling. By this, we mean that informal diagrams can be partially captured as domain-specific models without the need

to include all aspects of them. This will provide opportunities, e.g., for bottom-up meta-modeling of an existing graphical notation. The final goal is not necessarily to end up with a model but rather to optimally work with diagrams as views of malleable models.

c) Generating synchronization mechanisms: To support the continued work on diagrams along the spectrum, it is necessary to automatically generate synchronization mechanisms to keep an informal diagram consistent with its respective underlying model. An example of such a synchronization mechanism is shown in our earlier work [7].

d) Analysis: Malleable models can be used for automated analysis, since we provide an underlying model for parts of the informal diagrams.

e) Start from existing tools: To increase relevance and opportunities for adoption, we propose to focus on implementing an approach that enhances existing tools for modeling and informal diagramming of software architectures.

III. Usage scenarios for malleable modeling

In this section, we describe scenarios in which architects and engineers involved in the development of complex software-intensive systems may benefit from *malleable* modeling.

a) Scenario 1: Starting from system modeling: Here, we consider a setting at one of our industrial partners in which a system model is designed and a software implementation related to the system is manually created. Parts of the system model prescribe the structure and intended behavior of the software and simultaneously serve as documentation for the implementation, to be consulted for tasks such as analyzing the reusability of components across products. The system model itself is too complex for software engineers to navigate. For effective communication with software engineers about new functionalities to be implemented, system engineers make derivations of the system model in the form of screenshots.

In this setting, it would be beneficial for engineers to navigate fluidly across levels of abstraction to eventually end up with simplified models that carry exactly the information they need. In addition, the company works at various confidentiality levels and engineers may have different access rights to models. Consequently, the reuse of components and communication between engineers may be hampered and would benefit from simple means to limit the information shown in the model to those portions needed for communication, excluding other potentially confidential parts. Finally, allowing informal elements inside models could serve as documentation that is now only carried by the presentations (documents) created when work is handed over from system to software engineers.

b) Scenario 2: Informal diagramming and C4 levels: In this scenario, we consider a different industrial partner that has adopted the C4 model [3] to describe other parts of the architecture. To document the code, PlantUML diagrams are embedded in markdown files that are included in the same folders as the implementation. Synchronization is manual, but more easily enforced since both artifacts reside in the same version control system; for example, during code reviews, it can be checked whether the diagrams are updated according to the reviews if needed. Additional diagrams show the "container" level view of the C4 model, where containers and contained components are shown. This view is needed to show the product line setup that the company has. In parallel to the C4 diagrams, the company uses and maintains informal diagrams, similar to deployment diagrams, that specify which components are available in each product and for the platform. Engineers could benefit more from the knowledge captured in these informal diagrams if they were malleable models so that they could be used to analyze, e.g., completeness of implementation or reuse possibilities across products.

c) Scenario 3: From informal diagramming to modeling: We consider a scenario similar to Scenario 2, where engineers start from informal diagramming but want to analyze or execute their informal models as soon as possible to get insights into their designs. This cannot be done with entirely informal diagrams. If they were, on the other hand, partially formalized models, we could run early executions of those formalized portions (e.g., state machines). A realistic workflow is one in which engineers start with whiteboard drawings that are later refined via informal diagramming tools (due to remaining uncertainties and unknowns) and eventually refined partly and formalized in SysML for analysis purposes. This current scenario is initially one-way, and the information contained in the diagrams is reused minimally. Our proposal emphasizes the need to alternate diagramming and modeling perspectives and thus it is not limited to, e.g., creating a domain-specific modeling language by example.

d) Scenario 4: Domain-specific informal diagramming: Here, we consider a scenario where a graphical notation exists in informal diagramming tools and we want to support partial bottom-up metamodeling. This notation may be based on aspects that are not easily captured in most modeling languages, such as the placement of model elements or the distances between them. Our approach can assist engineers who work with informal diagrams by migrating them to malleable models that can be used as input for automated analysis tasks. At the same time, engineers are free to continue to modify the informal diagrams and add other informal aspects or extend the graphical notation without impacting the models.

e) Scenario 5: From modeling to drawing: The previous scenarios depicted the transition from informal diagramming to modeling. Now, we consider instead a scenario where formalized models, e.g. state machines, are created strictly conforming to their metamodel. Imagine that the vendor of the modeling tool wanted to extend it to support users to include more informal elements in the diagrams. The parts of the diagram that conform to the metamodel shall still be executable, and the other elements should not hinder or break executability. One of the potential added values of allowing informal modeling in this scenario is if the tool helps users migrate informal elements to formal elements that become part of the model. That is to say, the point is not simply to add a drawing palette to the tool, but to assist engineers in migrating informal elements to formalized elements, thereby

conforming to the metamodel. This scenario is of interest to users of the tool. For example, we consider a customer that has created a large number of state machines in the vendor's tool and wants to more easily describe things that are not yet clear, not yet finalized, concepts that are left as to-be-done, or perhaps elements for more detailed documentation of design choices. We have noticed that for companies that have adopted modeling quite largely, this transition (from modeling to drawing) is more interesting than the opposite along the same spectrum.

IV. Research directions

We discuss research directions for both spectra and for both directions along the horizontal spectrum, from informal diagramming to modeling and from modeling to informal diagramming. In the first direction, we can study how to generalize approaches for creating malleable models from informal diagrams, as well as how to help engineers create "better" diagrams so that they can be partially considered models. In the opposite direction, we can consider, for example, how to boost modeling tools with informal diagramming abilities and how to help engineers in migrating informal concepts to concepts conforming to the metamodel. Moreover, we can study whether it is possible to discover which model views the engineers would like to have when they add informal concepts and whether that indicates if they are missing certain capabilities in the modeling language.

Furthermore, we can study how to help engineers communicate the right information by adjusting the level of detail shown in their diagrams (be them informal or concrete model representations). We can study how to allow modelers to specify in a simple way what information to show and what to hide. Alternatively, we could consider automatically summarizing or abstracting parts of models for simplified communication. Moreover, we can study how to enhance modeling environments so that engineers can leverage ad-hoc sliders to navigate the spectrum fluidly.

V. Technical perspective

To support fluid movement along the horizontal spectrum, we propose a generalized version of the approach shown in [7], in the following directions:

1) generalizing the generation of transformations between informal diagrams and underlying textual models representing parts of these informal diagrams,
2) generalizing the generation of those transformations for multiple informal diagramming tools, and
3) simplifying the definition of the legend by allowing annotation or modifying the meta-information of elements.

To support the orthogonal spectrum of showing the right amount of detail in diagrams, we see two parallel approaches of interest. The first is to define, at the metamodel level, which elements are shown at what "zoom-level". Consider a class diagram. We could zoom out, and see only class names and associations, or zoom in to see attributes and operations. Alternatively, in cases where we are diagramming, we can imagine automated support that attempts to summarize/cluster model elements when the engineer zooms out.

VI. XM, FlexMDE, and other related work

At past instances of the MODELS conference, there have been workshops on extreme modeling (XM) and flexible modeling (FlexMDE) that are of high relevance for this topic. Research has been done in both directions, from drawing to modeling and from modeling to drawing. For instance, bottom-up metamodeling has similarities with our proposal, but does not allow the flexibility we aim for, and it is only for one direction, from diagramming to modeling. Flexisketch [8] is a tool resulting from one of the previous research efforts, although it focused only on bottom-up metamodeling and is a standalone tool instead of starting from existing tools.

The paper on "On the Quest for Flexible Modeling" [6], explicitly considers adding more capabilities for informal aspects in modeling tools. Our proposals share arguments on the importance of combining informal and formal aspects, but our proposal differs in two crucial places: (1) we want not only to support the direction from informality to modeling but also the other way around, and, moreover, facilitate back-and-forward moving along this spectrum, and (2) we include the orthogonal vertical spectrum of showing more and less information in the diagrams.

VII. Conclusion

Malleable models can be bent and worked with without breaking them. In this paper, we aim to enhance model-based software architecture by balancing the informality of diagramming and the strictness of modeling to maximize the effectiveness of software architecting by benefiting from (1) informal diagramming, for quick design and communication, and (2) modeling for automated validation and analysis. In this vision paper, we have shown use cases in which it is relevant to allow engineers to fluidly navigate along both of these spectra. Furthermore, we have outlined relevant research directions that would be necessary to be able to enhance engineering activities with malleable models.

References

[1] J. Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[2] A. Blouin, B. Combemale, B Baudry, and O. Beaudoux. Kompren: modeling and generating model slicers. *Software & Systems Modeling*, 14(1):321–337, 2015.

[3] S. Brown. The C4 model for visualising software architecture. https://c4model.com/. Accessed: 2022-12-06.

[4] A. Bucchiarone, J. Cabot, R. Paige, and A. Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, pages 1–9, 2020.

[5] F. Ciccozzi, M. Tichy, H. Vangheluwe, and D. Weyns. Blended Modelling – What, Why and How. In *Procs of MPM4CPS workshop*, 2019.

[6] E. Guerra and J. de Lara. On the quest for flexible modelling. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 23–33, 2018.

[7] R. Jongeling, F. Ciccozzi, A. Cicchetti, and J. Carlson. From informal architecture diagrams to flexible blended models. In *European Conference on Software Architecture*, pages 143–158. Springer, 2022.

[8] D. Wüest, N. Seyff, and M. Glinz. Flexisketch: a lightweight sketching and metamodeling approach for end-users. *Software & Systems Modeling*, 18(2):1513–1541, 2019.