Towards Facilitating the Exploration of Informal Concepts in Formal Modeling Tools

Martin Gogolla
University of Bremen
Bremen, Germany
gogolla@uni-bremen.de

Robert Clarisó
Universitat Oberta de Catalunya
Barcelona, Spain
rclariso@uoc.edu

Bran Selic
Malina Software Corp.
Ottawa, Canada
selic@acm.org

Jordi Cabot
ICREA
Barcelona, Spain
jordi.cabot@icrea.cat

***Abstract*—This contribution proposes to apply informal ideas for model development within a formal tool. The basic idea is to relax the requirements expressed with particular modeling language elements and allow developers to dynamically customize the level of formality in a visual and intuitive way. For UML and OCL class models, the requirements for usual object typing, role typing, role multiplicity, attribute typing and constraint satisfaction are relaxed in order to achieve flexible object models. The long-term aim is to support flexible, iterative model development with qualified tool feedback.**

***Index Terms*—UML class model, UML object model, OCL constraint, flexible development process**

I. Introduction

Many software-intensive systems of the future, such as those that control complex physical processes (*e.g.*, "smart" cities or autonomous vehicles), involve close interaction with the physical world. That world, unfortunately, is often eminently unpredictable. No matter how carefully we analyze the possibilities, unexpected events and behaviors (*i.e.*, the infamous "unknown unknowns" [1]) are likely to occur after the system has been deployed. This inability to fully predict the full set of possible inputs to a system has inspired new research initiatives related to design in the presence of uncertainty [2]. How should we design software in such circumstances? The view in [3], suggests that we may have to shift our focus "from correctness to utility" and "from precise to approximate". That is, we may have to abandon the long-sought (but never fully realized) ideal of full logical correctness with a resulting loss of full determinism. This places the problem of software design in such circumstances in the interstices between the informal and the formal. In this paper, we explore an approach that involves a continuum between the two.

In our context, formal modeling environments [4] such as USE [5]–[7] or Alloy [8] provide quality-related features to software developers. First, they enable developers to *verify* correctness properties about models, *e.g.*, whether the model contains contradictory constraints. Moreover, developers can *validate* whether a model fulfills their expectations by constructing (or exploring) sample instantiations or traces in which relevant properties and integrity constraints can be checked.

This work is partially funded by ECSEL JU projects AIDOaRt (grant agreement No 101007350) and TRANSACT (grant agreement No 101007260), and the Spanish government (LOCOSS project - PID2020-114615RB-I00).

Nevertheless, these verification and validation features do not come for free: they require the model to be *completely* and *correctly* specified using a formal notation (*e.g.*, a textual notation based on UML[1] and OCL[2] in USE or the Alloy notation for Alloy). As a result, it is only possible to benefit from verification and validation in the final stages of the MDE process, when all the information on the system to be developed is available, and all design decisions have been made. Moreover, if an existing model needs to be updated for some reason, it is necessary to define all details about the change precisely before regaining the ability to verify and validate it.

In this paper, we propose a metaphor that can be used to introduce flexibility in a formal modeling tool and to let developers control and adjust the desired level of (in)formality in their models. The degree of (in)formality is presented as a set of *sliders* controlled by the developer, providing a visual and intuitive representation. We illustrate how this metaphor can be incorporated in the USE specification environment for selected modeling features.

Thanks to this slider metaphor, developers can relax the formality requirement and take advantage of verification and validation in scenarios of early prototyping, design space exploration or model evolution. For example, suppose that the developer decides that an integrity constraint should be temporarily considered as a soft constraint, *i.e.*, instantiations that violate the constraint are tolerated. In that case, the model finder (the component for constructing instantiations conformant to the model) should switch its behavior from *satisfiability* (try to fulfill all constraints) to *max-satisfiability* (try to fulfill all hard constraints and as many soft constraints as possible). Furthermore, the developer could aim for even more diverse constraints and test what type of instantiations are produced if the integrity constraint is ignored during the model finding process by setting the slider accordingly.

Besides, it also enables developers to operate with "informal" models and iteratively increase the level of formality until a final model is reached. Overall, we believe that this proposal improves the user experience of formal modeling tools, creating new usage scenarios. User experience is an open

[1]Unified Modeling Language. https://www.omg.org/spec/UML
[2]Object Constraint Language. https://www.omg.org/spec/OCL.

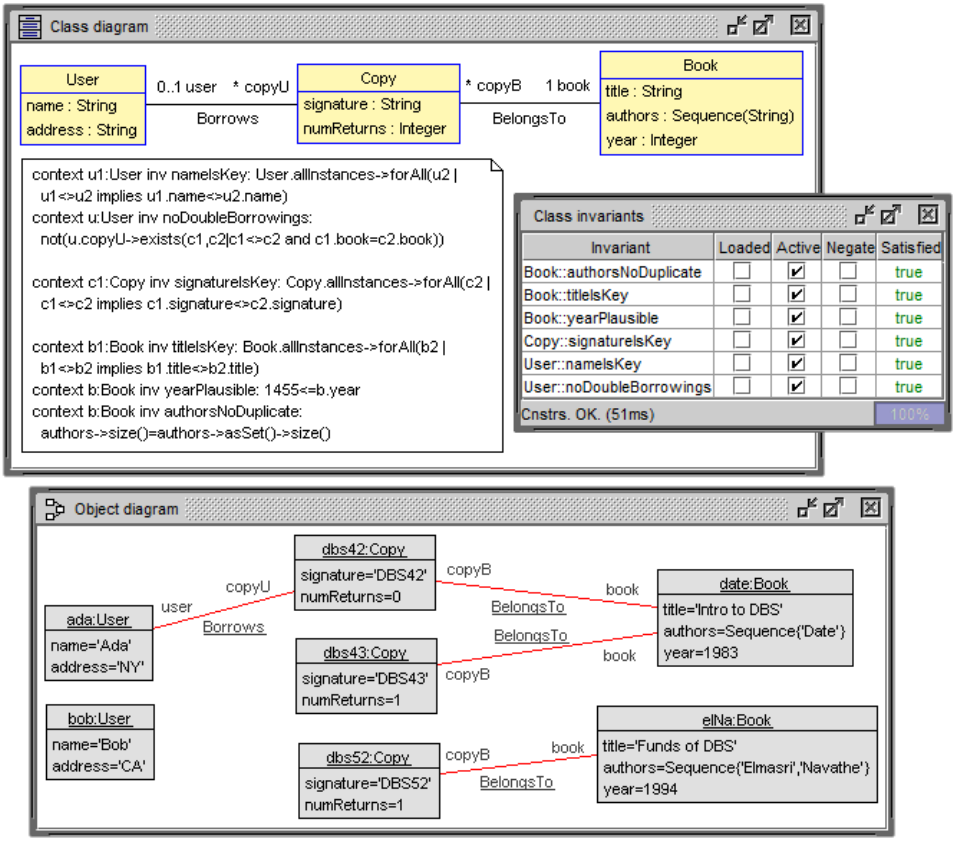

Fig. 1. Example class model with invariants and object model (instantiation).

concern both for model-driven engineering tools in general [9] and formal methods tools in particular [10].

Paper organization. The remainder of the paper is organized as follows. Section II presents a motivating example. Then, Section III discusses how flexibility is introduced in the modeling process and which model elements are involved. The role of this flexibility in the development process is discussed in Section IV. Finally, Section V discusses related work and Section VI concludes.

II. Motivating Example

Before discussing our proposal along the modeling features that we want to relax in a flexible development process, we motivate our work by introducing a simple, mainstream UML and OCL modeling example and demonstrate how the relaxation will be formulated in terms of a hypothetical user interface and in terms of the resulting, more relaxed, "informal" object models. Figure 1 presents a small UML class model together with invariants for a library system. A valid object model together with the evaluation of the invariants is shown as well. As indicated in the class invariants window, invariants can be active or inactive.

Figure 2 suggests a user interface for determining the nature of (in)formality that the developer chooses to work with in the next development steps. Basically, in the user interface sliders are provided to control the degree of (in)formality. The sliders concern the typing of objects, attributes and roles, the association multiplicities and the constraint satisfaction. The intention is that, depending on the chosen settings, more relaxed and "informal" object models are allowed.

Figure 3 displays (a) an object model that conforms to the shown transformed class model in the upper part of the figure and (b) the evaluation of the invariants in which one invariant is inactive. That is, the original class model was transformed according to the slider settings to a class model in which a most general class `Object` has been introduced, the associations have been lifted to the new class `Object`, the attributes are now typed with `OclAny` and for collection-valued attributes the abstract class `Collection(OclAny)` is used. Please note that in formal terms, the object model displayed in the lower part of Fig. 3 is a formally valid model for the new class model. This object model is understood to be a relaxed, "informal" object model of the original class model. The transformation process is assumed to be realized internally without developer interaction. Meanwhile, the original model is preserved, but every change in the sliders generates a new transformed model, including a "no-transformation" option when all the slider settings are set to strict.

Five parts have been highlighted in the object model from Fig. 3. These parts correspond to the modeling features that have been relaxed in Fig. 2 by sliders.

(a) Object typing: The object `cyd` is typed through the

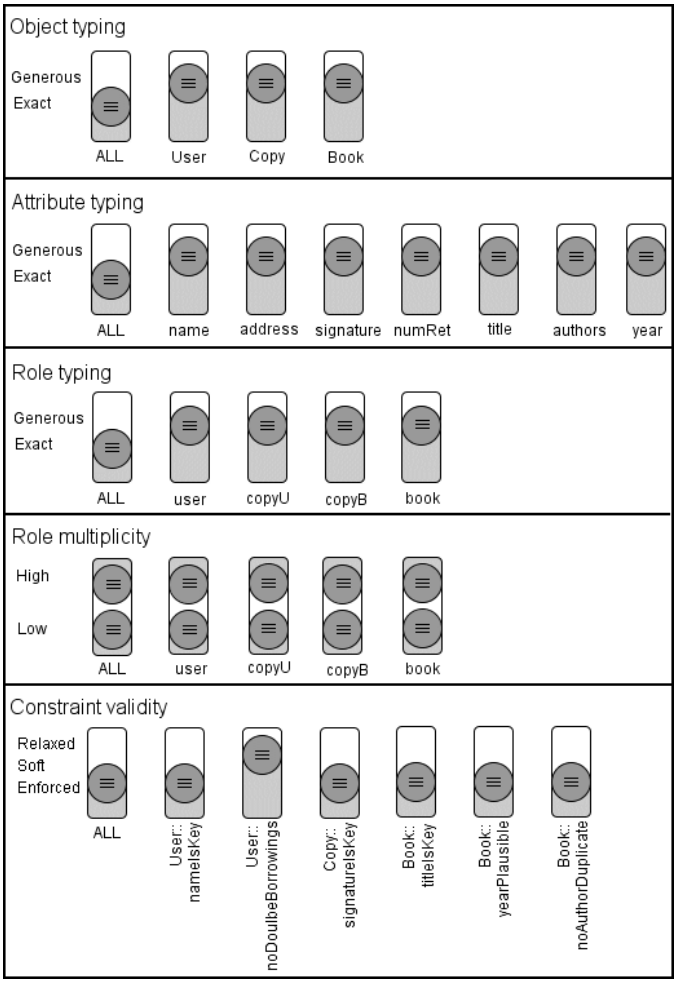

Fig. 2. Hypothetical user interface with sliders setting (in)formality degree.

general class `Object`. It participates in the role `copyB` in the association `BelongsTo`. In the link, the `Copy` object `dbs42` plays the role `book`. This highlighted part shows the relaxation for typing objects that is made possible through the introduction of class `Object` and the lifting of the associations.

(b) Constraint satisfaction: The highlighted link makes the invariant `noDoubleBorrowings` invalid: The user `ada` is borrowing (in an "unfair" way) two copies of the same book. This shows the relaxation of constraint satisfaction.

(c) Multiplicity satisfaction: The highlighted link is a second `BelongsTo` link for the copy `dbs52` which now belongs to the two books `date` and `elNa`. This violates the original multiplicity `1` of role `book` in `BelongsTo` and shows the relaxation of the original multiplicities.

(d) Role typing: Originally the `Borrows` association was defined between classes `User` and `Copy`. The `Borrows` link in this part is between classes `User` and `Book`. This highlighted part shows a relaxation for role typing.

(e) Attribute typing: Originally the attribute `authors` was typed as `Sequence(String)`. Now it gets a value of type `OrderedSet(OclAny)`. This relaxation for attribute typing happens due to the newly introduced type `Collection(OclAny)` for attribute `authors`.

III. Modeling elements for relaxation

Our proposal for handling (in)formality is generic, nevertheless we now turn to show how it can be realized in USE in which the level of formality of a UML and OCL model can be relaxed according to several perspectives. In this section we discuss the semantics and realization of relaxing the following language features: *class typing*, *attribute typing*, *role typing*, *multiplicity satisfaction* and *constraint satisfaction*.

a) Class typing: Flexible class typing refers to the ability to define and use objects of an unknown, undecided or purposely undefined type during validation. On the UI, the slider `ALL` controls the introduction of a most general class `Object` for representing objects effectively without type. A slider on position `Generous` for a single classes expresses that any attribute or any role resulting in this respective class can be substituted with an untyped object from class `Object`.

b) Attribute typing: The attribute value may be taken from `OclAny` or `Collection(OclAny)` for collection-valued attributes. An attribute slider controls this option.

c) Role typing: Untyped role substitution may become possible by lifting associations to the most general class `Object` and by explicitly controlling the role substitution with a slider. Not necessarily all associations have to be lifted (as done in our example), some associations and roles may remain "exactly" typed.

d) Multiplicity satisfaction: In each single association, through sliders, lower bounds may be decreased and upper bounds may be increased. Switching between single-valued (`0..1`, `1`) and multi-valued (e.g., `0..*`, `1..*`) multiplicities may lead to the fact that an OCL constraint cannot be evaluated which will be reported to the developer. Manipulating "High" and "Low" both change the numerical range setting, e.g., from `0` to `1` to `2`. "Low" will always stay under "High".

e) Constraint satisfaction: We provide three settings in the sliders for constraints: (1) 'Enforced' for a usual hard constraint that is always satisfied; (2) 'Soft' for a constraint that is satisfied only if that is possible; relevant in cases when our Model Validator automatically constructs object models; (3) 'Relaxed' completely ignores the constraint.

We propose to have combined relaxation control (`ALL`) for all elements in each single group. Furthermore, we suggest an overall relaxation control for the complete model. For larger models a manageable grouping mechanism must be established, e.g., it will be unpractical to handle all attributes (as we have done in our toy example) as a single group. Relaxation control for language elements must be grouped by class and/or package.

Regarding correctness, intuitively each slider either removes a conformance requirement of object diagrams with respect to class diagrams (*e.g.*, satisfying integrity constraints) or sets a weaker requirement (*e.g.*, less restrictive multiplicities). A formal proof of correctness is out of the scope of this paper.

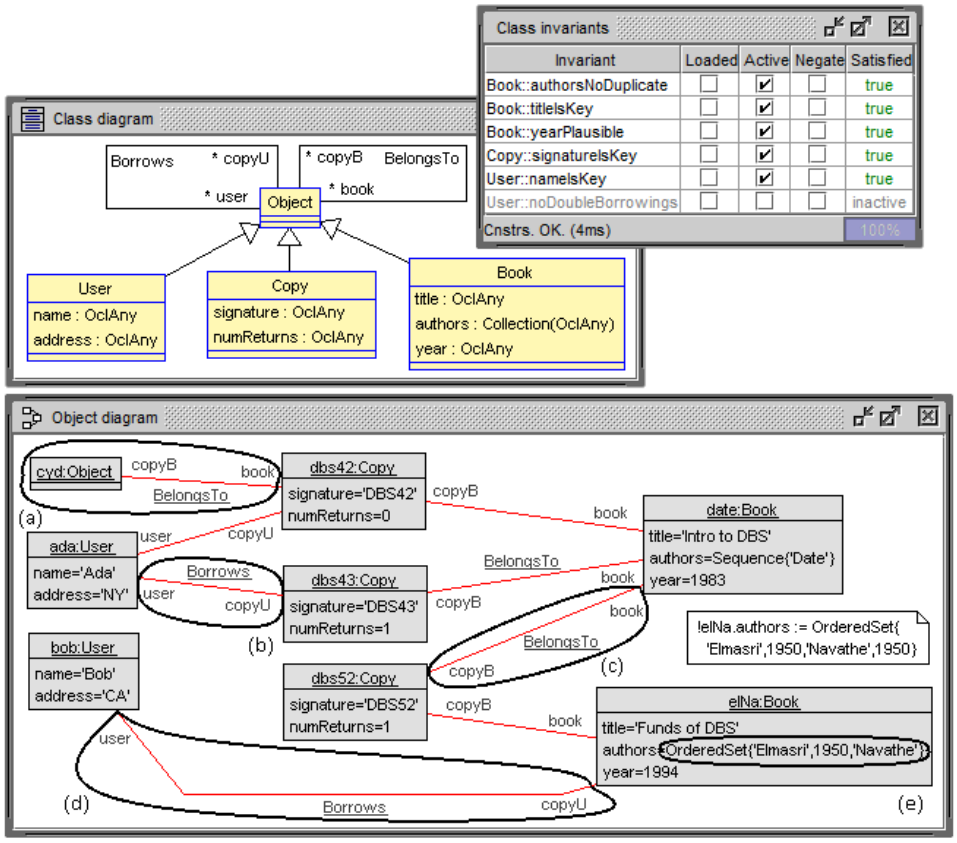

Fig. 3. Transformed class model with "informal" object model.

IV. Embedding of relaxation into the development process

So far, this contribution has been considering that a generic formal model is relaxed in order to obtain "informal" model instantiations. In a practical development process, one will take a different approach, namely one wants to start with a high degree of informality and in an iterative process achieve more and more formal descriptions. We have here taken first the former view (from formal to informal), because we wanted to point out that informal models can be achieved when working in a formal modeling tool. The tool support that we envision has to ensure that the latter (from informal to formal) is made possible by offering the right tool options.

Our overall aim with relaxation is to improve the development process. We want to facilitate iterative development steps that lead from loose (class) models to sharper ones (e.g., with iteratively introduced integrity constraints, sharp multiplicities and more specialized object, attribute and role types). A high degree of flexibility for developers should enable them to let their ideas flow without tool complaints, e.g., about missing or unsatisfied typing details. Support for development with imperfect, even inconsistent intermediate models contributes to flexibility as well. Qualified tool feedback through automatic object model construction for checking implications of design decisions should be supported as well, see e.g., [6], [7].

V. Related work

A. Flexible model-driven engineering

Most notations for software modeling require the developer to produce *complete* and *correct* models upfront. Nevertheless, there are benefits in relaxing these requirements [11], [12], which are studied by a family of methods known as *flexible modeling* [13]. Flexible models may include incomplete or unspecified information, either because it is unknown or because the developer is interested in postponing some decisions until a later stage. This flexibility enables rapid prototyping and design space exploration [14]; supports modeling in the presence of uncertainty [2]; improves the reusability of the modeling artifacts [15]; and facilitates collaborative modeling [16].

Flexible modeling approaches are concerned with providing modeling notations that support incomplete or uncertain informations, *e.g.*, generic metamodels [17] or incomplete/inferred typing information [15], [18], [19]; tool infrastructures that support the definition of flexible models; and the theoretical foundations that ensure the conformance between a concrete model and the flexible model from which it was derived.

Among existing works, the most related to ours is [13], which describes the requirements of KITE, a modern flexible modeling tool. Among these requirements is the ability to customize inconsistency tolerance. In contrast, this paper proposes a particular interface for managing this feature as

well as its integration in a formal design tool, USE, in a way that leverages its formal analysis capabilities.

B. Uncertainty in software modeling

Several works have classified the different types of uncertainty in software modelling [1], [2]: behavior (dynamic evolution of the model is not fully specified), belief (stakeholders have different opinions about the model), measurement (inaccurate sensors may yield imperfect data for attributes), occurrence (it is unknown whether a certain observation exists or not), design (several design alternatives may be possible) and spatiotemporal (location and time of an event is unknown).

In this paper, we only consider *design-time uncertainty*, *i.e.*, some design decisions about the model are still open. An existing approach for dealing with design-time uncertainty is *partial models* [20]–[23], which include variation points where several alternative design decisions are being considered. Thus, a partial model describes a *set of candidate models* that collapses into a single model once all design decisions are made (*i.e.*, one alternative per variation point is selected).

In contrast with our approach, partial models require explicitly defining the variation points and alternatives. On the other hand, we provide the developer with the freedom to select in which aspects of the model flexibility is required, providing a visual an intuitive mechanism to adjust this choice dynamically. However, our approach does not support the fine-grained definition of alternatives offered by partial models.

C. Informal modeling in formal modeling tools

Several modeling tools can dynamically adjust the degree of formality, a process also known as *relaxation/tightening* [24]. For example, USE supports deactivating class invariants [6]. In JSMF [25], a modeling framework based on JavaScript, developers can deactivate checks for attribute types, role types, and class multiplicities. Nevertheless, the metaphor and interface we propose for this adjustment is innovative.

Other strategies for improving flexibility in the modeling process have been proposed, such as inferring models from example instantiations or fragments, e.g., [26], [27].

VI. Conclusion

We have proposed a metaphor for controlling the level of (in)formality in formal design environments. Developers can select, for diverse aspects of the model, the degree of strictness with which formality is enforced. In this way, "informal" instantiations that are not fully conforming to the model can be tolerated and still provide verification and validation features. An example of its application is proposed in the USE specification environment for UML models. This flexibility opens new usage scenarios for formal modeling tools.

As future work, we plan to implement this feature in USE. Moreover, we are interested in computing the *least informal* setting that should be selected in order to tolerate a set of instantiations of interest, as this information highlights potential issues. Finally, we want to look into supporting the relaxation of other modeling elements, such as whole-part relationships or operation contracts.

References

[1] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: a conceptual model," in *ECMFA*. Springer, 2016, pp. 247–264.

[2] J. Troya, N. Moreno, M. F. Bertoa, and A. Vallecillo, "Uncertainty representation in software models: a survey," *SoSyM*, 2021.

[3] D. Garlan, "Software engineering in an uncertain world," in *Workshop on Future of Software Eng. Research (FoSER)*, 2010, pp. 125–128.

[4] J.-M. Bruel, S. Ebersold, F. Galinier, M. Mazzara, A. Naumchev, and B. Meyer, "The role of formalism in system requirements," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[5] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27–34, 2007.

[6] M. Gogolla, F. Hilken, and K.-H. Doan, "Achieving Model Quality through Model Validation, Verification and Exploration," *COMLAN*, vol. 54, pp. 474–511, 2018.

[7] L. Burgueño, J. Cabot, R. Clarisó, and M. Gogolla, "A systematic approach to generate diverse instantiations for conceptual schemas," in *ER*, ser. LNCS, vol. 11788. Springer, 2019, pp. 513–521.

[8] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[9] S. Abrahão, F. Bourdeleau, B. Cheng, S. Kokaly, R. Paige, H. Stöerrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *MODELS*. IEEE, 2017, pp. 229–236.

[10] S. Krishnamurthi and T. Nelson, "The human in formal methods," in *Int. Symposium on Formal Methods (FM)*. Springer, 2019, pp. 3–10.

[11] B. Nuseibeh, S. Easterbrook, and A. Russo, "Making inconsistency respectable in software development," *JSS*, vol. 58, no. 2, pp. 171–180, 2001.

[12] B. Selic, "What will it take? a view on adoption of model-based methods in practice," *SoSyM*, vol. 11, no. 4, pp. 513–526, 2012.

[13] E. Guerra and J. de Lara, "On the quest for flexible modelling," in *MODELS*, 2018, pp. 23–33.

[14] H. A. A. Alfraihi and K. C. Lano, "The integration of agile development and model driven development: A systematic literature review," *MODELSWARD*, pp. 451–458, 2017.

[15] J. D. Lara and E. Guerra, "A posteriori typing for model-driven engineering: Concepts, analysis, and applications," *ACM TOSEM*, vol. 25, no. 4, pp. 1–60, 2017.

[16] M. Franzago, D. Di Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: a classification framework and a research map," *IEEE TSE*, vol. 44, no. 12, pp. 1146–1175, 2017.

[17] L. Rose, E. Guerra, J. De Lara, A. Etien, D. Kolovos, and R. Paige, "Genericity for model management operations," *SoSyM*, vol. 12, no. 1, pp. 201–219, 2013.

[18] A. Zolotas, N. Matragkas, S. Devlin, D. S. Kolovos, and R. F. Paige, "Type inference in flexible model-driven engineering," in *ECMFA*. Cham: Springer, 2015, pp. 75–91.

[19] A. Zolotas, R. Clarisó, N. Matragkas, D. S. Kolovos, and R. F. Paige, "Constraint programming for type inference in flexible model-driven engineering," *COMLAN*, vol. 49, pp. 216–230, 2017.

[20] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *ICSE*. IEEE, 2012, pp. 573–583.

[21] R. Salay and M. Chechik, "A generalized formal framework for partial modeling," in *FASE*. Springer, 2015, pp. 133–148.

[22] M. Famelis, J. Rubin, K. Czarnecki, R. Salay, and M. Chechik, "Software product lines with design choices: reasoning about variability and design uncertainty," in *MODELS*. IEEE, 2017, pp. 93–100.

[23] M. Famelis and M. Chechik, "Managing design-time uncertainty," *SoSyM*, vol. 18, no. 2, pp. 1249–1284, 2019.

[24] R. Salay and M. Chechik, "Supporting agility in MDE through modeling language relaxation," in *Extreme Modeling Workshop (XM)*, 2013, p. 21.

[25] J.-S. Sottet and N. Biri, "JSMF: a flexible JavaScript modelling framework," in *Workshop on Flexible Model Driven Eng. (FlexMDE)*, 2016.

[26] A. Kästner, M. Gogolla, and B. Selic, "From (imperfect) object diagrams to (imperfect) class diagrams: New ideas and vision paper," in *MODELS*, 2018, pp. 13–22.

[27] J. J. López-Fernández, J. S. Cuadrado, E. Guerra, and J. De Lara, "Example-driven meta-model development," *SoSyM*, vol. 14, no. 4, pp. 1323–1347, 2015.