The Conformance Relation Challenge: Building Flexible Modelling Frameworks

Nicolas Hili
School of Computing, Queen's University
Kingston, Ontario, Canada
Email: hili@cs.queensu.ca

Jean-Sebastien Sottet
Luxembourg Institute of Science and Technology
5 avenue des Hauts Fourneaux,
Esch/Alzette, Luxembourg
Email: jean-sebastien.sottet@list.lu

***Abstract*—Recent works in Model-Driven Engineering (MDE) advocate the need for more flexibility in modelling environments by relaxing the conformance relation between a model and its metamodel. Most of the current works on flexibility only focus on the creation of the concrete syntax, leaving out important considerations about the development of the abstract syntax. Different approaches to relaxing the conformance relation between a model and a metamodel exist, but no one has so far endeavoured to build a unified vision of the different dimensions that flexibility could comprise. This paper is an attempt to define such a unified view of flexibility in MDE, focusing on the conformance relation problem.**

I. Introduction

The development of Domain Specific Modeling Languages (DSMLs) traditionally follows a top-down approach where both abstract and concrete syntaxes are respectively formulated and sketched, before user models can be created [1]. However, in some contexts (early insights, problem discovery, evolution, etc.), such rigid approaches are no longer tenable. In order to give enough freedom to the modeler, models should no longer be limited to a predefined structure.

Recent works are promoting the need of bringing flexibility into modelling frameworks and tools. However, the semantics of "flexibility" remain fuzzy, and flexible tools address it in a variety of forms and degrees. It goes beyond pure tooling concerns and affects different aspects of the modelling language: syntaxes, semantics, pragmatics, and usage. For instance, concrete syntax flexibility is of importance but it is widely addressed. We believe the conformance relaxation is a complex issue and largely undervalued compared to existing work in the literature focusing on features such as collaboration or sketching diagramming whiteboards, for instance see [2]–[5]. Thus, we focus in this article on the conformance relation problem, based on our previous works [6], [7].

To our knowledge, no one has so far built such a unified view, leaving users and developers who want to choose, compare, or evaluate flexible modelling tools that fit their needs. This paper presents a preliminary attempt to structure such a view of flexibility. We hope that this view can be used as a guideline for users, developers, and toolsmiths who want to choose or build flexible modelling tools and frameworks.

Relaxing the conformance relation is not enough and has to be balanced with the need of enforcing validation when the modelling language is stabilized. Therefore, it is important to answer the relevant questions about *when* and *what* we need to relax. Such questions are left out in existing flexible modelling tools, so the attention is drawn to the graphical aspect and collaborative features. It requires building a view on the different dimensions that flexibility could comprise.

This paper is structured as follows: Past efforts for introducing flexibility into modeling frameworks are detailed in Section II; Section III is an attempt of defining a view of flexibility focusing on the conformance relation problem; With respect to these dimensions, we have compared some existing approaches relaxing the conformance with the two reference implementations [6], [7] in Section V; Section VI concludes the paper.

II. Background

A. Approaches to Relaxing the Model Conformance Relation

Traditionally, Model-Driven Engineering (MDE) is promoting the use of models that conform to a given and well defined metamodel. The conformance relation then has to be strictly enforced in order to exploit the models correctly [8]. However, in certain situations e.g., early exploratory, the strict conformance relation is not always desired and has to be relaxed.

Several solutions have been proposed to relax the conformance relation. The approaches of [9], [10] propose to decouple the metamodel and model conception/creation and evolution. More precisely, in [10] the authors decouple the two functions of a metamodel regarding a model: it has a templating function (for instantiation) and its typing function. In [9], the objective is to allow the model to evolve without considering the problem of conformance. It uses an intermediary representation called GIMM [9], which is a generic representation (domain agnostic) in which every model can be expressed. Then, metamodel could be updated to ensure that conformance is correct.

The way of relaxing the conformance relation is explored in [11]. The relaxed models are expressed in a graphical language where constraints can solely be specified by the graphical tool. In this work, the authors proposed a solution to tighten the conformance relation: identifying inconsistencies and applying repairing algorithms.

In [12], the authors addressed the notion of flexibility through type polymorphism. They allow a model to be conform to multiple types, or metamodels. It allows the models to be reused in different contexts, e.g., different interfacing DSLs, by modelling different tools.

Other approaches address specific issues such as model evolution or problem discovery. Approaches for model/metamodel co-evolution [13]–[15] help to withstand metamodel evolution and to automatically or semi-automatically adapt models. These techniques usually rely on the identification of some transformation patterns: creation of new concepts, deletion of existing concepts, addition to some attributes, etc.

Bottom-up metamodeling [16] is another approach which consists in inferring what the metamodel should be, regarding a set of existing models. It allows for the creation of a model independent of the metamodel definition. By analogy, we can compare this approach with NoSQL databases for which schemas do not have to be defined.

B. Emerging Frameworks and Technologies for Flexibility

Some recent research efforts have brought flexibility into modelling frameworks, either by relying on and extending existing modelling languages (such as Ecore), or by using new technologies and languages.

Among them, some endeavoured to bring capabilities of Eclipse Modeling Framework (EMF) [17] into web-based environments. Some attempts are direct ports of EMF into Javascript [18] using the EMF prototypes. Other interesting frameworks include Ecore.js [19], which is developed in JavaScript and available through NodeJS, and EMF-Rest [20], [21], which intends to bring EMF capabilities through a REST API.

The muddle approach [22], based on Epsilon [23], proposes to use an intermediate representation (called *Muddle*) of Ecore models in order to work on some uncertainty. It allows working with models (e.g., using model to model transformation) that partially conform to a given metamodel.

Due to its lightweight notation, JavaScript Object Notation (JSON) is used in MDE over XML for defining models and metamodels. *Moddle* [24] is a language for designing models using JavaScript and JSON. It was developed for BPMN.io [25]. Metamodels are defined dynamically, however, models cannot be created without defining the metamodel first. Besides, models cannot be validated and Moddle provides a limited model coverage.

MoDiGen [26] uses JSON to address scalability. Metamodels and models are defined using JSON. However, no implementation appears to exist as no programming language is mentioned. Besides, there is no mention about how models are concretely "instantiated" from the metamodel definition in JSON.

III. Dimensions of Flexibility

In our previous work [6], [7], we addressed the flexibility according to two orthogonal and complementary dimensions (see Fig. 1). To these dimensions we add the semantic dimension related to the "ontological level" of information described in the metamodel.

The first dimension (Time/Phase of flexibility) is the process view on model flexibility. It is related to the maturity and alignment of the models/metamodels during the modelling process. In early phases of the development, modelers are exploring the problem scope and nature. Thus, free modelling is favored as a way of prototyping, using sketching tools for example, in order to increase productivity. This allows for defining models even before any metamodel is available up to the traditional MDE phase (where a model is said to conform to the metamodel). In late phases of the development process, production, strict modelling should be encouraged, as metamodels are stabilised and the focus should be put on a strict definition of the models conforming to the metamodels. The intermediate point of the axis represents phases where the progressive consolidation of models/metamodels triggers possible problems that require a convergence of the definitions. We have defined a reference implementation of this dimension in Fleximeta[1].

The second dimension (Conformance/Granularity Level) is the structural/object view on model flexibility. It is related to the tuning of flexibility regarding model elements. For the first point of the axis, the model element could have no reference metamodel (no conformance check, no type). The model is then fully flexible. For the second point, the model is still flexible but is related to a metamodel element (i.e., a first kind of typing). he third point (partially flexible) encompasses all the constraints releases on conformance relation: it can be set on specific elements (such as a reference cardinality, attribute value type, etc.). The last point (full conformance) is the classical definition of conformance. This dimension has for reference implementation the JSMF framework[2].

The third dimension deals with the gradient of the *specificity* of semantics. During the elaboration of a language the semantics (semantic mapping, semantic domain) may not be clearly defined first (fuzzy language semantic as defined in [27]). Indeed, it should be the result of discussions amongst the language stakeholders. The level of granularity of the semantic domain could also be more or less precise like in ontologies [28]. The four levels of semantics go from agnostic models (i.e., core level ontologies) to application (or problem) specific models. The intermediary levels depict the domain specificity (e.g., a metamodel for modelling wireframes in user interface design) or multi-domain (e.g., a metamodel for covering the engineering of user interfaces). The level of knowledge genericity (e.g., domain agnostic) expressed in the metamodels, allows for a certain flexibility: the conformance relation is thus relaxed by a broader domain coverage.

The related work covers one or more graduations of the dimensions. For example, GIMM [9] relies on a higher level of *semantic* genericity when it is needed to be less strict (i.e., tolerant modelling phase). A comparison of tools dealing with the abstract syntax flexibility is proposed in the next section.

[1]available at: https://github.com/nicolas-hili/FlexiMeta [7]

[2]source and documentation available at: https://js-mf.github.io/ [6]

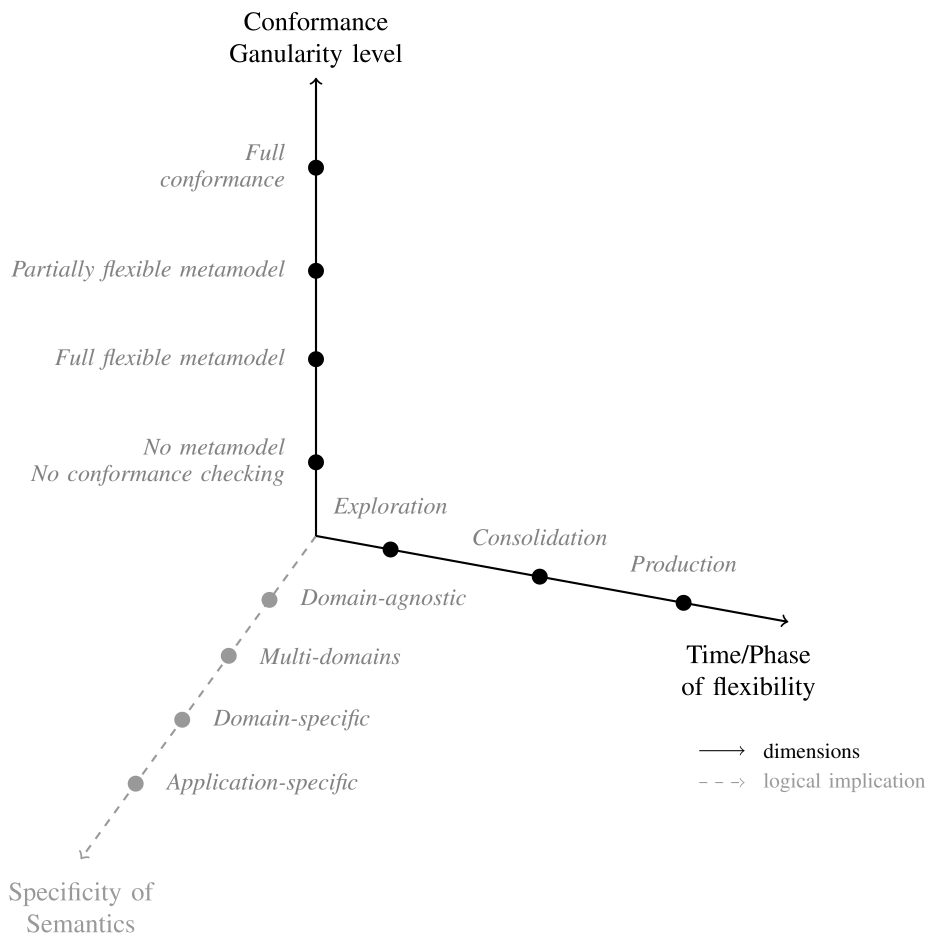

Fig. 1: Dimensions of Flexibility

IV. Reference Implementations

In this section we describe the two reference tools [6], [7] that drive this initiative. FlexiMeta and JSMF illustrate the time/phase of flexibility and the conformance granularity axis respectively.

A. FlexiMeta

FlexiMeta[3][4] is a modelling framework that promotes creativity over a strict model conformance. It is intended to address the problem of co-conception of models and metamodels in an arbitrary order, i.e., without assuming one is created before the other. To do so, it distinguishes several phases over time, corresponding to the different levels of definition of a metamodel with which a model can comply. For each phase, a right balance between flexibility and validation is found in order to bridge the gap between creativity and strict model conformance. The three phases are:

Exploratory Phase: The main focus during the *exploratory* phase is to encourage creativity. It allows designers to shape models and notations as they like, without being concerned by what the abstract syntax should be. This phase favors fast prototyping and in this sense, is close to the philosophy of Agile methods. No validation is possible during this phase as the abstract syntax has not been defined yet. At most, metamodels can be inferred from the exploratory phase and give some suggestions to the designers. Therefore, metamodel inference techniques can be complementary at the exploratory phase to assist in the migration to the *consolidation* phase.

Consolidation Phase: The *consolidation* phase is an intermediary phase during which metamodels are built. It is intended to remain until the metamodels reach a stable state. During this phase, partial validation of models created during the *exploratory* phase is possible, but is unobtrusive, as metamodels are not stable yet. Tolerant validation is intended to warn designers about possible incoherence and digressions between the models and the metamodel. This phase is the most important one and could be considered as the *meet-in-the-middle* which aims at filling the gap between creativity and model conformance. During this phase, both models and metamodels may evolve. This phase terminates when metamodels are stable and models conform to them. Model and metamodel co-evolution techniques can be used as a complement to help designers align both models and metamodels.

[3]FlexiMeta is available at: https://github.com/nicolas-hili/FlexiMeta [7]
[4]An online demo is available at: http://fleximeta.net/demo-simpson

Finalisation Phase: The *finalisation* phase is the production phase. It does not permit creativity any longer but instead, ensures a full validation of the produced models. through a strict model conformance. Every model created during this phase should comply with the constraint rules defined by the abstract syntax.

The different phases of FlexiMeta are balancing flexibility and validation (through model conformance) over time. While the nominal scenario is a top-down process (from exploration to finalisation), some situations, such as model and metamodel migration, may require to go backwards the process.

The architecture of FlexiMeta has been roughly inspired by the work promoting the use of dynamic programming languages over strict type checking. It is written in JavaScript and relies on JSON for model persistence. The choice of these two languages with respect to some design considerations on flexibility and persistence. JavaScript is well suited as it provides dynamic typing constructs through the concept of *prototype*, and JSON is schema-less which supports model persistence even if no user-defined metamodel has been yet defined.

FlexiMeta seeks the best trade-off between the two disciplines. Therefore, JavaScript and the prototype concept are convenient to use in early phases of the development where creativity is encouraged over type safety checking, while during the phase of maturation and alignment of models / metamodels, increasing the typing ability of the language (through code generation / generation of assertions) is vital to increase the quality of the modelling framework.

B. JSMF

The JavaScript Modelling Framework (JSMF)[5] is a flexible framework for implementing web-based MDE. JSMF is made to support all the modelling continuum as defined in Natural Modelling [29]. Notably, it is able to support the early stage of natural modelling using raw models (i.e., where there is no metamodel or no conformance checking) and it should permit to support structured/formal models as well (where conformance has to be enforced). Between these two situations, JSMF allows for defining which elements have to be checked regarding the conformance relations (e.g., entire model, attribute types, multiplicity, etc.). This is made possible by the separation of models and metamodels life-cycle with an external conformance checking.

1) Flexibility: Like any other MDE frameworks, in JSMF, a model element has to conform to its metamodel by default. Like in EMF, a model can be created from a metamodel definition enforcing the model elements to conform to its source metamodel (full conformance of Fig. 1). Flexibility can be set with a very coarse grain at the metamodel level. The entire metamodel can be flexible (i.e., there is no conformance checking) or only some specific classes can be set flexible whilst for the other classes, the model elements should remain conform to them. he flexibilty definition can change all along the modelling life-cycle. In JSMF we define the flexibility at finer grain:

Optional/mandatory attributes: By default in JSMF, attributes are optional but they can be set as mandatory thanks to an option in the attribute definition.
Attribute types: Types can be checked or not. The same applies to type-imposed constraints (e.g., a Number between 0 and 20).
References multiplicity: The multiplicity can be checked or not. Not checking the multiplicity means setting 0..* to it.
Reference targeted types: Targeted types can be loosely defined using *JSMFAny* alike in the EMF framework. However, we allow not to (completely) check the type. There is a slight difference between using *JSMFAny* and not checking type. For the latter, the declared type can be a hint for further conformance checking.
Reference opposite: The opposite relation can be checked or not (including the opposite multiplicity).

2) Specific functions: discovery and recovery: During the modelling life-cycle, one can go back and forth between full flexibility and full conformance (see modelling continuum [29]: between free/natural modelling and strict modelling. Going to a more flexible model is quite easy relaxing the conformance checking (i.e., using the JSMF functions presented before). Going to a less flexible model, to reach full conformance (see Fig. 1), it requires to reconcile metamodel and model definitions. This *reconciliation* can be made in two ways. First, the metamodel is the right specification consequently, the models have to be updated: *recovery*. Second, the model is the right specification (e.g., it is an archetypal example), then the metamodel has to be updated: *discovery*. Obviously, this can occur on a (meta)model element or at a global (meta)model level.

The **recovery** is implemented in JSMF using a *refresh* function which is applied to each model element regarding current metamodel elements. Recovery in JSMF could be a support for the consolidation phase (e.g., correcting the attributes present in model which are not conform to). To go further, recovery could encompass a (semantic) name proximity computation (such as in [30]) to keep the value during the recovery phase: `lastName -> familyName`.

The **discovery** is based on the notion of providing metamodel definition from (archetypal) examples. One can decide that a model element is an *archetypal* representation of a class, thus updating the metamodel accordingly. Such approach could be used at the end of the *consolidation* phase: stabilizing. Currently, it works only with one metamodel element but in a future version, it should be possible to reconcile different archetypal models used in the definition one metamodel element.

[5]JSMF is available at: https://js-mf.github.io/ [6]

TABLE I: Conformance Granularity (left), Time/Phase of Flexibility (right)

Approach	No conformance	Full flexible metamodel	Partially flexible metamodel	Full conformance	Exploration	Consolidation	Production
Salay's et Al.	○	◐	●	●	○	●	●
GIMM	○	○	○	●	○	◐	●[2]
Muddle	◐[1]	◐	●	◐[2]	●[1]	●	○
Modelverse	○	◐[3]	◐[3]	◐[3]	○	●	●
FlexiMeta	●	◐	○	●	●	●	●
JSMF	●	●	●	●	◐	●	◐

○ no support ◐ partially supported feature ● full support

1: supported by GraphML 2: done by EMF 3: depends on the chosen implementation

V. Comparative Evaluation Using the Dimensions

We made this preliminary tool selection from authors in the MDE community who addresses flexibility in tools. We have selected tools and approaches that mainly address the relaxation of the conformance relation: GIMM [31], Muddle [22], Agility in MDE [11], and Modelverse [32]. The comparison with our two reference implementations is presented in table I.

The GIMM approach is a pragmatic way to ensure flexibility: it removes all the conformance issues by transforming a specific metamodel into a generic entity-relation metamodel to which any model element could conform to. As such, it does neither redefine the degree of conformance which has still to be strictly ensured, but it could be applied for free or strict modelling.

The Muddle approach is considered here beyond its relation to concrete syntax. As for the abstract syntax, Muddle allows for working with some uncertainty – i.e., exploration/consolidation modelling phases – with to some extent full or partial flexibility. Beyond the full flexibilty supported here by graphical tool, conformance is still present, in a form that make it possible to use Muddle as input for Epsilon's model to model transformations. Full conformance is achieved by targeting EMF models whilst no conformance is given by the graphical representation in GraphML.

The Agile approach described in [11] focuses on the omission of information provided in the models. Flexibility manifests in the partially free concrete syntax the approach supports (see [11] for more examples). The conformance is then partially defined by the authors as a transformation for relaxing/tightening the conformance relation.

The approach of Modelverse is trying to get rid of hardcoded conformance relations allowing language designers to program their own conformance checking. It allows developer to provide a very strict conformance checking or rather flexible one. However, it enforces to have a minimal kind of a conformance. Without neglecting the other linguistic dimensions such concrete syntax, semantics, it focuses on the abstract syntax: it proposes, such as in [10], to separate the typing relation from the instantiation one.

Compared to the above approaches and tools, FlexiMeta promotes creativity over a strict model conformance. It goes from no conformance where no metamodel has been defined, to a full conformance during the finalisation phase. In between, a tolerant validation mechanism is used for assisting the user in the alignment of models/metamodels. While the nominal scenario is a top-down process from exploration to production, some situations, such as model and metamodel migration, may require to go backwards in the process. However, it does not support the gradual definition of the metamodel. Metamodels are currently defined outside FlexiMeta using EMF and integrated with FlexiMeta via code generation using Acceleo [7].

JSMF proposes a gradual definition of metamodels and conformance check. Compared to the other approach, it allows for a finer tuning and controlling of flexibilty. It goes from no conformance or even no pre-defined metamodel (i.e., using raw JavaScript objects) to a full conformance check (explicitly calling a checking function). As in many modelling situations, models are both partially well defined and partially require some flexibility (e.g., a model is still under design). JSMF allows fine tuning of conformance checking (i.e., attribute, reference type, multiplicity, etc.). In addition JSMF covers some specific processes to control the flexibility: *recovery* and *discovery* as described in Section IV.

VI. Conclusion

This paper is an attempt to give a unified view of flexibility in MDE. It focuses on the conformance flexibility (related to the abstract syntax) in terms of two orthogonal and complementary dimensions. These dimensions are intended to introduce a gradual definition of the conformance related to the gradient of *specificity* of semantics, and aim at answering the relevant questions about *how* (data-related) and *when* (process-related) the conformance between a model and a metamodel has to be enforced. Thus, this view goes beyond the simple vision which consists in seeing the conformance relation either entirely relaxed or entirely enforced. We have used the two dimensions for comparing existing approaches and tools related to the relaxation of the conformance relation.

It helped us to detect some shortages in existing work in terms of flexibility. In the future, we hope that this view will be used as a guideline to assist researchers, tool users, and practitioners in their search of the best flexible tool or framework that fits their needs.

While we believe that all aspects of DSMLs where flexibility is required have to be considered, yet it appeared to us that flexibility in the abstract syntax via the relaxation of the conformance relation is the most challenging problem and undervalued compared to other aspects of modelling languages. Especially, advocating flexibility via the use of free-form graphical modelling environments, hybrid textual/graphical notations, and collaborative tool support appears to be a hot topic that has largely been covered in many flexible tools and approaches. Thus, we have limited the scope of this paper to the abstract syntax only. Nevertheless, we plan to extend this work to establish a unified view encompassing all the different aspects modelling languages comprise.

Acknowledgment

The authors would like to thank Hans Vangheluwe, adjunct professor in School of Computer Science at McGill University. Han's comments during the past edition of FlexMDE'16 raised an interesting discussion that led to the identification of the dimensions presented in this paper. In addition, the authors would like to thank the reviewers of this paper for their insightful comments and suggestions.

References

[1] J. Sánchez-Cuadrado, J. De Lara, and E. Guerra, "Bottom-up metamodelling: An interactive approach," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2012, pp. 3–19.

[2] N. Mangano and A. van der Hoek, "A Tool for Distributed Software Design Collaboration," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*. ACM, 2012, pp. 45–46.

[3] D. Wüest, N. Seyff, and M. Glinz, "FLEXISKETCH TEAM: Collaborative Sketching and Notation Creation on the Fly," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2. IEEE, 2015, pp. 685–688.

[4] R. Jolak, B. Vesin, M. Isaksson, and M. R. Chaudron, "Towards a New Generation of Software Design Environments: Supporting the Use of Informal and Formal Notations with OctoUML," in *Second International Workshop on Human Factors in Modeling (HuFaMo 2016). CEUR-WS*, 2016, pp. 3–10.

[5] F. R. Golra, A. Beugnard, F. Dagnat, S. Guerin, and C. Guychard, "Using Free Modeling as an Agile Method for Developing Domain Specific Modeling Languages," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2016, pp. 24–34.

[6] J.-S. Sottet and N. Biri, "JSMF: a Javascript Flexible Modelling Framework," in *Proceedings of the 2nd Workshop on Flexible Model Driven Engineering*, vol. 1694. CEUR Workshops Modeling, 2016, pp. 42–51.

[7] N. Hili, "A Metamodeling Framework for Promoting Flexibility and Creativity Over Strict Model Conformance," in *Proceedings of the 2nd Workshop on Flexible Model Driven Engineering*, vol. 1694. CEUR Workshops Modeling, 2016, pp. 2–11.

[8] J. Bézivin, "On the unification power of models," *Software & Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[9] P. Gómez, M. Sánchez, H. Florez, and J. Villalobos, "Co-creation of models and metamodels for enterprise architecture projects," in *Proceedings of the 2012 Extreme Modeling Workshop*. ACM, 2012, pp. 21–26.

[10] J. De Lara, E. Guerra, and J. S. Cuadrado, "A-posteriori typing for model-driven engineering," in *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*. IEEE, 2015, pp. 156–165.

[11] R. Salay and M. Chechik, "Supporting agility in mde through modeling language relaxation." in *XM@ MoDELS*, 2013, pp. 20–27.

[12] T. Degueule, B. Combemale, A. Blouin, O. Barais, and J.-M. Jézéquel, "Safe model polymorphism for flexible modeling," *Computer Languages, Systems & Structures*, vol. 49, pp. 176–195, 2017.

[13] G. Wachsmuth, *ECOOP 2007 – Object-Oriented Programming: 21st European Conference, Berlin, Germany, July 30 - August 3, 2007. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. Metamodel Adaptation and Model Co-adaptation, pp. 600–624.

[14] A. Cicchetti, D. D. Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*, 2008, pp. 222–231.

[15] J. Schoenboeck, A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Schwinger, M. Wimmer, and M. Wischenbart, "CARE – A Constraint-Based Approach for Re-Establishing Conformance-Relationships," in *Asia-Pacific Conference on Conceptual Modelling (APCCM 2014)*, ser. CRPIT, G. Grossmann and M. Saeki, Eds., vol. 154. Auckland, New Zealand: ACS, 2014, pp. 19–28.

[16] J. Sánchez-Cuadrado, J. Lara, and E. Guerra, *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Bottom-Up Meta-Modelling: An Interactive Approach, pp. 3–19.

[17] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, 2008.

[18] "JSMF: JavaScript Modeling Framework Github Page," https://github.com/dslmeinte/jsmf, accessed: 2017-04-29.

[19] "Ecore (EMOF) implementation in JavaScript Github Page," https://github.com/emfjson/ecore.js, accessed: 2017-04-29.

[20] B. Costa, P. F. Pires, F. C. Delicato, and F. Oquendo, "Towards a view-based process for designing and documenting restful service architectures," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 50.

[21] H. Ed-Douibi, J. L. C. Izquierdo, A. Gómez, M. Tisi, and J. Cabot, "Emf-rest: Generation of restful apis from models," *arXiv preprint arXiv:1504.03498*, 2015.

[22] D. S. Kolovos, N. D. Matragkas, H. H. Rodríguez, and R. F. Paige, "Programmatic muddle management." in *XM@ MoDELS*, 2013, pp. 2–10.

[23] "Epsilon Family of Language," https://www.eclipse.org/epsilon/, accessed: 2017-04-29.

[24] "Moddle Github Page," https://github.com/bpmn-io/moddle, accessed: 2017-04-29.

[25] "Web-based tooling for BPMN, DMN and CMMN," http://bpmn.io/, accessed: 2017-04-29.

[26] M. Gerhart, J. Bayer, J. M. Höfner, and M. Boger, "Approach to define highly scalable metamodels based on json," *BigMDE 2015*, p. 11, 2015.

[27] D. Harel and B. Rumpe, "Meaningful modeling: what's the semantics of" semantics"?" *Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[28] N. Guarino, D. Oberle, and S. Staab, "What is an ontology?" in *Handbook on ontologies*. Springer, 2009, pp. 1–17.

[29] Z. Zarwin, M. Bjekovic, J.-M. Favre, J.-S. Sottet, and H. A. Proper, "Natural modelling." *Journal of Object Technology*, vol. 13, no. 3, pp. 4–1, 2014.

[30] K. Garcés, F. Jouault, P. Cointe, and J. Bézivin, "Managing model adaptation by precise detection of metamodel changes," in *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 2009, pp. 34–49.

[31] P. Gomez, M. E. Sánchez, H. Florez, and J. Villalobos, "An approach to the co-creation of models and metamodels in enterprise architecture projects." *Journal of Object Technology*, vol. 13, no. 3, pp. 2–1, 2014.

[32] Y. Van Tendeloo and H. Vangheluwe, "Explicit type/instance relations," McGill University, Tech. Rep., 2016.