Quality of a Metamodel

(Correctness, Completeness, …)

Compiled by: Zahra Rajaei and Atefeh Nirumand

MDSE Research Group, University of Isfahan

The correctness of a metamodel becomes extremely important. But this raises one problem: how can one make sure that a metamodel is correct? One solution to this problem, examined in this paper, is to automatically generate metamodel instances, and use these generated instances to test or verify the properties of a metamodel [1].

A practical example might be a metamodel for the Java programming language, where the classes are the elements of the abstract syntax, the constraints define the static semantics, and the instances of the metamodel are valid and compilable Java programs. Generating instances for such a Java programming language metamodel is equivalent to producing syntactically and semantically correct Java programs [1].

Correctness is the extent to which the metamodel descriptions have been checked against their specifications [2].

Completeness is the extent to which all features of the modeling language (including multiview consistency constraints) are supported in the metamodeling approach [2].

A metamodel is considered correct, if it only allows valid instances. Expressiveness is the degree to which it is able to express the instances it is supposed to [3].

Authors in [4] verify their metamodel in this way: To ensure the quality of the learning ecosystem metamodel it is necessary to validate it through a Model-to-Model transformation. Specifically, it is required to verify that the learning ecosystem metamodel allows defining real learning ecosystems based on the architectural pattern. Namely, it allows defining models that represent learning ecosystems based on the architectural pattern that can be deployed in real contexts to solve learning and knowledge management problems.

Quality metrics are measured as per following [5]:

1) Simplicity: Is our metamodel simple to model cloud systems? The response to this question is yes as our metamodel contains only twelve concepts – eight from the OCCI Core Model and four new ones – allowing to model seven cloud computing domains, when other concurrent cloud standards like CIMI or TOSCA define a huge set of concepts, and only addressed IaaS and cloud applications, respectively.

2) Consistency: Is our precise semantics of OCCI consistent for modeling any OCCI systems? The response to this question is yes as there are no contradictions between our twenty five OCL invariants, else EMF-VF will not evaluate our whole dataset as correct (last column in Table I is equals to the number of instances multiplied by the number of invariants).

3) Correctness: Is our metamodel of OCCI correct for modeling any OCCI system? The response to this question is yes as our metamodel allows to correctly model all OCCI extensions in all observable aspects.

4) Completeness: Is our precise semantics of OCCI complete for modeling any OCCI systems? The response to this question is yes as our metamodel covers all situations encountered in the seven works

proposing an OCCI extension. Moreover, our metamodel fully covers all the OCCI core concepts and addresses the five drawbacks presented into Section II.

5) Usefulness: Is our metamodel useful for modeling all OCCI systems? The response to this question is yes as our new introduced classes – Extension, EDataType, Configuration – are useful for modeling the seven OCCI extensions and fully address the five drawbacks (cf Section II). Moreover, our extensible data type system is useful in all covered extensions, i.e., new EDataType instances are created in each extension.

References

[1] Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., & Tata, S. (2015, June). A precise metamodel for open cloud computing interface. In *2015 IEEE 8th International Conference on Cloud Computing* (pp. 852-859). IEEE.

[2] Paige, R. F., Brooke, P. J., & Ostroff, J. S. (2007). Metamodel-based model conformance and multiview consistency checking. *ACM Transactions on Software Engineering and Methodology (TOSEM), 16*(3), 11-es.

[3] Gomez, J. J. C., Baudry, B., & Sahraoui, H. (2012, April). Searching the boundaries of a modeling space to test metamodels. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (pp. 131-140). IEEE.

[4] García-Holgado, A., & García-Peñalvo, F. J. (2019). Validation of the learning ecosystem metamodel using transformation rules. Future Generation Computer Systems, 91, 300-310.

[5] Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., & Tata, S. (2015, June). A precise metamodel for open cloud computing interface. In *2015 IEEE 8th International Conference on Cloud Computing* (pp. 852-859). IEEE.

Metamodel Quality [1]

Metamodel quality can be subdivided into several quality characteristics. Metamodel quality is, however, still a research field and characteristics are still discussed and adjusted. Therefore, the characteristics are adapted, extended or linked to other characteristics, where it is meaningful.

Bertoa and Vallecillo [BV10] transfer the quality model for software from the ISO/IEC standard 9126 [ISO01] to modeling. The characteristics are functionality, reliability, usability, efficiency, maintainability, and portability.

- The correctness of a metamodel refers to whether a metamodel expresses the abstractions it is supposed to. This means whether it is able to express all the instances it is supposed to without any unnecessary information. Correctness can be further split into two dimensions: completeness and preciseness. To be able to describe these terms, the term set of intended models (IM) has to be introduced. IM contains all models that the metamodel should be able to express. It is, therefore, potentially infinite. A correct metamodel is complete as well as precise. An incorrect metamodel is incomplete, imprecise, or both.

- The preciseness [GBS12] of a metamodel specifies to which degree a metamodel is able to only allow models from IM. E.g., an imprecise metamodel allows to model irrelevant concept. A precise metamodel does not allow any models that are not in IM. Preciseness is also related to the relevance and correctness sub-characteristics of Bertoa [BV10].

- The completeness [BV10] of a metamodel specifies to which degree a metamodel is able to express the models of IM. A complete metamodel can express all models of IM. For an incomplete metamodel, there are models in IM that it cannot express. Metamodel completeness can be seen as an adaption of the functional completeness sub-characteristic of software products [ISO11].

- The reusability of a metamodel refers to how well parts of the metamodel can be reused for other languages. For example, it is detrimental for the reusability of a metamodel if the metamodel is too specific. If parts of the metamodel are reused by other languages, they contain irrelevant abstraction. This means they are imprecise in their new context. The problem of too high specificity can be alleviated by improved modularity. By separating abstract concepts from their specifics, the concepts are more suited for reuse.

- The extensibility of a metamodel refers to how well it lends itself to be the basis of extensions. A metamodel is suited for extensions if the extension can be applied in the metamodel in a way that includes no irrelevant abstractions when the extension is used. In this regard, metamodel reusability is analogous to metamodel reusability. A monolithic metamodel with many specifics has bad extensibility. A modular metamodel that separates the specifics from its abstractions offers a proper basis for extensions. The modularity of a metamodel is “the extent that its parts are systematically structured and separated such that they can be understood in isolation” [BV10, p. 9]. Strong coupling of many parts of a metamodel degrades its modularity. Parts of a metamodel should only be coupled if it is necessary. This is the case when dependencies cannot be avoided or modeled differently.

- The evolvability of a metamodel refers to how well the metamodel can be evolved. Evolvability can be broken down into analyzability and modifiability.

- The analyzability [BV10] of a metamodel refers to how easy it is to inspect it for deficits or to identify parts of the metamodel that have to be modified. Analyzability is influenced by modularity.

- The understandability is “the degree in which a metamodel is self-describing”. The understandability of a metamodel is influenced by its complexity [BV10]. Understandability and analyzability are related, as an understandable metamodel is also easier to analyze. By the same argumentation as for analyzability, the understandability of a metamodel is also influenced by its modularity.
- The modifiability of a metamodel refers to how well its structure supports modifications. The modifiability of a metamodel is influenced by its modularity [BV10].

References:

[1] Strittmatter, M. (2020). A Reference Structure for Modular Metamodels of Quality-Describing Domain-Specific Modeling Languages (Vol. 31). KIT Scientific Publishing.