Consistency Rules for UML-based Domain-specific Language Models: A Literature Review

Bernhard Hoisl and Stefan Sobernig
Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna)
{bernhard.hoisl, stefan.sobernig}@wu.ac.at

***Abstract*—The Unified Modeling Language (UML) has become a popular implementation vehicle for domain-specific modeling languages (DSMLs). A UML-based DSML is typically defined by multiple specification artifacts, i.e. inter-related models, describing different views on the DSML. These separate, yet inter-related models are potential sources of specification inconsistencies which bear a high risk of affecting all subsequent DSML development phases (e.g., platform integration). In a large-scale literature review of more than 8,000 publications, we collected evidence on consistency-rule usage for 84 UML-based DSML designs. In this paper, we report on the identified patterns of consistency-rule usage (e.g., rule formalization, rule scopes, and supported development activities) and specification defects which challenge the use of consistency rules in DSML specifications.**

I. Introduction

Domain-specific modeling languages (DSMLs) are specialized modeling languages tailored primarily for graphical modeling tasks in a particular application domain to support the model-driven development (MDD) of software systems for this domain. As a special kind of domain-specific languages (DSLs), DSMLs provide end users with at least one graphical or diagrammatic concrete syntax—in contrast to, for example, textual or form-/table-based DSLs (see, e.g., [1], [2]).

In recent years, developing DSMLs based on the Meta Object Facility (MOF [3]) and integrated with the Unified Modeling Language (UML [4]) has become a widely adopted option (see, e.g., [5], [6]). A UML-based DSML tailors its host language (i.e. the UML) to the needs of a particular domain (e.g., by introducing domain-specific model elements or by restricting the semantics of existing UML elements). These domain-specific aspects are specified on the level of a DSML's language model, which captures all relevant domain abstractions and specifies the relations between these abstractions (see, e.g., [7]). To tailor a DSML's language model, language-model constraints are employed, for example, specified by informal textual annotations (e.g., UML comments [4]) or in a formal language (e.g., OCL [8]).

In the DSML context, consistency rules are devised to ensure that the different artifacts of a UML-based DSML do not contradict each other due to conflicting syntax and semantics specifications (see, e.g., [9]–[11]). A DSML specification covers also the phases of defining the DSML's concrete syntax, behavior, and platform integration [7]. The result of such a DSML specification are multiple interdependent specification artifacts. For example, DSML-specific constraints—as part of a DSML's language model—need to be enforced for all instance models to ensure compliance with their respective metamodel (i.e. the DSML's language model). Furthermore, the UML provides 14 different model and diagram types to specify different (structural and behavioral) concerns of a software system [4]. DSMLs can build on multiple model and diagram types at the same time, therefore, putting emphasis on inter-model consistency.

In a recent systematic literature review (SLR), we extracted design decisions from UML-based DSMLs and collected the corresponding DSML specification artifacts [12]. The review is a data source for two aspects of consistency rules for UML-based DSML specifications. First, we extracted data on consistency-rule usage in DSML specifications. From 84 DSML designs, we retrieved details on employed consistency-rule formats, DSML language-model formalizations, consistency-rule scopes, supported software-engineering activities, the underlying UML model and diagram types, and supporting software tools. This complements the work on consistency rules by [10], [11] from the perspective of DSMLs realized as UML extensions. Second, the review spotted critical specification defects for UML-based DSMLs. These defects in the UML formalization of a DSML's language model (e.g., incomplete and insufficient specification of UML profiles, incorrect use of constraint-language expressions) result in issues for defining consistency rules.

In summary, the key contributions of this paper are the extraction, analysis, and discussion of consistency-rule usage in UML-based DSML designs. This complements the work by Torre et al. ([10], [11]) which focusses on UML in general. In addition, the paper highlights challenges specific to UML-based DSMLs when it comes to providing an infrastructure for defining consistency rules, including recommendations to avoid commonly observed pitfalls in DSML development. On top, we provide descriptive findings on (extended) UML usage (e.g., UML diagram types) adding to the current body of empirical research on UML (see, e.g., [13], [14]).

The remainder of the paper is structured as follows. Section II summarizes important background information with respect to DSML development, SLR procedure, and specification consistency in this context. Results of the data-extraction process are presented in Section III, limitations of the SLR in Section IV. Section V puts the extracted data on consistency-rule usage in DSMLs into perspective and discusses the role

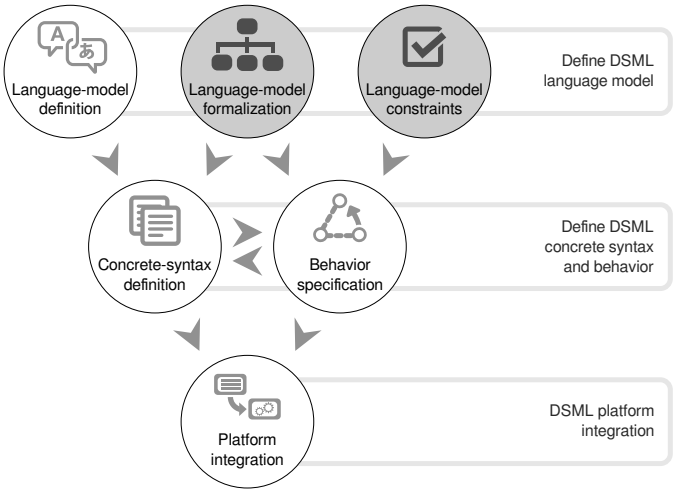

Fig. 1. Language-model driven DSML development process.

of specification defects in this context. Our study is compared to related work in Section VI and Section VII concludes the paper.

II. Background

The following three sections recap details of developing DSMLs (Section II-A), conducting the SLR (Section II-B), and evaluated UML consistency aspects (Section II-C) important to interpret the results presented in Section III.

A. DSML Development

DSML development is an exploratory, iterative process. A process view (such as [7]) treats DSML development as a complex flow of characteristic development activities (e.g., language model definition, constraint specification etc.). We focus on a language-model driven DSML development activity which discriminates between the following development phases [7]: define DSML language model, define DSML concrete syntax and behavior, and DSML platform integration (see Fig. 1).

In our work of evaluating consistency rules for UML-based DSMLs, we target the phase of the domain-specific language model definition (see Section I and Fig. 1). In this first phase of a language-model driven DSML development activity, a core language model and the corresponding language model constraints for the selected target domain are defined. By following a domain analysis method, such as domain-driven design (see, e.g., [15]), domain abstractions are identified and form the language model of a DSML. This initial *language-model definition* may not be UML-compliant (e.g., textual descriptions, informal models) and need to be turned into a *formal language model*. By formal model, we refer to a realization of the language model using a well-defined metamodeling language such as the MOF/UML metamodeling infrastructure. A metamodeling language is itself based on a well-defined and well-documented language model (i.e. CMOF for the UML metamodel [3]) and provides at least one well-defined and well-documented concrete syntax to define an own language model (e.g., the CMOF diagram syntax to specify a UML metamodel extension).

Because the language model often cannot (or only insufficiently) capture all restrictions and/or semantic properties of the DSML elements, *language-model constraints* are added. These language-model constraints prevent the language model to be formalized incomplete, ambiguous, and/or inconsistent with other DSML artifacts. As such, language-model constraints form the basis for the definition of consistency rules and are specified, for example, by employing special-purpose constraint languages (such as the OCL [8]) or unstructured textual annotations.

After the definition of a DSML's language model, the *concrete syntax* of a DSML is defined (i.e. suitable notation symbols as well as composition and production rules) which serves as a DSML's user interface (see Fig. 1). In parallel, the *behavior* of DSML language elements is specified to produce the behavior intended by the DSML designer. In the last phase, all artifacts defined for a DSML are *integrated* into a selected software platform to produce platform-specific (executable) models (e.g., by employing model transformations to generate source code [16]).

B. Systematic Literature Review

We performed a systematic literature review (SLR) to distill generic design decisions from UML-based DSML design documents for the different development phases discussed in the former section. Here, we briefly summarize the process and the results of the SLR; details are published in [12] and in [17]. The main goal of the SLR was to identify a maximum number of high-quality scientific publications which document design decisions on UML-based DSMLs as primary sources.

The SLR was performed in three steps. First, to provide a basis for evaluation of the search procedure, we established a corpus of reference publications as *quasi-gold standard* (QGS [18]). In essence, a QGS is a set of hand-picked publications considered relevant for a specific SLR. In the end, the constructed QGS corpus consisted of 37 publications (24 journal and 13 proceedings articles). Based on these QGS publications, the relevant search engines for the automated search were identified (SpringerLink, IEEE Xplore, Scopus, and ACM Digital Library) and a search string for the automated search was constructed (the query expression represented 544 unique pairs of search terms).

Second, we performed the actual *engine-based publication search* using the search string developed in the previous step on the four selected search engines. The search execution yielded 5,778 search hits split into four result sets, one for each of the search engines. After enforcing the QGS-based capping, having evaluated the papers based on our selection criteria and having completed the quality assessment, 73 papers representing 1.3% of the original search hits remained. For this final publication set, we extracted the publication-specific data (15 metadata items for each paper, including bibliographical entries, selection decision, and decision-mining entries).

Third, based on the bibliographical records extracted from the 73 publications selected up to this point, we then performed a *backward-snowballing search*. Backward snowballing is the practice of manually identifying additional publications for selection from the reference lists (citations) of a given set of publications [19]. Via the backward snowballing search, we reviewed a total of 2,337 references. After evaluation and quality assessment of the papers, eight were included into the paper corpus (0.3%). From these additional publications, we extracted publication-specific data in the same way as was done for papers retrieved by the main search.

We considered a total of 81 articles as relevant: 73 from main search plus eight from snowballing. To complete the paper corpus, we re-considered the QGS publications not retrieved by the main and the snowballing searches for inclusion based on the selection criteria. This way, we classified two QGS journal articles and one QGS conference article as relevant. We so arrived at a paper corpus of *84* publications (the complete list of publications is provided in [20]). The corpus was composed of 54 conference articles (64%) and 30 journal articles (36%).

C. Consistency in UML-based DSMLs

In this paper, we investigate six aspects of model-level consistency in UML-based DSML designs in line with [10], [11].

Language-model formalization: After the identification of language-model concepts, the corresponding definitions serve as input for the phase of formalizing the domain constructs into a MOF/UML-compliant language model (see Section II-A). As we focus on consistency rules at the level of a DSML's language model, we establish how the domain abstractions are formalized using the MOF and/or the UML. Available options are *UML M1 structural model* (e.g., UML class models), *UML profile definition* (i.e., extending UML metaclasses with stereotypes), and *UML metamodel extension* (i.e., adding new metaclasses and/or new associations between metaclasses to the UML metamodel) [20].[1]

Consistency-rule formats: A DSML's language model formalization is limited by the expressiveness of the MOF/UML (e.g., part-of relations). Semantic variation points in the MOF/UML may render a DSML's language-model specification incomplete and/or ambiguous. This risks introducing inconsistencies across different DSML modeling artifacts ([4], [20]). Therefore, we assess whether consistency rules are provided for a DSML to cover such variation points [10]. If so, we document the choice of rule representation (e.g., OCL expressions [8]).

Consistency-rule scopes: We record whether consistency rules target a *single model* only (e.g., to resolve ambiguities in the definition of a model) or whether the rules relate multiple models. For inter-model scenarios, *horizontal* consistency refers to consistency between different, but complementing models at the same level of abstraction (e.g., between different platform-independent models). *Vertical* consistency refers to consistency between models at different levels of abstraction (e.g., between platform-independent and platform-specific models). *Evolution* consistency refers to consistency between different versions of the same model (e.g., between an input and an output model of a model transformation [10]).

Software-engineering activities: Model-level consistency rules are employed in support of different software-engineering activities. Observed activities are *refinement* ("semantics-preserving changes applied to a model, to reduce non-determinism" [11]), *verification* ("determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase" [11]), *checking constraints* ("of models according to consistency rules and producing a list of violations" [11]), *transformation* ("describe the application of mapping rules on one model to create a new model" [11]), and *heuristics* (rules that are written as plain text "for solving a UML consistency problem without the exhaustive application of an algorithm" [11]).[2]

Model and diagram types: We document which of the 14 structural and behavioral UML model and diagram types [4] are actually tailored by the DSMLs. These are, therefore, the model and diagram types for which consistency rules are defined for various rule scopes ([10], [11]).

Tool support: We evaluate whether consistency rules (in a given representation) can be automatically processed and validated. In addition, we provide an inventory of supporting software tools for rule processing and validation (e.g., constraints evaluators [10]).

III. Extracted Data on DSML Consistency

This section presents the data on the six consistency aspects in the corpus of 84 DSML designs collected via the SLR ([12], [17]). For data extraction, we studied the corresponding publications as the primary design documents for cues on each of the six consistency criteria. 52 out of 84 DSML designs (62%) explicitly specified consistency rules at the level of the DSML's language model. For 32 DSML designs (38%), we did not find any documentation hints of consistency-rule definitions.

Table I shows the frequency of UML-based language-model formalization options identified for the 52 DSML designs. The majority of DSMLs (84%) employ UML profiles to formalize their language model. Only one DSML defines its UML-based language model via an M1 structural model. The language model of four DSMLs is specified by using a combination of a UML profile and a UML metamodel extension.

To quantify the specification size of these 52 DSML designs, we evaluated the size of their core language-models. Depending on the different, underlying UML language-model formalization options, the specification size was established differently. For 47 DSMLs defining their language models

[1] We only discuss formalization options actually observed for DSMLs in our SLR study (see Section III). A complete list of language-model formalization options is documented in [20].

[2] Again, our analysis is limited to activities observed in our SLR (see Section III). The complete list of relevant software-engineering activities is available from [11].

TABLE I
UML-BASED LANGUAGE-MODEL FORMALIZATION OPTIONS.

Language-model formalization	Frequency
UML profile definition	47 (84%)
UML metamodel extension	8 (14%)
UML M1 structural model	1 (2%)
Total	56 (100%)

TABLE II
EMPLOYED FORMATS TO SPECIFY CONSISTENCY RULES.

Consistency-rule format	Frequency
Unstructured text	36 (50%)
OCL	33 (46%)
Mathematical expressions	2 (3%)
ATL	1 (1%)
Total	72 (100%)

using UML profiles, we counted the stereotype definitions and the corresponding, distinct base UML metaclasses. In this group, we find a median of 14 ± 9.6[3] stereotype definitions per DSML. A typical profile extends a median of 5 ± 3 distinct base metaclasses per DSML. For the eight DSMLs using a UML metamodel extension, we collected the number of newly introduced UML metaclasses. A typical DSML adds a median of 19.5 ± 11.9 UML metaclasses. For one DSML defining its language model using a UML structural model at level M1, we were unable to count the number of UML classes due to its incomplete design documentation.

The 52 DSML designs containing consistency rules adopted four different rule formats (see Table II). 33 DSMLs (63%) use one (either unstructured text, OCL, or mathematical expressions), 18 DSMLs (35%) use two (both, unstructured text and OCL), and one DSML three different formats (unstructured text, OCL, and ATL [21]) to specify consistency rules. There are nearly equal shares of DSMLs adopting unstructured (informal) text (50%) and (formal) OCL expressions (46%). Mathematical and transformation-language expressions (e.g., in ATL) are rarely used with three DSMLs only.

The majority of 52 DSMLs (79%) apply consistency rules for the scope of a single model (see Table III). Inter-model consistency rules for a horizontal scope (i.e., at the same abstraction level) were found for seven DSMLs (12%). A minority of three DSMLs define consistency rules spanning different abstraction levels (vertical consistency). Consistency rules between different versions of a language model (evolution consistency) were reported for two DSMLs only. In five DSMLs, consistency rules had mixed scopes: single model/vertical consistency (2 DSMLs), single model/evolution consistency (2), and horizontal/vertical consistency (1).

As for software-engineering activities supported by the consistency rules, almost equal shares of DSMLs relate to three activities of heuristics (32%), verification (32%), and constraint checking (30%; see Table IV). In turn, rules rarely target model transformation and refinement activities with only four and three cases, respectively. In 17 DSMLs (33%), rules are employed for one software-engineering activity only (15x heuristics, 2x verification). Mixed usage is reported for 16 DSMLs (31%) with two supported activities (14x verification/constraint checking, 1x heuristics/transformation, 1x heuristics/refinement), and for 15 DSMLs (29%) with three activities (heuristics/verification/constraint checking). More than three supported activities are limited to a minority share of four DSMLs.

[3]We report the variance in terms of the *median absolute deviation from the median* using the $\pm$ notation along with the median value.

TABLE III
IDENTIFIED SCOPES OF CONSISTENCY RULES.

Consistency-rule scope	Frequency
Single model consistency	45 (79%)
Horizontal consistency	7 (12%)
Vertical consistency	3 (5%)
Evolution consistency	2 (4%)
Total	57 (100%)

TABLE IV
RELATING CONSISTENCY RULES TO SOFTWARE-ENGINEERING ACTIVITIES.

Software-engineering activity	Frequency
Heuristics	36 (32%)
Verification	35 (32%)
Constraint checking	33 (30%)
Transformation	4 (4%)
Refinement	3 (3%)
Total	111 (100%)

One DSML is unspecific about the UML model and diagram types it is tailoring and is therefore omitted in Table V. For the remaining 51 DSMLs, class diagrams are ranked first with a 35% share, followed by activity (12%), and component as well as package diagrams (each 11%). No DSML tailored communication, profile and timing diagrams. Given that each of the 51 DSMLs can build on multiple model and diagram types, a total of 95 tailored UML diagram types were identified. 65 (68%) are structural and 30 (32%) are behavioral diagram types. Typically, a DSML tailors more than one UML diagram type. There exists 28 unique combinations of different diagram types tailored by the DSMLs. Most DSMLs build on either class diagrams only (8 DSMLs, 16%) or class diagrams in combination with package diagrams (8). Five DSMLs adopt activity diagrams only and three DSMLs combine class and object diagrams. All other combinations of diagram types are employed by at most two DSMLs each; and are omitted for brevity.

Software tools for processing and enforcing consistency rules are shown in Table VI. In total, we identified relevant tool support for 22 out of 52 DSML designs (42%; 16 unique tools). The majority of DSMLs (58%) did not document any tool usage. Five DSMLs (23%) use the OCL project of the Eclipse Model Development Tools (MDT) and three DSMLs

TABLE V
TAILORED UML DIAGRAM TYPES. AN ASTERISK (*) DENOTES A STRUCTURAL, ALL OTHERS ARE BEHAVIORAL DIAGRAM TYPES [4].

UML diagram type	Frequency
Class*	33 (35%)
Activity	11 (12%)
Component*	10 (11%)
Package*	10 (11%)
State machine	7 (7%)
Composite structure*	6 (6%)
Use case	6 (6%)
Sequence	5 (5%)
Object*	4 (4%)
Deployment*	2 (2%)
Interaction overview	1 (1%)
Communication	0 (0%)
Profile*	0 (0%)
Timing	0 (0%)
Total	95 (100%)

TABLE VI
EMPLOYED TOOLS TO VALIDATE CONSISTENCY RULES.

Tool	Frequency
OCL project of the Eclipse Model Development Tools (MDT)	5 (23%)
IBM Rational Software Architect	3 (14%)
CompSize	1 (5%)
Eclipse Atlas Transformation Language (ATL)	1 (5%)
Eclipse EMF Compare	1 (5%)
EIS plug-in	1 (5%)
Gentleware Poseidon for UML	1 (5%)
ITEM ToolKit	1 (5%)
Kent Modeling Framework (KMF)	1 (5%)
LTSA	1 (5%)
No Magic MagicDraw	1 (5%)
Oclarity	1 (5%)
Octopus	1 (5%)
Telelogic Tau (G2)	1 (5%)
TOPCASED	1 (5%)
WebRatio	1 (5%)
Total	22 (100%)

(14%) IBM's Rational Software Architect (RSA) to validate consistency rules.[4] The remaining 14 software tools are each deployed in a single DSML project only.

IV. SLR LIMITATIONS

SLRs have the general problem of finding a representative set of relevant primary studies. We closely followed established guidelines on designing and conducting SLRs available from research on evidence-based software engineering to avoid any pitfalls ([18], [19], [23]). However, we may risk having missed further relevant primary studies on UML-based DSMLs. For example, by extracting data from our paper corpus, we did not find empirical evidence for every consistency-rule format proposed by related work, such as, code annotations or constraining model-to-text transformations [20]. Nevertheless, we addressed this threat right from the beginning, by building our review procedure around the principle of continuous search validation and search refinement driven by a QGS as a recommended practice [18].

We intentionally limited our SLR exclusively to DSMLs embedded into UML 2.x [4], thereby excluding DSMLs based on former UML versions and other metamodeling infrastructures (e.g., Kermeta or MetaGME). We only considered consistency rules specified on the level of UML-based DSML language models; i.e. we restricted data extraction to the DSML development phases of formalizing a UML-compliant language-model and defining accompanying language-model constraints (see Section II-A). Therefore, on the one hand, we excluded rules applied on non-UML artifacts (e.g., non-UML platform-specific models generated during the platform integration phase). On the other hand, we also excluded consistency rules relating to other UML models besides a DSML's language model (e.g., UML M1 behavioral models as part of a DSML's behavior specification). Furthermore, we excluded exemplary as well as application-specific consistency rules found in DSML reports (e.g., as part of a case study exemplifying the application of a DSML).

We exclusively report on tools used to enforce consistency rules on DSML language models. We do not consider tool support for other phases in DSML development, such as, language-model editors, generators for concrete-syntax editors, model-execution engines, model-transformation engines, or orchestration engines.

V. DISCUSSION

First, we elaborate on the relevance of the extracted data presented in Section III. Against this background, we reiterate over frequently reoccurring specification defects in UML-based DSML language models, as revealed by [12], [17].

A. Interpretation of Review Data

Language-model formalization: The preponderance of UML profiles might partly be explained as they are the native UML extension mechanism [4], their application is known to modelers, and plenty of supporting tools exists (e.g., language-model and concrete-syntax editors). UML profiles provide for packaging and for scoping intra-model consistency rules (i.e., OCL expressions) as part of a DSML's language-model formalization. To this date, portability of such OCL consistency rules between different evalution engines remains limited due to the OCL/UML language specifications leaving critical details to language and tool implementers (e.g., navigation semantics between extension and extended model elements; see [24] for an overview).

Another critical issue pertaining to (esp. formally specified) consistency rules in multi-level and shallow-instantiation-based metamodeling environments such as MOF/UML is their confinement to direct instantiations (e.g., M1) of model elements (e.g., M2). Consider as an example a DSML language model defined at level M2 which must enforce consistency

[4]We separately report the Eclipse MDT/OCL project and IBM's RSA because RSA bundles a couple of the MDT/OCL plugins deviating these in unknown ways from the official Eclipse OCL plugins [22].

rules at level M0, i.e. the occurrence (instance) level of DSML models. This requirement is documented for DSMLs in the business-process modeling domain in which consistency conditions are stipulated for the scope of business-process instances (see, e.g., [25], [26]). To date, there are certain conventions (e.g., escaping to informal rule definitions [20]), implementation idioms (e.g., prototypical concept pattern [27]), and alternatives to metamodeling based on shallow instantiation (e.g., potency and deep instantiation [28]) to work around or to address this limitation. However, no comprehensive solution has yet become available in the family of MOF/UML/OCL languages as a DSML development infrastructure.

We found that a language model can been realized by multiple formalizations (e.g., a combination of a UML profile and a UML metamodel extension as observed four times). This bears a double risk. On the one hand, consistency rules must be defined for the scope of two different artifacts, metamodel and profile packages, causing ambiguity in rule specification and possible rule conflicts. On the other hand, such a mixed DSML language model can potentially be used in different configurations (e.g., different profile and metamodel compositions). As an extreme, when integrating two or more DSMLs which are realized as (otherwise independent) UML extensions, reconciling the original consistency rules becomes a challenge [20].

Consistency-rule formats: We observed a comparatively high frequency of unstructured text and OCL expressions employed in combination (in 37% of the DSMLs). This is partly explained by the reporting needs of a scientific publication, requiring a certain level of elaboration on otherwise formal constraint expressions. Another driver might be that consistency rules expressed in some constraint-expression language must be complemented with textual explanations when applied beyond the context of a single model. To express consistency conditions between two or more models (horizontally and vertically), missing any direct and/or navigable inter-model links, alternative approaches (e.g., constructs in model-transformation languages, non-standard constructs in constraint-expression languages such as in the Epsilon Validation Language, EVL [29]) must be evaluated for adoption on a case-by-case basis. In doubt, complementary textual explanations (as we found in this study) are a viable option.

Similarly, in the context of evolution consistency, one DSML [30] specified consistency rules in a combination of OCL expressions evaluated in ATL-based model transformations (ATL can be used to define constraints on models [21]). The authors of [30] present an approach for model execution by a series of model transformation steps (exemplified by an evolving state machine diagram). In this case, OCL expressions are still employed to ensure the consistency of a single model. However, with the combination of ATL transformations and, thus, different model versions on which the OCL expressions are evaluated against step-by-step, the consistent evolution of a model is ensured.

Consistency-rule scopes: Intra-model consistency rules for DSML language models are the most frequent rule scope identified by our SLR. As for inter-model consistency, consistency was ensured by using (at least) unstructured textual artifacts. For example, in [31] textual rules are defined for horizontal consistency between composite structure and activity models in support of a heuristic activity. Vertical consistency was always observed in combination with a refinement activity. In [32], an abstract user-interface (UI) class model is refined into a UI deployment model. Consistency rules integrated with model transformations for evolution support have already been given as an example above [30].

Software-engineering activities: According to our definitions, we classified consistency rules formulated as unstructured texts as related to the software-engineering activity "heuristics" and OCL expressions as related to the constraint-checking activity (see Section II-C). This data-extraction convention explains the closely aligned figures reported for these consistency-rule formats and the corresponding software-engineering activities. Furthermore, we classified constraint checking as a verification technique (i.e. as part of the verification activity).

We did not find any evidence for the management, validation, and maintenance software-engineering activities as defined in [11]. The reason for their absence may be that these are not primary activities in the process of designing a research-driven DSML (see Section II-A). Managing consistency, evaluating the satisfaction of user requirements, or maintaining interdependencies between platform-independent models and platform-specific implementations may not be of high priority when developing scientific UML-based DSMLs (and, thus, are postponed).

Model and diagram types: In this study, we put emphasis on consistency rules formulated at the level of a DSML's language model (M2 level). These rules are enforced on instance models of a DSML (M1 level). A DSML's language model was frequently found formulated as a UML profile (in 84% of the DSMLs). Overall, multiple combinations of tailored diagram types could be observed (28 unique combinations). This combinatorial variety indicates the domain-specific application requirements matched by the diagram types adopted by each DSML found by our SLR.

Tool support: All of the 16 software tools found support the automatic evaluation of consistency rules. Because of diversified tool usage, we could not identify repeated occurrences except for the Eclipse MDT/OCL project (5 times, 23%) and IBM's RSA (3 times, 14%). Nevertheless, the small number of tooling support found (no consistency-enforcing tool was mentioned in 58% of the DSMLs) does not necessarily indicate that in these cases consistency rules are evaluated manually. In particular, we found OCL expressions being documented for 33 out of 52 DSMLs (63%), which can in principle be automatically evaluated.

B. DSML Specification Defects

Our SLR exposed six defect kinds in DSML specifications for 31 reviewed design documents ([12], [17]). As an extreme case of a DSML specification defect, metamodel and/or profile

definitions were found entirely missing (e.g., in [33]). Rather, we found that stereotypes are often applied in UML instance models without a proper profile definition ([12], [17]). In such cases, any kind of consistency rule lacks the foundation of a valid interpretation. However, there are also less obvious sources of challenges.

Such defects often reveal misconceptions about UML extension techniques. In addition, they pose particular challenges to formulating consistency rules and prevent consistency rules, if any, to serve their intended purpose. In this section, we reiterate over relevant defects related to consistency rules on DSML language models defined using the UML.

Defects in metamodel definition: The DSML's language model definition does not reference a corresponding metamodel specification, therefore, essential details about the semantics of DSML-specific metaclasses and their relationships are omitted ([12], [17]). In at least five DSML designs, we found an underspecification of metamodel elements. For example, newly introduced metaclasses did not inherit from well-defined base metaclasses (e.g., in the case of a UML metamodel extension, from metaclasses of the UML specification [4]). In such cases, any consistency rule defined for the scope of the underspecified metamodel elements remains ambiguous. This is because it is potentially redundant, restating consistency conditions already available for base metaclasses; or it is potentially conflicting with the latter.

Missing mappings between language model and profile: A frequently observed problem is that a MOF-based or modeling-language independent metamodel is implicitly aligned to a corresponding UML profile. Nevertheless, in at least seven cases, the mapping between metamodel and profile was not documented explicitly ([12], [17]). The lack of explicit documented correspondences lets the reader assume a 1:1 mapping between, for example, non-UML-compliant elements of an initial language-model definition and equally named stereotypes of a UML profile formalization.

Such an implicit mapping, often only based on simple name matching between metamodel and profile, raises the issue of porting any consistency rules from one to the other, which is often not straight forward. A possible approach is to define such mappings between (non-UML-compliant) metamodel and UML profile definitions explicitly. For example, elements of a MOF-based metamodel can be mapped to stereotypes of a UML profile in the form of model-to-model transformations expressed in ATL to ensure their consistency (see, e.g., [20]). This would allow for rendering intended semantics UML-compliant or, if not possible (e.g., semantics of UML stereotypes would contradict the UML specification), providing an explicit trace back to the semantics of the originating metamodel elements. This way, there would also be clear hints how to interpret any consistency rules defined for one artifact (metamodel) in the context of the other (profile).

Defects in profile definition: We identified 21 cases where the definition of a UML profile does not adhere to the UML specification ([12], [17]). Semantic defects encountered included stereotypes inheriting from non-stereotype classes, multiplicity declarations on stereotype extensions, composite aggregation between stereotypes, or inheritance cycles between stereotypes ([12], [17]). These defects introduce semantic variation points (e.g., the possibility of multiple behaviors), which carry over to the interpretation of consistency rules defined over these elements.

Vendor- and tool-specific extensions: In at least three cases ([12], [17]), language models are defined using vendor-specific extensions to OMG specifications that are built into a particular modeling tool (e.g., undefined visibility properties [34]). On the one hand, this raises the issue of defining non-portable consistency rules. On the other hand, to provide consistency between these proprietary additions and elements of the UML metamodel, we recommend specifying their precise semantics in the same way as was done in the UML specification (see, e.g., semantics sub-clauses in [4]).

Defects in constraint-language expressions: We encountered numerous syntactic and semantic defects, including logical errors, calling undefined functions, missing keywords, unbalanced parentheses, and misspelled metamodel elements ([12], [17]). It is obvious that consistency can only be ensured and automatic evaluation can only be provided if formal rules (e.g., OCL expressions) are defect-free. Therefore, increasing the documentation quality of constraint-language expressions is key. Documentation guidelines which require authors to check syntax and semantics of OCL expressions with dedicated tools is a starting point. Table VI provides a non-exhaustive overview of available tools.

VI. Related Work

Our data extraction criteria are closely aligned to the ones presented in [10], [11], in which the authors present a systematic mapping study identifying consistency rules for UML diagrams. The main difference to the approach of Torre et al. is that we focus on domain-specific language models extending the UML, instead of general-purpose UML diagrams. Therefore, our work can be seen as complementary. However, a key difference is that we do not strive for providing an exhaustive collection of concrete consistency rules for UML diagrams in the sense of [10], [11]. Most of the consistency rules for DSMLs are inherently specific to one application domain and to one design of a corresponding language model. As this prevents their general applicability (e.g., to the UML metamodel in general), we did not compile a catalog of consistency rules.

When comparing the collected data with the original work in [10], [11], we can confirm the predominance of consistency rules defined as unstructured text and as OCL expressions, both targeting a single model and multiple models at the same abstraction level (horizontal consistency), for the reviewed DSMLs. Verification and constraint checking are also frequently employed, although we rank heuristics activities first unlike in [10], [11]. This may be due to our data-extraction process, in which we classified each text-based consistency rule as related to the heuristics software-engineering activity as specified by the definition in Section II-C.

Regarding UML model and diagram types, a majority of empirical studies report UML classes as the most exhibited one (see, e.g., [13], [14]). At the same time, we cannot confirm the previously reported high frequency of sequence and state machine diagrams. Similarily, in the review by [10], [11] the otherwise reported frequent adoption of activity, component, and package diagram types is not confirmed. A further confirmatory finding to recent empirical studies (see, e.g., [11]) is the large amount of unique combinations of different diagram types (28). There is also an important overlap regarding supporting tools (Eclipse-based projects, No Magic MagicDraw etc.), but the small tool-specific study population at our side prevents drawing robust conclusions.

VII. Conclusion

In this paper, we analyzed consistency aspects extracted from 84 UML-based DSML designs collected via a SLR [12]. We exclusively focused on consistency rules defined on the level of a DSML's language model. For the evaluation of UML consistency aspects, we adopted criteria from close related work ([10], [11]). By interpreting extracted consistency-related data, we discussed frequently identified defects in UML-based DSML language models. Results of our study show that a UML-based DSML language model is predominantly formalized via profile definitions which tailor mostly class, activity, component, and package diagrams. Textual descriptions and the OCL are most frequently used in combination to define consistency rules on a single model for verification purposes. In the majority of cases, the DSML reports do not document any tool support for enforcing these rules. Results of our study partly confirm findings from as well as add to the observations by related work.

References

[1] D. Spinellis, "Notable design patterns for domain-specific languages," *J. Syst. Softw.*, vol. 56, no. 1, pp. 91–99, 2001.

[2] U. Zdun and M. Strembeck, "Reusable architectural decisions for DSL design: Foundational decisions in DSL development," in *Proc. 14th Europ. Conf. Patt. Lang. Prog.*, 2009.

[3] Object Management Group, "OMG meta object facility (MOF) core specification," Available at: http://www.omg.org/spec/MOF, 2015, version 2.5, formal/2015-06-05.

[4] ——, "OMG unified modeling language (OMG UML)," Available at: http://www.omg.org/spec/UML, 2015, version 2.5, formal/2015-03-01.

[5] L. Nascimento, D. L. Viana, P. A. M. S. Neto, D. A. O. Martins, V. C. Garcia, and S. R. L. Meira, "A systematic mapping study on domain-specific languages," in *Proc. 7th Int. Conf. Softw. Eng. Adv.* IARIA, 2012, pp. 179–187.

[6] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Proc. 33rd Int. Conf. Softw. Eng.* ACM, 2011, pp. 471–480.

[7] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Softw. Pract. Exper.*, vol. 39, no. 15, pp. 1253–1292, 2009.

[8] Object Management Group, "Object constraint language," Available at: http://www.omg.org/spec/OCL, 2014, version 2.4, formal/2014-02-03.

[9] J. Simmonds, R. V. D. Straeten, V. Jonckers, and T. Mens, "Maintaining consistency between UML models using description logic," *RSTI – L'Objet*, vol. 10, no. 2-3, pp. 231–244, 2004.

[10] D. Torre, Y. Labiche, and M. Genero, "UML consistency rules: A systematic mapping study," in *Proc. 18th Int. Conf. Eval. Assess. Softw. Eng.* ACM, 2014, pp. 6:1–6:10.

[11] D. Torre, Y. Labiche, M. Genero, and M. Elaasar, "A systematic identification of consistency rules for UML diagrams," Available at: http://squall.sce.carleton.ca/pubs/tech_report/TR_SCE-15-01.pdf, Carleton University, Tech. Rep. SCE-15-01, 2015.

[12] S. Sobernig, B. Hoisl, and M. Strembeck, "Extracting reusable design decisions in UML-based domain-specific languages: A multi-method study," submitted.

[13] D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, "Empirical evidence about the UML: A systematic literature review," *Softw. Pract. Exper.*, vol. 41, no. 4, pp. 363–392, 2011.

[14] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Sci. Comput. Program.*, vol. 89, Part B, pp. 144–161, 2014.

[15] E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*, 1st ed. Addison-Wesley, 2004.

[16] T. Mens and P. v. Gorp, "A taxonomy of model transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, 2006.

[17] S. Sobernig, B. Hoisl, and M. Strembeck, "Protocol for a systematic literature review on design decisions for UML-based DSMLs," Available at: http://epub.wu.ac.at/4467/, WU Vienna, Tech. Rep. 2014/02, 2015.

[18] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inform. Softw. Tech.*, vol. 53, no. 6, pp. 625–637, 2011.

[19] S. Jalali and C. Wohlin, "Systematic literature studies: Database searches vs. backward snowballing," in *Proc. ACM-IEEE Int. Sym. Empir. Softw. Eng. Meas.* ACM, 2012, pp. 29–38.

[20] B. Hoisl, S. Sobernig, and M. Strembeck, "A catalog of reusable design decisions for developing UML/MOF-based domain-specific modeling languages," Available at: http://epub.wu.ac.at/4466/, WU Vienna, Tech. Rep. 2014/03, 2015.

[21] J. Bézivin and F. Jouault, "Using ATL for checking models," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 69–81, 2006.

[22] A. S.-B. Herrera, E. Willink, R. Zanzeebarr, A. Igdalov, C. W. Damus, and N. Boldt, "OCL/FAQ – Eclipsepedia," Available at: http://wiki.eclipse.org/OCL/FAQ, 2015.

[23] B. Kitchenham, "Procedures for performing systematic reviews," Keele University & National ICT Ltd., Joint Tech. Rep. (Keele University Tech. Rep., NICTA Tech. Rep.) TR/SE-0401, 0400011T.1, 2004.

[24] P. Langer, K. Wieland, M. Wimmer, and J. Cabot, "EMF profiles: A lightweight extension approach for EMF models," *J. Object Technol.*, vol. 11, no. 1, pp. 1–29, 2012.

[25] S. Schefer and M. Strembeck, "Modeling support for delegating roles, tasks, and duties in a process-related RBAC context," in *Proc. Int. Worksh. Inform. Syst. Secur. Eng.*, ser. LNBIP, vol. 83. Springer, 2011, pp. 660–667.

[26] S. Schefer-Wenzl and M. Strembeck, "An approach for consistent delegation in process-aware information systems," in *Proc. 15th Int. Conf. Bus. Inform. Syst.*, ser. LNBIP, vol. 117. Springer, 2012, pp. 60–71.

[27] C. Atkinson and T. Kühne, "Processes and products in a multi-level metamodeling architecture," *Int. J. Softw. Eng. Know.*, vol. 11, no. 6, pp. 761–783, 2001.

[28] ——, "A tour of language customization concepts," *Adv. Comput.*, vol. 70, pp. 105–161, 2007.

[29] D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige, "The Epsilon book," Available at: http://www.eclipse.org/epsilon/doc/book/, 2015.

[30] E. Cariou, C. Ballagny, A. Feugas, and F. Barbier, "Contracts for model execution verification," in *Model. Found. Applicat.*, ser. LNCS. Springer, 2011, vol. 6698, pp. 3–18.

[31] T. Schattkowsky, J. Hausmann, and G. Engels, "Using UML activities for system-on-chip design and synthesis," in *Model Driven Eng. Lang. Syst.*, ser. LNCS. Springer, 2006, vol. 4199, pp. 737–752.

[32] J. Bergh and K. Coninx, "CUP 2.0: High-level modeling of context-sensitive interactive applications," in *Model Driven Eng. Lang. Syst.*, ser. LNCS. Springer, 2006, vol. 4199, pp. 140–154.

[33] N. Ubayashi, T. Tamai, S. Sano, Y. Maeno, and S. Murakami, "Model compiler construction based on aspect-oriented mechanisms," in *Gen. Program. Compon. Eng.*, ser. LNCS. Springer, 2005, vol. 3676, pp. 109–124.

[34] E. Soler, J. Trujillo, E. Fernández-Medina, and M. Piattini, "Building a secure star schema in data warehouses by an extension of the relational package from CWM," *Comp. Stand. Inter.*, vol. 30, no. 6, pp. 341–350, 2008.