Using Incremental Consistency Management for Conformance Checking in Feature-Oriented Model-Driven Engineering

Roberto E. Lopez-Herrejon
Alexander Egyed
Institute for Systems Engineering and Automation
Johannes Kepler University Linz, Austria
{roberto.lopez, alexander.egyed}@jku.at

Salvador Trujillo
Josune de Sosa
IKERLAN Research Laboratory
Mondragon, Spain
{STrujillo,jdesosa}@ikerlan.es

Maider Azanza
University of the Basque Country
San Sebastian, Spain
maider.azanza@ehu.es

***Abstract*—Feature-Oriented Model-Driven Engineering (FOMDE) is an approach that lies at the intersection of two complementary paradigms for software construction, Model Driven Engineering (MDE) and Software Product Line Engineering (SPLE). MDE aims at raising the abstraction level of application specification and automating the realization of these abstractions down to the platform level, while SPLE focuses on the synthesis of applications using a pre-planned set of assets. In Feature-Orientation, features are modules that contain all assets needed for their realization. The products of a Software Product Line (SPL) are synthesized by composing different combinations of features. When constructed following MDE, features also contain metamodels, models and model transformations. In this context, it is crucial to check that models, metamodels, and their compositions conform to (i.e. meet all the constraints of) their metamodels and meta-metamodels. In this problem statement paper we describe how to use incremental consistency checking to check this conformance. We sketch some of the potential benefits of this approach and highlight the open questions our work raised.**

***Index Terms*—incremental consistency management; Model-Driven Engineering Software Product Lines; Feature Orientation; conformance checking**

I. Introduction

Feature-Oriented Model-Driven Engineering (FOMDE) is an approach that lies at the intersection of two complementary paradigms for software construction, *Model Driven Engineering (MDE)* [1] and *Software Product Line Engineering (SPLE)* [2]–[4]. MDE aims at raising the abstraction level of application specification and automating the realization of these abstractions down to the platform level with a set of model to model and model to text transformations. On the other hand, SPLE focuses on the synthesis of applications using a pre-planned set of assets.

Feature-Orientation is an specific approach for constructing *Software Product Lines (SPL)* [5], [6]. In this approach, *features* are increments in program functionality [7]. Features are implemented in modules that contain all assets, that is, artifacts that they require for their realization. For example, a feature may be implemented with UML diagrams, scripts, XML files, configuration files, etc. The members of the SPL are synthesized by composing different combinations of features. Tools that implement this approach provide mechanisms to compose these distinct artifact types in a uniform way. For the combination of Feature-Orientation and MDE, the features typically contain multiple models, metamodels and transformations [8].

In this context, it is crucial to check that models, metamodels, and their compositions conform to (i.e. meet all the constraints of) their metamodels and meta-metamodels respectively. The driving motivation of our problem statement paper is describing an approach to check this conformance.

Consistency checking derives from work on *Multi-View Modeling (MVM)* [9]. MVM advocates that multiple, different and yet related models are required to represent the perspectives and information needs of diverse system stakeholders throughout the development process [10], [11]. The elements in these distinct views have semantic relationships that must be expressed and maintained. Consistency rules capture and serve to enforce these semantic relationships. An example of MVM is UML where the different types of diagrams can represent the distinct views of a system [12]. A classical example of consistency rule in UML, between sequence and class diagrams, is that if a sequence diagram has a message `m` whose target is an object of class `C`, the class diagram of class `C` must contain method `m`.

Our work raises consistency checking beyond the traditional MVM perspective. We show how incremental consistency checking, a special form of consistency checking, can be used to check conformance as described above. The key is treating both model and metamodel composition similarly and generating conformance rules based on the well-formedness rules defined at the meta-metamodel. We sketch some of the potential benefits of this approach and highlight the open questions our work raised.

II. Background

Our work brings together two, until now, disjoint research areas. In this section we present the basic background of both.

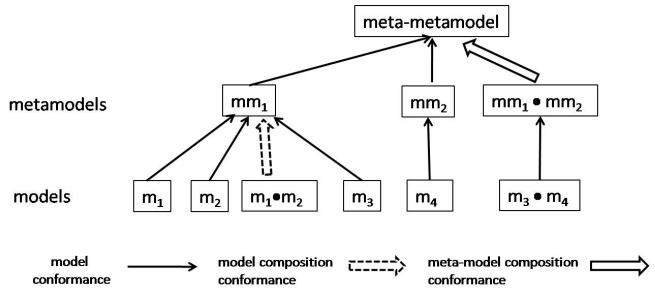


Fig. 1. Scenarios where model conformance is used

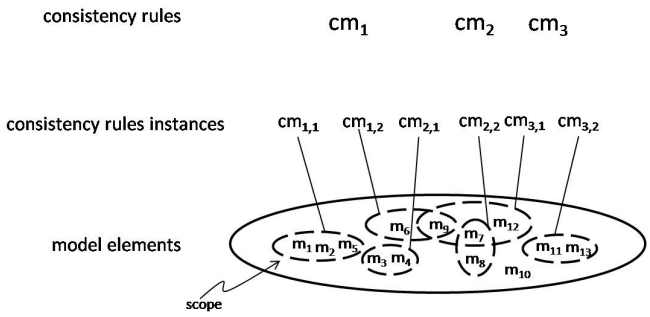


Fig. 2. Pictorial view of Incremental Consistency Checking

A. Feature-Oriented Model-Driven Engineering

Feature Oriented Model Driven Engineering (FOMDE) is a blend of *Feature Oriented Programming (FOP)* [6] and MDE that shows how products in an SPL can be synthesized in an MDE way by composing features to create models, and then transforming these models into executables [8]. FOP and MDE are complementary paradigms [13] and several case studies show the advantages of combining them [14], [15]. However, in these cases, features were implemented using XML and they were composed using XAK [16], that is, feature composition was purely text-based. This work laid out the foundations for our current research on FOMDE. The lack of conformance checking in FOMDE captured our interest and motivated this research.

Recent work on *Domain Specific Languages (DSL)* has raised the need of reusing metamodels as features of a SPL [17], [18]. The selection of different features produces thus different DSLs that are tailored to particular application scenarios. For this scenario to work properly, it is crucial to check that the composition of metamodels conforms to the meta-metamodel used and their corresponding models are kept conformant when their metamodels change.

In summary, in the context of FOMDE there are three main scenarios where checking conformance is important: *i)* model conformance to metamodel, *ii)* conformance during model composition, and *iii)* conformance during metamodel composition. These three scenarios are depicted in Figure 1. In this figure, $\mathtt{m}$ stands for model and $\mathtt{mm}$ for metamodel, and the dot indicates composition.

B. Incremental Consistency Checking

There exists an extensive body of work in consistency checking. Recent literature surveys identified over 30 approaches which rely on different formalisms to represent and validate consistency [19], [20]. They typically have in common that consistency is expressed via rules. A recent trend in consistency checking is work on incremental approaches which react to changes and evaluate only those rules on those model elements that are affected and can potentially cause an inconsistency. An advantage of these approaches is reduced verification time over systems that follow a batch strategy. A leading tool among the incremental approaches is UML/Analyzer [21], [22]. In this tool, when a model change occurs, it automatically, correctly and efficiently identifies what consistency rules to evaluate and on what model elements. If inconsistencies are detected, they are highlighted for the user to take an appropriate corrective action.

Incremental consistency in UML/Analyzer works as follows. First the tool loads the model to analyze. Then it identifies the places where each consistency rule defined can be applied. A *consistency rule instance* is an application of a consistency rule, and its scope is the set of model elements that are part of the instance.

It is common that a model element is part of the scopes of multiple and distinct consistency rules instances. An example of this scenario is shown in Figure 2. This figure shows three consistency rules $\mathtt{cm}_1$, $\mathtt{cm}_2$, and $\mathtt{cm}_3$. For notational convention we denote these constraint rules with a suffix $\mathtt{m}$, that stands for metamodel because these rules are defined in terms of metamodel elements, and a number subscript. We use a second subscript to denote instances of the constraint rules. Figure 2 shows instances of these rules such as $\mathtt{cm}_{1,1}$ and $\mathtt{cm}_{2,1}$. In this figure, model elements $\mathtt{m}_7$ and $\mathtt{m}_9$ belong to two distinct scopes.

The work of UML/Analyzer has been mostly used in the context of UML models; however, its underlying principles are applicable to any types of models and constraints [21], [22]. In the next section we show how these principles are adapted for checking model and metamodel conformance in the context of FOMDE.

III. Conformance Checking

In this section we draw a connection between incremental consistency checking and conformance checking for the three scenarios described above and sketch the potential benefits of this connection.

A. Conformance of Model to Metamodel

Let us illustrate the key ideas of UML/Analyzer with an example. Consider a hypothetical SPL of questionnaires. A typical questionnaire has a title and a brief introduction. The questions are grouped in blocks that have a header and a description. A block needs to have at least one question, and for each question requires two to four answer options. Figure 3 presents the metamodel for the features of this product line as an Ecore metamodel (a subset of UML class diagram).

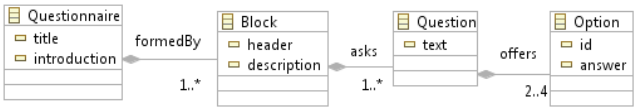


Fig. 3. Questionnaire metamodel

In FOMDD a feature contains models that are instances of a metamodel. Feature `F`, depicted in Figure 4, is an example in our questionnaire product line. For sake of simplicity we use an abbreviated object model to denote instantiation of the metamodel and annotate the relations amongst the objects using the aggregation names of the metamodel (`formedBy`, `asks`, and `offers`). This feature contains a block `B1` with two questions (`A` and `B`), each with two answer options.

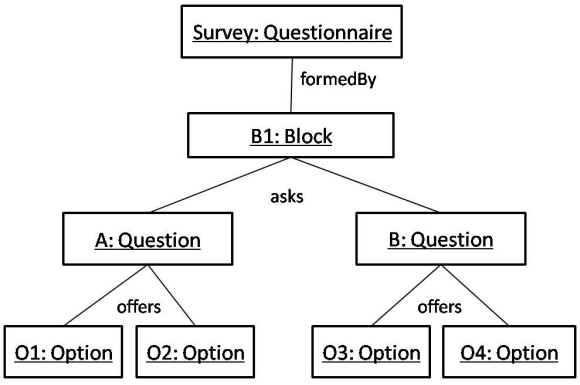
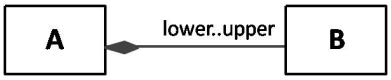


Fig. 4. Feature F, model instance of Questionnaire metamodel

The key for leveraging incremental consistency checking is using conformance rules as the consistency rules to check against. In other words, conformance rules and consistency rules have in common that they evaluate a portion of a model and return a boolean result, true if the rule holds or false otherwise. Consistency checkers can thus be used readily to check conformance rules. In our `Questionnaire` one of such conformance rules can be defined as follows [1]:

Conformance rule for aggregation. Let `A` and `B` be two classes. If an aggregation between `A` and `B` exists, an object of type `A` can aggregate `n` objects of type `B` where `lower` $\leq$ `n` $\leq$ `upper`.



Now we describe how to check conformance of the model in Figure 4 against this rule. In this figure, there are four instances of the conformance aggregation rule. The first instance contains `Question A` and its two `Option` answers from `offers` aggregation. Similarly, the second instance contains `Question B` and its two answers `Option`. The third instance contains a `Block` and the two questions, from `asks` aggregation. Finally, the fourth instance contains a `Questionnaire` object and one `Block`, from `formedBy` aggregation. In this model, the number of objects aggregated falls within the lower and upper limits of the corresponding rule instances, thus it conforms to its metamodel, according to this rule.

[1]To be more precise, the filled rhomb in this association denotes containment such that an object of type `B` can only be associated with one object of type `A`. For simplicity, we do not utilize this part of the standard semantics of this symbol as it is not relevant for our current exposition.

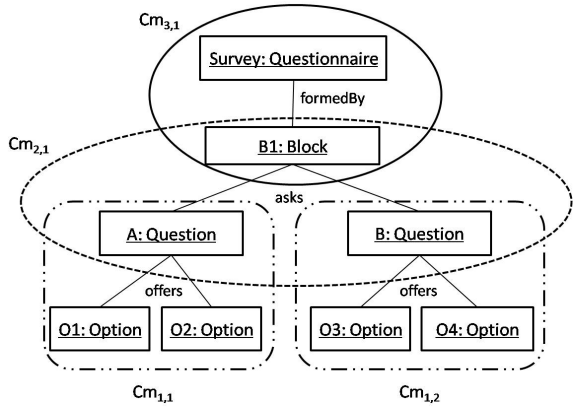


Fig. 5. Feature F with scopes

Notice however that this general conformance rule can be fine-tuned according to the types of the classes participating in the aggregation and their cardinality. We call this adaptation process *constraint rule generation*. For our `Questionnaire` metamodel the generated conformance rules are:

- cm_1 = Aggregation rule with `A` is `Question` and `B` is `Option`
- cm_2 = Aggregation rule with `A` is `Block` and `B` is `Question`
- cm_3 = Aggregation rule with `A` is `Questionnaire` and `B` is `Block`

The four instances of these generated rules are depicted in Figure 5. Thus, every feature will have an associated set of conformance rule instances with their corresponding scopes. In this figure the scopes are [2]:

- $cm_{1,1}$ = {`A, O1, O2`}
- $cm_{1,2}$ = {`B, O3, O4`}
- $cm_{2,1}$ = {`B1, A, B`}
- $cm_{3,1}$ = {`Survey, B1`}

Our conformance rule for aggregation is a simplified example of the well-formedness rules that are commonly defined for meta-metamodels such as Ecore [23] and MOF [24]. Therefore, to check conformance of a model to a metamodel it is required to define all well-formedness rules from which conformance rules can be generated for a particular metamodel.

In addition to the well-formedness rules, it is possible to include constraints that are specific to a domain. An example in our domain questionnaire would be requiring that the maximum number of questions per questionnaire be 40 questions irrespective of how they are grouped into blocks. These

[2]For notational simplicity, we equate the scopes with the rule instances.

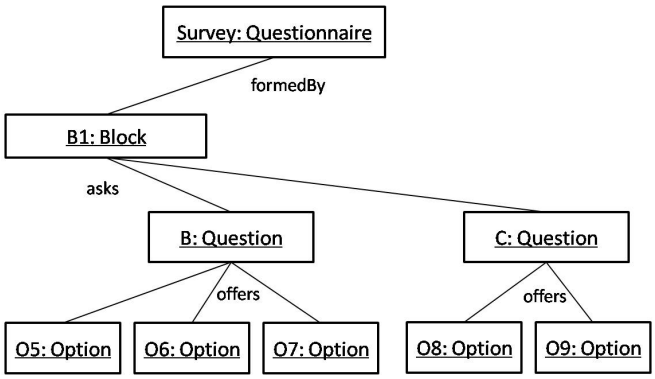


Fig. 6. Feature G, instance of Questionnaire metamodel

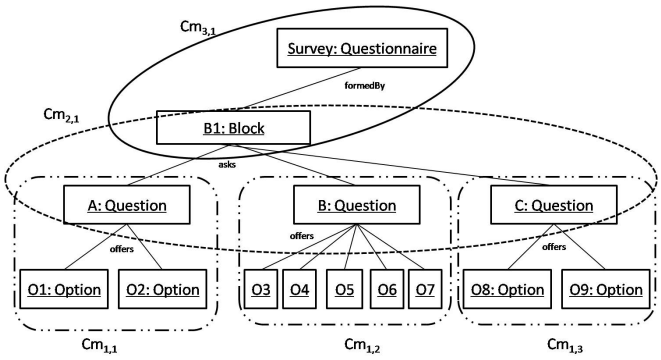


Fig. 7. Composed feature with scopes

domain-specific conformance rules are treated identically to those generated from well-formedness rules.

In summary, the conformance of a model to a metamodel can be checked using incremental consistency checking where the constraints rules are: *i)* rules generated from the well-formedness rules of the meta-metamodel that apply in a metamodel, *ii)* domain-specific constraints defined for the metamodel.

B. Conformance during Model Composition

Let us describe now how model conformance is checked during model composition. Consider the feature `G` in Figure 6. This feature also has a block `B1` with `Question B` with three new `Option` answers, and another `Question C` with its two `Option` answers. It is clear from our description above that this feature conforms to its metamodel as all the conformance rules instances are valid. Regardless of the technology used [25], model composition can be seen as applying a successive set of changes to an existing model. In our example, the composition of this second feature `G` to feature `F` in Figure 4 adds three new options to question `B`, and a new question `C` with its options.

The first step prior to start composition is cloning copies of the features involved and their scopes. On these copies will composition and conformance checking be performed. This step is necessary because features can be used to compose different products. In FOMDE, features are composed hierarchically starting from the root element. Elements that have the same name and type at the same hierarchical level are composed together, elements that do not have a corresponding matching element are copied along hierarchically. In our example, elements `Survey`, `B1` and `B` from feature `G` have a matching element in feature `F` thus they will compose hierarchically. The remaining elements in feature `G` are copied along at their corresponding level.

As composition proceeds, a rule instance is re-evaluated if a change in its scope elements is detected. Additionally, if model elements are deleted or new ones are added, new rule instances can be removed or created. The result of composing our two features is depicted in Figure 7 with their corresponding consistency rule scopes. For visual simplicity, the types `Option` of the objects of question `B` are omitted in the figure.

In our example, the addition of options `O5`, `O6`, and `O7`, causes a re-evaluation of rule instance $\texttt{cm}_{1,2}$. Recall that rule $\texttt{cm}_1$ checks the aggregation between `Option` and `Question`, such that each question has from two to four possible answers. Therefore, this instance re-evaluation detects a violation of this rule because `Question B` now has five available options. It is important to notice that this violation is signaled as soon as option `O7` is added. This immediate notification allows the developer to take any corrective actions deemed necessary. A possibility is backtracking composition to trace the source of the non-conformance.

Continuing with the composition, the addition of the new `Question C` creates a new instance $\texttt{cm}_{1,3}$ whose evaluation meets the conformance rule. Because there was a change in the scope of $\texttt{cm}_{2,1}$ resulting from the addition of a new question, this instance is also re-evaluated. Recall that rule $\texttt{cm}_2$ checks aggregations between `Block` and `Question` such that a block has at least one question. Thus the re-evaluation of $\texttt{cm}_{2,1}$ does not detect any inconsistencies. In conclusion, the only non-conformance to the metamodel of the composed features is because five options are available for `Question B`.

Summarizing, conformance during model composition follows the same process described in the previous section. The insight here is considering the composition of a feature with another as applying a set of finer-grain model changes to another model.

C. Conformance during Metamodel Composition

The same principles of incremental consistency checking are applicable for composing metamodels. To illustrate that, first consider Figure 8 that shows a metamodel feature that has an aggregation for `Block` to itself, and a navigable association from `Block` to a new class `Scale`. We will compose this metamodel with `Base` metamodel in Figure 3.

Let us explain how incremental consistency works when metamodels are composed. The first consideration to keep in mind is that a metamodel is in itself an instance of a meta-metamodel. Thus a metamodel can be viewed as a set of instances of meta-metaclasses. For example, using Ecore

Fig. 8. Scale metamodel and feature

[23], Figure 9 shows a simplified view of the questionnaire metamodel in Figure 3. A package (meta-metaclass `EPackage`) aggregates zero or more classes (meta-metaclass `EClass`) in an aggregation called `eClassifiers`. In turn, each `EClass` instance aggregates its attributes (meta-metaclass `EAttribute`) and its references to other classes (meta-metaclass `EReference`). Note that these references are the aggregations between the metaclasses in Figure 3. For example, the `EReference formedBy` in `EClass Questionnaire` corresponds to the `formedBy` aggregation between metaclasses `Questionnaire` and `Block`[3].

Using this perspective of considering metaclasses as instances (model elements) of the meta-metamodel, we can apply exactly the same process we followed for checking conformance with model composition. First, clone copies of the metamodels and their scopes are made to apply composition and conformance checking on them.

Two conformance rules that check aggregation, but now at the metamodel level, can be generated. We use suffix `mm` to denote these rules as they are now defined in terms of meta-metamodel elements as follows:

- $\texttt{cmm}_1$ = Aggregation rule with `A` is `EPackage` and `B` is `EClass`
- $\texttt{cmm}_2$ = Aggregation rule with `A` is `EClass` and `B` is `EStructuralFeature`

[3]For simplicity the endType of the EReference is not depicted. For instance in the case of formedBy this type is metaclass Block.

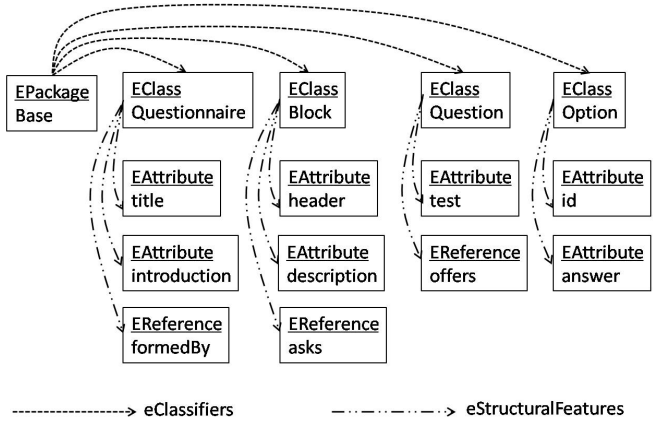


Fig. 9. Simplified metamodel view in terms of metaclasses

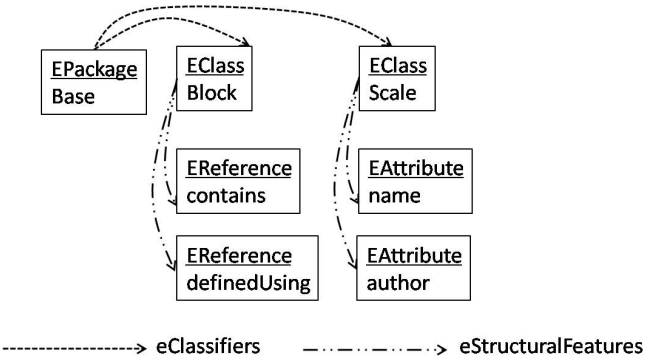


Fig. 10. Simplified metamodel view of Scale

In Ecore, `EStructuralFeature` is an interface implemented by both `EClass` and `EReference`. Furthermore, we can now identify the following instances of these rules and their corresponding scopes in `Questionnaire` metamodel:

- $\texttt{cmm}_{1,1}$ = {`Base, Questionnaire, Block, Question, Option`}
- $\texttt{cmm}_{2,1}$ = { `Questionnaire, title, introduction, formedBy` }
- $\texttt{cmm}_{2,2}$ = {`Block, header, description, asks` }
- $\texttt{cmm}_{2,3}$ = {`Question, test, offers`}
- $\texttt{cmm}_{2,4}$ = {`Option, id, answer`}

Using this same perspective, the metaclasses view of `Scale` metamodel is depicted in Figure 10. Metamodel composition, performed along the lines illustrated in previous section, modifies the scopes of the rule instances $\texttt{cmm}_{1,1}$ and $\texttt{cmm}_{2,2}$ (changes are underlined) and creates a new rule instance $\texttt{cmm}_{2,5}$ as follows:

- $\texttt{cmm}_{1,1}$ = {`Base, Questionnaire, Block, Question, Option,` <u>`Scale`</u> }
- $\texttt{cmm}_{2,2}$ = {`Block, header, description, asks,` <u>`contains`</u>, <u>`definedUsing`</u> }
- $\texttt{cmm}_{2,5}$ = {`Scale, name, author`}

Consequently instances $\texttt{cmm}_{1,1}$ and $\texttt{cmm}_{2,2}$ need to be re-evaluated, and instance $\texttt{cmm}_{2,5}$ evaluated for a first time. In this case these instances do not cause any conformance violations as they meet the constraint given that a package can have zero or more classes, and a class can have zero or more attributes and references. In other words, the composed metamodel conforms to the meta-metamodel.

Despite of not causing any inconsistency, the changes in the metamodel can still trigger the generation of new rule instances as new metaclasses can be added. In this example, the addition of `EReference contains` causes the generation of a new instance $\texttt{cm}_4$ of the aggregation rule with `A` is `Block` and `B` is `Block`. This generation in turn triggers a search for instances of $\texttt{cm}_4$ at the model level. In our model composition examples we have no such case, so the checking process stops there.

In summary, checking conformance of metamodel composition follows the same process as the case of model composition

but with the additional step that changes at the metamodel level can trigger the generation of new rule instances or the re-evaluation of existing instances at the model level.

D. Potential benefits

Based in our experience, early conformance checking of features by means of incremental consistency checking can offer three major advantages when compared to that batch checking:

- Consistency of modeling artifacts throughout the entire development process, including their correctness and well-formedness [26].
- Earlier identification of inconsistencies.
- Traceability of the origin of the inconsistency.

IV. Related Work

There is extensive research on models, model composition and SPL. In this section we shortly present those pieces of research that most closely relate to our work.

Safe composition is the guarantee that programs composed according to the product line constraints are type safe [27], i.e. they do not have undefined elements to structural elements such as classes, methods, and fields. Contrary to this paper that focuses on checking conformance of a given product, safe composition focuses on validating properties for all members of a product line. Our recent work has shown how to use UML consistency rules as the constraints to validate safe composition in UML-based SPL [28].

There are several approaches and technologies to perform model composition [25]. Only a few are specific to SPL. Trujillo et. al motivate the need of realizing variability not only at model level but also at metamodels and model transformations [18]. FeatureHouse uses model superimposition to compose basic UML models at the XMI level [29], and MATA uses graph transformations as composition mechanisms for UML models [30]. However, in these two approaches it is unclear how (if at all) conformance checking is performed.

Another approach implements feature composition uses Maude, a high-performance logical framework [31]. In this work, feature composition is expressed in terms of rewrite rules. Conformance checking becomes reduction according to the rewrite rules, which in our context effectively are our conformance rules. In other words, if a composed model reduces to a canonical form (one that cannot be further reduced), the model conforms, otherwise an error can be detected. We have not investigated how this approach could be tailored to represent arbitrary domain-specific constraints. This issue is part of our future work.

V. Open Questions

In this section we sketch some of the open questions we identified in our work, as such, they are venues for our future work.

Living with inconsistencies. In this paper we assumed that the composition of models and metamodels should at anytime conform respectively to their metamodels and metametamodels. However, there may be intermediate stages during composition at which this assumption may not hold, but still yield a conforming result at the end. For example, if the composition paradigm used were non-monotonic (permitting to remove model elements) and our features `F` and `G` were composed with a third feature that removed one of the options of `Question B`, the result would be a conforming model in despite of the partial composition of `F` and `G` being non conformant. This type of inconsistency is *tolerable* as composition may potentially be able to "fix" it. There may be also cases where an inconsistency cannot be remedied. A case from our `Questionnaire` metamodel would be a feature that contains two `Option` objects without any associations. This type of inconsistency is *intolerable* because for this feature to be composable the `Question`, the `Block`, and the `Questionnaire` the options belong to must be also defined. Thus, living with inconsistencies [32], [33] (tolerating some of them) also plays an important role in our work. Characterizing, identifying and managing both types of inconsistencies may have an impact on how we define and implement our conformance rules.

Impact of a change. We expect changes not to be isolated. Constraint instances may have complex relationships amongst them in such a way that a single change may trigger a cascade of inconsistencies for which subsequent fixes may be required. Efficiently determining the impact of a change and computing an order in which to fix the triggered inconsistencies may be a crucial point for our approach to adequately scale.

Consistency at other development stages. Conceivably, there are other scenarios where changes can also occur and thus conformance checking may become necessary. An example is when features, either models or metamodels, themselves change as consequence of changes in the requirements. These changes may themselves trigger conformance checks of the modified models and metamodels. Another possible scenario is when concrete products evolve and such changes must be propagated back into the SPL architecture and its features. In summary, we plan to study all other possible scenarios where conformance checking may be needed and evaluate the applicability of incremental consistency to them.

Consistencies between feature artifacts. Our paper focused on checking conformance within an artifact type, namely models or metamodels. However, it is common that a feature involves more than one artifact type, such as code, models, XML files, script files, etc. Thus it is important to keep consistency amongst the elements of a feature. Our recent work has started to address this issue with UML artifacts [28].

Evolution direction. In our work, when changes occur at the metamodel level, other changes can be triggered down at the model level. However, it is conceivable that changes may flow in the opposite direction. This means that a change committed at the model level imposes changes at the metamodel level which in turn may trigger other changes at other instances of the metamodel.

Safe composition. The work presented in this paper focuses on checking the conformance of one concrete product. The goal of safe composition is the verification that certain

constraints are met in all the possible configurations (allowable combinations of features) of a product line. However, because SAT solvers are used for this validation, there may be scalability issues as the size of feature models, the types of constraints, and number artifact types increase. Knowing those potential limitations could help provide guidance on how to extend safe composition to address the above mentioned open questions.

Consistency between variability space and solution space. This goes a step beyond safe composition by not only detecting violation of the variability at the implementation level but also in attempting to keep consistent variability defined in a feature model with its realization across multiple artifacts. We believe that the intensive ongoing research in formal analysis of feature models can provide a foundation to address this question [34]–[36].

VI. Conclusions and Future Work

In this paper we drew a connection between FOMDE and incremental consistency checking. A crucial need in FOMDE is checking conformance in different scenarios: model to metamodel, during model composition, and during metamodel composition. We showed that the underlying principles of incremental consistency checking are applicable for checking conformance in these three scenarios. The key is using as consistency rules the conformance rules that are generated from meta-metamodel well-formedness specifications or constraints that are domain-specific.

As a first step, we plan to develop a metamodel-independent framework to facilitate the specification of constraint rules and their subsequent generation. We will use this framework to evaluate our approach in industry-motivated cases studies and address the identified open questions.

Acknowledgment

We thank Laura Vozmediano for her help with the questionnaires domain. This research is partially sponsored by the Austrian FWF under agreement P21321-N15. This work is co-supported by the Spanish Ministry of Science and Innovation, under contracts TIN2008-06507-C02-01 and TIN2008-06507-C02-02.

References

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

[2] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[3] K. Pohl, G. Bockle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[4] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[5] D. S. Batory, R. E. Lopez-Herrejon, and J.-P. Martin, "Generating product-lines of product-families," in *ASE*. IEEE Computer Society, 2002, pp. 81–92.

[6] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement," *IEEE TSE*, vol. 30, no. 6, 2004.

[7] P. Zave, "Faq sheet on feature interaction," http://www.research.att.com/ pamela/faq.html.

[8] S. Trujillo, D. Batory, and O. Diaz, "Feature Oriented Model Driven Development: A Case Study for Portlets," in *ICSE*, 2007.

[9] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: A framework for integrating multiple perspectives in system development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 1, pp. 31–57, 1992.

[10] A. Finkelstein, D. M. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency handling in multperspective specifications," *IEEE Trans. Software Eng.*, vol. 20, no. 8, pp. 569–578, 1994.

[11] B. Nuseibeh, J. Kramer, and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Trans. Software Eng.*, vol. 20, no. 10, pp. 760–773, 1994.

[12] "Unified Modeling Language (UML)," 2008, http://www.uml.org.

[13] D. S. Batory, M. Azanza, and J. Saraiva, "The objects and arrows of computational design," in *MoDELS*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds., vol. 5301. Springer, 2008, pp. 1–20.

[14] G. Freeman, D. S. Batory, and R. G. Lavender, "Lifting transformational models of product lines: A case study," in *ICMT*, ser. Lecture Notes in Computer Science, A. Vallecillo, J. Gray, and A. Pierantonio, Eds., vol. 5063. Springer, 2008, pp. 16–30.

[15] E. Uzuncaova, D. Garcia, S. Khurshid, and D. S. Batory, "A specification-based approach to testing software product lines," in *ESEC/SIGSOFT FSE*, I. Crnkovic and A. Bertolino, Eds. ACM, 2007, pp. 525–528.

[16] F. I. Anfurrutia, O. Diaz, and S. Trujillo, "On the Refinement of XML," in *International Conference on Web Engineering ICWE*, 2007.

[17] J. White, J. H. Hill, J. Gray, S. Tambe, A. S. Gokhale, and D. C. Schmidt, "Improving domain-specific language reuse with software product line techniques," *IEEE Software*, vol. 26, no. 4, pp. 47–53, 2009.

[18] S. Trujillo, A. Zubizarreta, X. Mendialdua, and J. de Sosa, "Feature-oriented refinement of models, metamodels and model transformations," in *FOSD*, ser. ACM International Conference Proceeding Series, S. Apel, W. R. Cook, K. Czarnecki, C. Kästner, N. Loughran, and O. Nierstrasz, Eds. ACM, 2009, pp. 87–94.

[19] F. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," in *To appear Information and Software Technology*, 2009.

[20] M. Usman, A. Nadeem, T.-H. Kim, and E.-S. Cho, "A survey of consistency checking techniques for uml models," in *Advanced Software Engineering and Its Applications, 2008. ASEA 2008*, 2008, pp. 57–62. [Online]. Available: http://dx.doi.org/10.1109/ASEA.2008.40

[21] A. Egyed, "Instant consistency checking for the uml," in *ICSE*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds. ACM, 2006, pp. 381–390.

[22] ——, "Fixing inconsistencies in uml design models," in *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 292–301.

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

[24] OMG, "Meta Object Facility (MOF)," 2010, http://www.omg.org/mof.

[25] C. Jeanneret, "An analysis of model compostion approaches," Master's thesis, Ecole Polytechnique Federal de Lausanne, 2008.

[26] F. Heidenreich, "Towards systematic ensuring well-formedness of software product lines," in *FOSD*, ser. ACM International Conference Proceeding Series, S. Apel, W. R. Cook, K. Czarnecki, C. Kästner, N. Loughran, and O. Nierstrasz, Eds. ACM, 2009, pp. 69–74.

[27] S. Thaker, D. S. Batory, D. Kitchin, and W. R. Cook, "Safe composition of product lines," in *GPCE*, C. Consel and J. L. Lawall, Eds. ACM, 2007, pp. 95–104.

[28] R. E. Lopez-Herrejon and A. Egyed, "Detecting inconsistencies in multi-view models with variability," submitted for publication.

[29] S. Apel, C. Kästner, and C. Lengauer, "Featurehouse: Language-independent, automated software composition," in *ICSE*. IEEE, 2009, pp. 221–231.

[30] P. Jayaraman, J. Whittle, A. Elkhodary, and H. Gomaa, "Model Composition and Feature Interaction Detection in Product Lines using Critical Pair Analysis," in *MoDELS*, 2007.

[31] R. E. Lopez-Herrejon and J. E. Rivera, "Realizing feature oriented software development with equational logic: An exploratory study," in *JISBD*, A. Vallecillo and G. Sagardui, Eds., 2009, pp. 269–274.

[32] R. Balzer, "Tolerating inconsistency," in *ICSE*, 1991, pp. 158–165.

[33] S. Fickas, M. Feather, and J. Kramer, "Living with inconsistency. icse workshop, boston, usa," 1997.

[34] D. Benavides, A. R. Cortés, D. S. Batory, and P. Heymans, "First international workshop on analysis of software product lines (aspl'08)," in *SPLC*. IEEE Computer Society, 2008, p. 385.

[35] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. R. Cortés, "Automated diagnosis of product-line configuration errors in feature models," in *SPLC*. IEEE Computer Society, 2008, pp. 225–234.

[36] D. Batory, "Feature Models, Grammars, and Propositional Formulas," in *Proceedings of the International Software Product Line Conference (SPLC)*, 2005, pp. 7–20.

[37] *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings*. IEEE Computer Society, 2008.

[38] S. Apel, W. R. Cook, K. Czarnecki, C. Kästner, N. Loughran, and O. Nierstrasz, Eds., *Proceedings of the First International Workshop on Feature-Oriented Software Development, FOSD 2009, Denver, Colorado, USA, October 6, 2009*, ser. ACM International Conference Proceeding Series. ACM, 2009.