

# Validation of Derived Features and Well-Formedness Constraints in DSLs<sup>\*</sup>

## By mapping graph queries to an SMT-solver

Oszkár Semeráth, Ákos Horváth, and Dániel Varró

Budapest University of Technology and Economics,  
Department of Measurement and Information Systems,  
1117 Budapest, Magyar tudósok krt. 2.  
so765@hszk.bme.hu, {ahorvath,varro}@mit.bme.hu

**Abstract.** Despite the wide range of existing generative tool support, constructing a design environment for a complex domain-specific language (DSL) is still a tedious task as the large number of derived features and well-formedness constraints complementing the domain metamodel necessitate special handling. Incremental model queries as provided by the EMF-IncQuery framework can (i) uniformly specify derived features and well-formedness constraints and (ii) automatically refresh their result set upon model changes. However, for complex domains, derived features and constraints can be formalized incorrectly resulting in incomplete, ambiguous or inconsistent DSL specifications. To detect such issues, we propose an automated mapping of EMF metamodels enriched with derived features and well-formedness constraints captured as graph queries in EMF-IncQuery into an effectively propositional fragment of first-order logic which can be efficiently analyzed by the Z3 SMT-solver. Moreover, overapproximations are proposed for complex query features (like transitive closure and recursive calls) Our approach will be illustrated on analyzing DSL being developed for the avionics domain.

**Keywords:** model validation, model queries, SMT-solvers

## 1 Introduction

The design of integrated development environments (IDEs) for complex domain-specific languages (DSL) is still a challenging task nowadays. Generative environments like the Eclipse Modeling Framework (EMF) [1], Xtext or the Graphical Modeling Framework (GMF) significantly improve productivity by automating the production of rich editor features (e.g. syntax highlighting, auto-completion, etc.) to enhance modeling for domain experts. Furthermore, there is efficient tool support for validating well-formedness constraints and design rules over large model instances of the DSL using tools like Eclipse OCL [2] or EMF-INCQUERY

---

<sup>\*</sup> This work was partially supported by the CERTIMOT (ERC\_HU-09-01-2010-0003), the TÁMOP (4.2.2.B-10/1-2010-0009) projects and the János Bolyai Scholarship.

[3]. As a result, Eclipse-based IDEs are widely used in the industry in various domains including business modeling, avionics or automotive.

However, in case of complex, standardized industrial domains (like ARINC 653 [4] for avionics or AUTOSAR [5] in automotive), the sheer complexity of the DSL is a major challenge itself. (1) First, there are hundreds of well-formedness constraints and design rules defined by those standards, and due to the lack of validation, there is no guarantee for their consistency or unambiguity. (2) Moreover, domain metamodels are frequently extended by derived features, which serve as automatically calculated shortcuts for accessing or navigating models in a more straightforward way. In many practical cases, these features are not defined by the underlying standards but introduced during the construction of the DSL environment for efficiency reasons. Anyhow, the specification of derived features can also be inconsistent, ambiguous or incomplete.

As model-driven tools are frequently used in critical systems design to detect conceptual flaws of the system model early in the development process to decrease verification and validation (V&V) costs, those tools should be validated with the same level of scrutiny as the underlying system tools as part of a software tool qualification process issues in order to provide trust in their output. Therefore software tool qualification raises several challenges for building trusted DSL tools in a specific domain.

In the current paper, we aim to validate DSL tools by proposing an automated mapping from their high-level specification to the state-of-the-art Z3 SMT-solver [6]. We assume that DSL tools are specified by their respective EMF metamodels extended with derived features and well-formedness constraints captured (and implemented) by graph queries within the EMF-INCQUERY framework [7,8]. We define a validation process, which gradually investigates derived features and well-formedness constraints to pinpoint inconsistency, ambiguity or incompleteness issues. We identify constraints and derived features which can be mapped to effectively propositional logic formula [9], which are a decidable fragment of first order logic with effective reasoning support. Moreover, we provide several approximations for constraints which lie outside of this fragment to enable formal analysis of a practically relevant set of constraints.

The main innovation of our approach is to provide a *combined validation* of metamodels, derived features and well-formedness constraints *defined by an advanced graph query language* (instead of OCL) using *approximations to cover complex query features*. Our approach is illustrated on validating several DSL tool features taken from an ongoing industrial project in the avionics domain.

The rest of the paper is structured as follows. Sec. 2 provides an overview of EMF metamodels enriched with derived features and well-formedness constraints captured by graph queries of the EMF-INCQUERY language in the scope of a DSL from the avionics domain. Sec. 3 describes a high-level overview of mapping DSLs to logic formula and validation scenarios from a domain expert's viewpoint. Details of the mapping are elaborated in Sec. 4, while Sec. 5 includes an initial evaluation of expressiveness and preliminary execution results. Related work is assessed in Sec. 6 while finally, Sec. 7 concludes our paper.

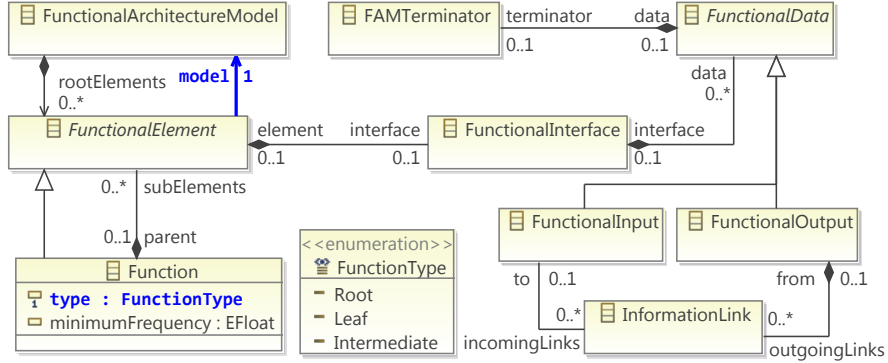


Fig. 1: Metamodel of the Functional Architecture

## 2 Preliminaries: Domain Modeling

To illustrate the proposed V&V technique, this paper elaborates a case study from DSL tool development for avionics systems. To create an advanced modeling environment, we augment the metamodel with *query-based derived features* and *well-formedness validation rules*. Both of these advanced features are defined using model queries. Within the paper, we use the language of the EMF-INQUERY [10] framework to define these queries over EMF metamodels.

### 2.1 Metamodel of the Case Study

In model-driven development of avionics systems, the *functional architecture* and the *platform description* of the system is often developed separately to increase reusability. The former defines the services performed by the system and links between functions to indicate dependencies and communication, while the latter describes platform-specific hardware and software components and their interactions. The functional architecture is usually partially imported from industry accepted tools and languages like AADL [11] or Matlab Simulink [12].

A simplified metamodel for functional architecture is shown in Fig. 1. The `FunctionalArchitectureModel` element represents the root of a model, which contains each `Function` (subtype of the `FunctionalElement`). Functions have a `minimumFrequency`, a `type` attribute and multiple `FunctionalInterfaces`, where each interface is either an `FunctionalOutput` (for invoking other functions) or an `FunctionalInput` (for accepting invocations). An output can be connected to an input through an `InformationLink`. Finally, if an input or output is not connected to an other `Function` then they must be terminated in a `FAMTerminator`.

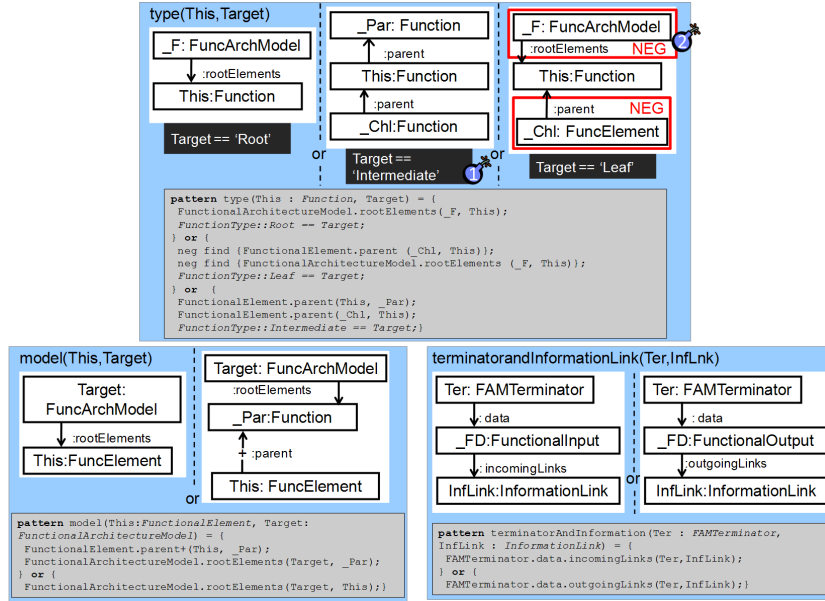


Fig. 2: The model and type DF and the terminatorandInformationLink WF constraint

## 2.2 Derived Features

Derived features (DF) are often essential extensions of metamodels to improve navigation, provide path compression or compute derived attributes. The value of these *features* can be computed from other parts of the model by a *model query* [7,13]. Such queries have two parameters, in case of (i) *derived EReferences* one parameter represents the source and another the target EObjects of the reference while in case of (ii) *derived EAttributes* one parameter represents the container EObject while the other one the computed value of its attribute.

FunctionalElements are augmented with the model derived EReference (highlighted in blue in Fig. 1) that represents a reference to the container FunctionalArchitectureModel EObject from any FunctionalElement within the containment hierarchy. Additionally, for the type EAttribute of the Function EObject a derived attribute is defined, which takes a value from the enumeration literals: Leaf, Root, Intermediate.

In Fig. 2 we use a custom graphical and the EMF-INCQUERY textual notation [3] to illustrate the queries defined for these derived features. On the graphical notation each rectangle is a named variable with a declared type, e.g. the variable `_Par` is a `Function`, while arrows represent references of the given EReference between the variables, e.g. the function `This` has the `_Par` function as its parent. A special reference between variables is the transitive closure depicted by an arrow with a `+` symbol, e.g., the `parent` reference between the `This`

and `_Par` variables in the model query. Finally, the `OR` pattern bodies represent that the matches of the query is the union of the matches of its or bodies.

For example, the `type` query (see in Fig. 2) has three `OR` pattern bodies each defining the value for the corresponding enum literal of the `type` attribute: (i) `Leaf` if the container `EObject` does not have a `child` function along the `subFunctions` `EReference` and it is not under the `FunctionalArchitectureModel` along the `rootElements` `EReference`, where both of these constraints are defined using negative application conditions (`NEG`), (ii) `Root` if container `EObject` is directly under the `FunctionalArchitectureModel` connected by the `rootElements` `EReference` or (iii) `Intermediate` if container `EObject` has both `parent` and `child` functions.

*Validation challenges:* We aim to validate the following properties for DFs:

- **Consistency** means that there is at least one valid instance model containing an object that has a target object or attribute value for the DF.
- **Completeness** means that in each valid instance model the derived feature is evaluated with at least one result (target object or attribute value).
- Finally, **unambiguity** means that in each valid instance model, DF can only be evaluated to a single result (target object or attribute value).

### 2.3 Well-Formedness Constraints

We also define some structural well-formedness (WF) constraints (usually derived from design rules and guidelines) to be validated on functional architecture models. In our current approach WF constraints define ill-formed model structures and thus they cannot have a match in a valid model. In our running example, a design rule captures that a `FunctionalData` `EObject` with a `FAMterminator` cannot also be connected to an `InformationLink`. It is specified by the `terminatorandInformationLink` query (see in Fig. 2) that has two `OR` pattern bodies, one for the `FunctionalInputs` and one for the `FunctionalOutputs` with their corresponding `incomingLinks` and `outgoingLinks`, respectively.

The aim of our case study is to demonstrate that its derived features and well-formedness constraints can be effectively validated using our mapping method (see in Sec. 4) to the Z3 SMT solver.

*Validation challenges:*

- **Consistency** of WFs can only be interpreted over the complete DSL specification, which in our understanding means that there is at least one valid instance model that satisfies all constraints.
- The **subsumption** property of a DSL is defined over its set of well-formedness constraints. If a WF constraint is subsumed by the set, then such a WF constraint does not express any additional restriction over the DSL. Therefore, it can be removed without changing the set of the valid instance models.

## 3 Overview of the Approach

Our approach (illustrated in Fig. 3) aims at validating complex DSL languages by automatically mapping from their high-level specification to the Z3 [6] SMT-solver. These complex DSLs are assumed to be defined by (i) a metamodel

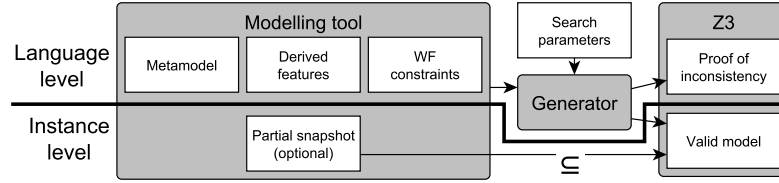


Fig. 3: Overview of DSL validation: Inputs and outputs

specified in EMF and augmented with both (ii) **derived features** and (iii) **well-formedness (WF) constraints** captured by model queries within the EMF-INCQUERY framework. These three artifacts form the input for our generator to provide the logical formulas that is fed into the Z3 solver. The output of the solver is either a **proof of inconsistency** or a **valid model** that satisfies all given constraints generated from the input artifacts.

Additionally, **search parameters** can be defined to impose additional restrictions or specific overapproximations to reduce the complexity of the formula to be proved. Moreover, as an optional input for the generator the user can define – based on the counter examples and proves provided by the solver – specific instance level constraints in the form of an **partial snapshot** [14,15] (also called input model) to restrict the domain of possible instance model and thus prune trivial valid models (e.g., empty model) provided by the Z3 solver.

*End User Validation Workflow* Our iterative validation workflow for complex DSLs (see Fig. 4) assumes the existence of the metamodel (captured in EMF), its derived features and well-formedness constraints (captured as graph queries).

First, each DF is investigated by adding them to the formal DSL specification (extending it with one new DF in a predefined order), and then by validating this specification in Z3. Then, WF constraints are validated similarly, by augmenting the specification with a single WF constraint at each validation step.

The validation fails, if the compiled set of formulas are inconsistent (formally, no models can be constructed). In such a case, the designer needs to either (i) fine-tune the search parameters, (ii) provide a new partial snapshot or (iii) modify the DSL specification itself based on the proof outcome. If the formal DSL specification with all DF and WF constraints is validated, then it is valid under the assumptions imposed by the search parameters and the partial snapshot.

The separation to start the iterative validation process with the derived features and then continue with the well-formedness constraints is based on the assumption that each derived feature eliminates a large set of trivial, non-conforming instance models (that are not valid instances of the DSL). This eases the refinement in case of an erroneous DF or WF is added in the actual step based on the proof provided by the solver.

*Example DSL validation scenario* To illustrate the execution of our validation workflow Fig. 5 shows a possible validation scenario for our running example.

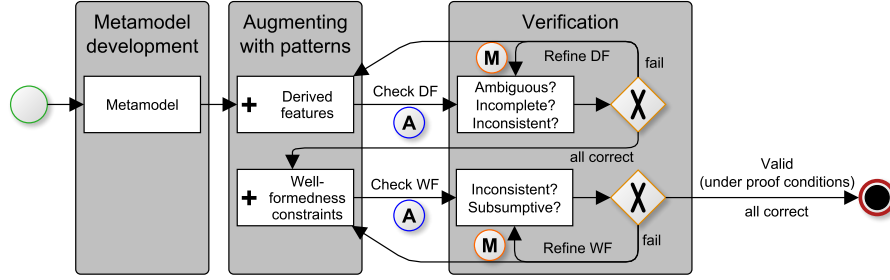


Fig. 4: End user workflow of validation of DSLs

As the input for the validation scenario we use the metamodel, DFs and WF constraints as defined in Sec. 2 with three modifications (to add hypothetical conceptual flaws in queries):

1. the second pattern body (marked as bomb 1 in Fig. 2) is missing from the DF query `type`, which defines `Function` elements of `Intermediate` type,
2. the third body of query `type` specifying the `Leaf` type is also changed: it forgets to define a NAC condition over the `rootElements` EReference (bomb2).
3. one WF constraint is added to the DSL specification expressed by the IL2T query, which prohibits that a `InformationLink` is connected to a `FAMTerminator`. This constraint only differs from the first body of the original WF constraint that it uses the inverse edges and thus it is a redundant.

Sec. 2 describes how the DFs and WF constraints are added to the formal DSL specification (after the metamodel is already added) and validated. To ease understanding the counter examples (`CE_i`) or partial snapshots (`PS`) produced or defined during the validation are depicted on the right side. Each row describes Fig. 5 which *validation step* was executed in the actual iteration, what was its *outcome* and what *action* has been taken to refine its validation.

First (Step 1) we add the `type` DF to the formal specification and validate its consistency by setting the default overapproximation for the transitive acyclicity constraint (see in Section 4.1) to a maximum of 2 levels. Then (Step 2), the completeness of the `type` DF is checked resulting in a failure due to the produced `CE1` counter example that is a function without its `Intermediate` type. This is fixed by adding the second pattern body with the `Intermediate` definition to the `type` pattern. By correcting it, the validation is successfully executed. After this the ambiguity of the attribute is tested (Step 3) and failed again (with a singular function node that is both a `Leaf` and a `Root`) that is fixed by adding the missing NAC condition on the `rootElements` to the third pattern body of `type`. The next step (Step 4) adds the `model` DF and followed in Step 5 with its completeness validation, which fails due to `CE3` that does not have a `model` EReference. A partial snapshot is defined with a `FunctionalArchitectureModel` object to prune the search space and avoid such counter examples, however, its revalidation (Step

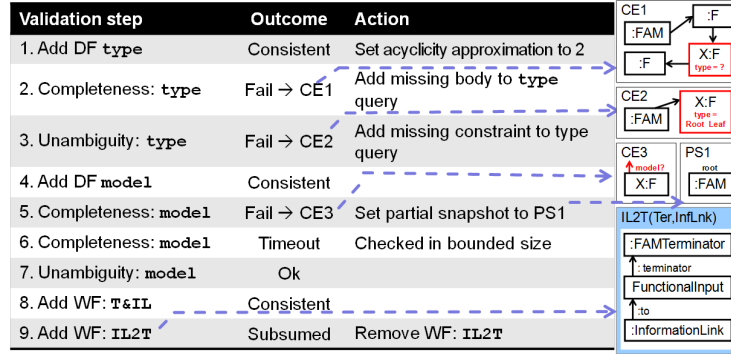


Fig. 5: Example DSL validation scenario

6) ends in a `Timeout` (more than 2 minutes) and thus this feature can only be validated on a concrete bounded domain of maximum 5 model objects. In Step 7 the unambiguity of the `model` DF is validated without a problem. Followed by the consistency validation of the `terminatorandInformationLink` WF constraint (Step 8). Finally, the `IL2T` WF constraint is checked for subsumption (Step 9) and found positive; thus it is already expressed by the DSL specification and can be deleted from the WF constraints.

## 4 Mapping DSLs to FOL Formulae

In this section, we demonstrate how Ecore metamodels augmented with derived features and well-formedness constraints captured as model queries (namely, EMF-INCQUERY graph patterns) are mapped to first order logical (FOL) formulae. Our idea is to map all DSL concepts to the effectively propositional fragment (EPR) [9] of FOL to guarantee decidability and efficient validation (that can be automatically proved using the Z3 [6] solver). It corresponds to formulae written in prenex normal form, which contain only constants, universal quantifiers, and functions that return boolean values (aka predicates). If it is failed, the specification is handled as a general FOL problem or can be approximated by EPR statements. Some feature like transitive closure is still inexpressible in FOL, these constraints are approximated.

**Mapping structure for DSL** In order to represent a DSL specification in FOL we use the following structure:  $DSL = META \wedge DFs \wedge WFs$ , where  $META$  represents the FOL statement set defined by the metamodel (e.g., type hierarchy),  $DFs$  symbolizes the statement set specified by the derived features and finally,  $WFs$  represents the set of statements for the well-formedness constraints.

A FOL statement may be handled using under- or overapproximations, where statements  $C^U$  or  $C^O$  under- or overapproximate statement  $C$  if they satisfy



that  $C^U \Rightarrow C$  or  $C \Rightarrow C^O$ , respectively. As a trivial example, the *true* constant overapproximates all statements and can substitute any DSL constraint.

This definition can be extended to statement sets, where a statement set  $CS$  is over- or underapproximated by a statement set  $CS^O$  (or  $CS^U$ ), if each statement  $C \in CS$  is over- or underapproximated by a statement  $C^O \in CS^O$  (or  $C^U \in CS^U$ ). This allows to validate properties of the *DSL* by proving the same properties on its under- or overapproximations. The construction of *META*, *DFs* and *WFs* and their corresponding approximations are defined and illustrated in the following sections.

#### 4.1 Mapping of the Ecore Model

The different features of the target domain specific language described as an Ecore metamodel are mapped to FOL formulae in the following way. Each generated statement is added to the *META* set.

**Type hierarchy** The elements of the output instance model are uniformly mapped to a **Z3** type *object* declared by the compiler. Type indicator predicates are used to describe that an *object* is an instance of an EClassifier. Additionally, to interpret supertype relations a *d* disjunction of  $c_i$  conjunctions of type predicates are constructed, formally  $d = c_1 \vee c_2 \vee \dots \vee c_n$ . For each non-abstract type in the metamodel one  $c_i = type_1 \wedge type_2 \wedge \dots \wedge type_m$  is constructed where only those type predicates  $type_i$  are positive literals that are either its direct type or one of its supertypes (e.g., for type **Function** the **FunctionElement** is also a positive literal and all other type predicates are negative literals). This way only one  $c_i$  can be true in *d* for any objects that conforms to the metamodel.

For example, the **Function** class in Table 1 is mapped to a formula where only the **Function(f)** and the **FunctionalElement(f)** predicates are positive literals.

**References and attributes** An *EReference* between two EObjects is a directed relation represented as *reference predicates* (boolean functions) and its target type is explicitly asserted to restrict their range to the specific EObject types. E.g. the FOL formula generated for **parent** EReference in Table 1 ensures that the source object **e** is a **FunctionalElement** while the target end **f** is a **Function**.

An *inverse reference* in Ecore is mapped to two separate predicates defined in the opposite direction and an additional equivalence operation is defined between them to assert their inverse nature. For instance, **parent(e,f)** is defined to be an inverse of **subElements(f,e)** in Table 1.

The objects of an EMF model are arranged into a directed tree hierarchy along the *containment* EReferences, which is mapped into two constraints: (i) The acyclicity constraint defined as a transitive closure stating that any object is unreachable from itself is defined similarly to the example in Section 4.2 and (ii) the singular root constraints is expressed as a statement that there is exactly one object in the model without a parent.

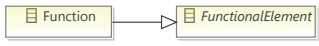
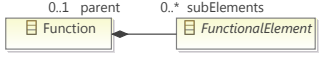
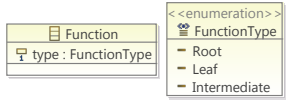
EObject	$\mapsto$ Object
<p><b>f : Function</b></p> 	<p><b>Type:</b> <math>Function(f)</math>  <math>c_{function}: Function(f) \wedge FunctionalElement(f)</math>  <math>\wedge \neg FAMTerminator(f) \wedge \neg InformationLink(f)</math>  <math>\wedge \dots \wedge \neg FunctionalInterface(f)</math></p>
<p><b>e.getParent == f</b></p> 	<p><b>Reference predicate</b> <math>Parent(e, f)</math>  <b>End types:</b> <math>\forall e, f : Parent(e, f) \Rightarrow (FunctionalElement(e) \wedge Function(f))</math>  <math>\mapsto</math> <b>Inverse edges:</b> <math>\forall e, f : Parent(e, f) \Leftrightarrow subElements(f, e)</math>  <b>At most one multiplicity:</b> <math>\forall e, f_1, f_{extra} : Parent(e, f_1) \wedge Parent(e, f_{extra}) \Rightarrow f_1 = f_{extra}</math></p>
<p><b>f.type : FunctionalType</b></p> 	<p><b>Attribute type:</b>  <math>Type = \{Root, Intermediate, Leaf\}</math>  <math>\mapsto</math> <b>Attribute value:</b>  <math>type(f, Leaf)</math></p>

Table 1: Examples of mapping the features of an Ecore metamodel.

Our approach currently supports *EAttributes* with enumeration types, where the enum literals are mapped to constants and the EAttribute is represented as a predicate with the source as the container object and the target as the value of corresponding constant. For example, the `type(f, Leaf)` defines that the value of the type EAttribute is a `Leaf` in Table 1. We plan to investigate [16] to extend to other types.

By default, the predicates model the Ecore links with the most general  $0..*$  *multiplicity*. An upper bound can be mapped to an EPR by assuring there that the number of different target objects are less than it defines, however, a lower bound cannot be expressed without existential quantifiers and thus leads out of EPR. Without going into details an example mapping of cardinality constraints are demonstrated in Row 2 of Table 1.

## 4.2 Mapping of the Graph Queries

In the current section we highlight how different features of the EMF-INCCQUERY graph query language are mapped to FOL formulae.

**Structure of a Query** On the top level, a graph query consists of: (i) a parameter list, which is a fix sized vector of variables over the objects of the instance model and (ii) one or more disjunctive (*OR*) pattern bodies, which define constraints over its parameters and additional existentially quantified internal variables. A query in our mapping is defined as a disjunction of its bodies, where the bodies are the conjunction of its constraints.

In Table 2, the mapping of the query of the `type` DF is exemplified, where the first row defines how its pattern bodies are mapped to three separate `body_i`

predicates. The second row demonstrates how the second body of the query is mapped to a FOL formula, where the `_Par` and `_Chl` are its inner variables. Simple constraints in a pattern body are handled as follows:

- *Attribute check* conditions are mapped to their corresponding equivalent in FOL as over enum literals only the equivalence and non-equivalence relation are defined. For instance, in Table 2 the third row defines an equivalence relation between the `Target` variable and the `Intermediate` constant.
- An *EClassifier* constraint defines the type of the object that is bound to a variable, which is simply mapped to its corresponding type predicate. E.g, in the fourth row in Table 2 the `This` variable can only be of type `Function`.
- *EReference* constraints are compiled into their corresponding reference predicates, for example, in Table 2 the `parent` EReference is mapped to the its corresponding `parent(_Chl, This)` reference predicate.
- Finally, a *negative application condition* is mapped to (i) a subpattern definition – for the `neg` pattern – identically to how a pattern body is constructed and a (ii) pattern call (PL) constraint. The PL constraint forbids the satisfaction of the subpattern with the parameter substitution specified by its defining pattern body. For instance, in Table 2 the `nacSubPattern(Child, Parent)` is constructed and it is called using the `¬nacSubPattern(_Chl, This)` formula with the `_Chl` and `This` variables as defined by in the third pattern body of the `type` query.

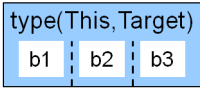
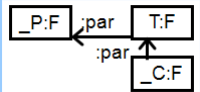
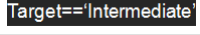

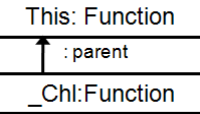
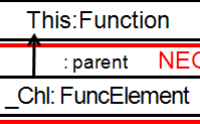
	<b>DF predicate:</b> $typeDF(This, Target)$ <b>Or queries:</b> $\forall Type, Target : typeDF(This, Target) \Leftrightarrow body_1(This, Target) \vee body_2(This, Target) \vee body_3(This, Target)$
	<b>Pattern body:</b> $body(This, Target) = \exists \_Par, \_Chl : \rightarrow Function(\_Par) \wedge Function(\_Chl) \wedge Function(This) \wedge parent(\_Chl, This) \wedge parent(This, \_Par)$
	$\rightarrow$ <b>Attribute Condition:</b> $Target = Intermediate$
	$\rightarrow$ <b>EClassifier constraint:</b> $Function(This)$
	$\rightarrow$ <b>EReference constraint:</b> $parent(\_CHL, This)$
	<b>Negative Application Condition:</b> <b>Subpattern:</b> $nacSubpattern(Chlid, Parent) \Leftrightarrow parent(Chlid, Parent)$ <b>Pattern call constraint:</b> $\neg nacSubPattern(\_Chl, This)$

Table 2: Mapping of graph query features.

**Constraint Approximation** The EMF-INCQUERY graph query language is more expressive than the EPR fragment of FOL thus some constraints (like recursively called patterns, transitive closures) cannot be expressed within its boundaries. Below, we sketch the overapproximation of the transitive closure feature of the query language, while more technical details are available in [17].

We use an overapproximation on the maximum iteration of the traversal on the transitive reference. The idea is to define unique transitive predicates for each iteration that defines how many more references it can traverse along the transitive reference, where finally the ending predicate is substituted with the *true* predicate (overapproximation). Additionally, to force acyclic traversal the predicates also specify uniqueness constraints over the visited objects.

For example, predicate  $\text{parentMatch}(This, P) \Rightarrow \text{parent2Match}(This, P)$  defines an overapproximation of length 2 for the transitive closure of the `parent` EReference in the second pattern body of the `model` query, in the following way:

- 2:**  $\text{parent2Match}(This, P) \Rightarrow \text{parent}(This, P) \vee \exists m1 : \text{parent}(This, m1) \wedge \text{parent1Match}(m1, P, This)$
- 1:**  $\text{parent1Match}(This, P, d1) \Rightarrow \text{parent}(This, P) \vee \exists m2(m2 \neq d1) : \text{parent}(This, m2) \wedge \text{parent0Match}(m2, P, d1, This)$
- 0:**  $\text{parent0Match}(This, P, d1, d2) \Rightarrow \text{parent}(This, P) \vee \exists m3(m3 \neq d1, m3 \neq d2) : \text{parent}(This, m3) \wedge \text{true}$

Note that a similar idea is used in case of recursive pattern calls, where after flattening the call hierarchy only the recursive calls are needed to be overapproximated based on the number of maximum allowed calls.

**Patterns as DF and WF** When constructing the set of axioms for a DSL from graph patterns, derived features and well-formedness constraints need to be handled differently. In case of DFs, we need to guarantee that the evaluation of the predicate of the derived feature and its graph query definition is equivalent. For example, in case of `type` of a `Function` where *type* is the attribute predicate and *typeDF* is a pattern it looks like this:  $\forall src, trg : \text{type}(src, trg) \Leftrightarrow \text{typeDF}(src, trg)$ . This statement is added to the statement set *DFs*.

When a pattern defines a WF constraint, by definition, it is not allowed to have any match in a valid model, thus the axiom needs to be quantified accordingly. For instance, for `patternTerminatorandInformationLink` we add to the *WF* set:  $\forall Ter, InfLink : \neg \text{terminatorandInformationLink}(Ter, InfLink)$ .

### 4.3 Search parameters

The verification can be parameterised by different optional search parameters:

- **Target partition:** In order to reduce the state space of the verification it can be defined to map only a part of the metamodel relevant to the verification.
- **Partial snapshot:** The verification may fail on trivial counterexamples that are theoretically correct but do not corresponds to the expected structure. The range of the checked models can be limited to the extension of an initial instance model, which constitutes the constants of the input and the assumptions partially define the truth-value of the predicates.

- **Maximum size of verified instance model:** Following the small scope hypothesis [18], the maximum size of the instance models to be checked during the validation can be optionally defined. This allows to solve the validation as a SAT problem or provide a minimal counter example.
- **Approximation level:** Whenever an over- and underapproximation is used to describe a certain metamodel, DF or WF feature it is required to explicitly define the boundaries (or level) of the approximation.

## 5 Evaluation

The aim of our evaluation is to illustrate that our mapping approach is capable of expressing and validating complex metamodel and query features either by directly mapping them to EPR (denoted as +), solve them as a general FOL proving problem (–) or approximate the general problem by relaxing it to an ERP (e.g. overapproximate the containment hierarchy by neglecting it). The Table 3 summarizes all relevant features of both the Ecore metamodels and the EMF-INCQUERY model query languages that can (+) or cannot (–) be mapped directly to EPR, needs approximation (A) to define it in FOL or is inexpressible (X) in FOL. As the DF and WF constraints in overall are validated using different polarity the quantifications of their variables will differ and thus; they cannot be mapped the same way (e.g., the same query may not be validated as DF or WF over the same properties). Detailed discussion about the mapping of these features is available in [17].

The runtime performance of our approach is negligible in cases when the mapping can be kept in EPR and then is usually under 1 sec for example, in our scenario it was less than 100 ms for all feature validation except for the completeness validation of the `model` DF (timeout). However, whenever the mapped features are outside of EPR the outcome of the validation relies on the underlying automated theorem prover, which may be able to validate the feature but there are no guarantees that it will ever produce a proof or refutation due to undecidability of FOL in general.

## 6 Related work

There are several approaches and tools aiming to validate UML models enriched with OCL constraints [19] relying upon different logic formalisms such as constraint logic programming [20,21,16], SAT-based model finders (like Alloy) [22,23,24,25], first-order logic [26,27], constructive query containment [28], higher-order logic [29,30], or rewriting logics [31]. Some of these approaches (like e.g. [21,23,24]) offer bounded validation (where the user needs to explicitly restrict the search space), others (like [27,29,26]) allows unbounded verification (which normally results in increased level of user interaction and decidability issues).

SMT-solvers have also been used to verify declarative ATL transformations [32] allowing the use of an efficiently analyzable fragment of OCL [27]. The

Features of the metamodel		DF	Features of model query	WF
EClasses	E +	E +	Classifier constraint	E +
Class hierarchy	E +	E -	EReference constraint	E +
EEnums	E +	E -	Acyclic pattern call	E +
EReferences	E +	E -	Negative pattern call	E -
EAttributes	E +	A -	Transitive closure	A +
Multiplicity upper bound	E +	A -	(Positive) pattern call recursion	A +
Multiplicity lower bound	E -	A -	Arbitrary call graph	A -
Inverse edges	E +	X	Aggregate (eg. Count, Sum)	X
Containment hierarchy	A -	X	Check expressions	X
Partial snapshot	E +			

E: Expressible A: Approximable X: Inexpressible +: in EPR -: not in EPR  
Table 3: Expressing Ecore and EMF-INCQUERY language features in Z3

FORMULA tool also uses the Z3 SMT-solver as underlying engine, e.g. to reason about metamodeling frameworks [15] where proof goals are encoded as CLP satisfiability problem. The main advantage of using SMT solvers is that it is refutationally complete for quantified formulas of uninterpreted and almost uninterpreted functions and efficiently solvable for a rich subset of logic. Our approach uses SMT-solvers both in a constructive way to find counter examples (model finding) as well as for proving theorems. In case of using approximations for rich query features, our approach converges to bounded verification techniques.

Graph constraints are used in [33] as means to formalize a restricted class OCL constraints in order to find valid model instances by graph grammars. An inverse approach is taken in [34] to formalize graph transformation rules by OCL constraints as an intermediate language and carry out verification of transformations in UML-to-CSP tool. These approaches mainly focus on mapping core graph transformation semantics, but does not cover many rich query features of the EMF-IncQuery language (such as transitive closure and recursive pattern calls). Many ideas are shared with approaches aiming to verify model transformations [34,35,32], as they built upon the semantics of source and target languages to prove or refute properties of the model transformation.

The idea of using *partial models*, which are extended to valid models during verification also appears in [14,15,36]. These initial hints are provided manually to the verification process, while in our approach, these models are assembled from a previous (failed) verification run in an iterative way (and not fully manually). *Approximations* are used in [37] to propose a type system and type inference algorithm for assigning semantic types to constraint variables to detect specification errors in declarative languages with constraints.

Our approach is different from existing approaches as it uses a graph based query language instead of OCL for capturing derived features and well-formedness constraints. Up to our best knowledge, this is the first approach aiming to validate queries captured within the EMF-IncQuery framework, and the handling of derived features is rarely considered. Furthermore, we sketch an iterative val-

validation process how DSL specifications can be carried out. Finally, we also cover the validation of rich language features (such as recursive patterns or transitive closure) which is not covered by existing (OCL-based) approaches.

## 7 Conclusion

In the paper, we addressed the validation of DSL tools specified by a combination of EMF metamodels and graph queries (of EMF-INCQUERY) capturing derived features and well-formedness rules. For that purpose, we defined an iterative (and semi-automated) validation workflow and a mapping of metamodels and queries to the effectively propositional (EPR) fragment of first-order logic, which can be efficiently analyzed by the Z3 SMT solver. In order to cover rich language features (such as transitive closure and recursion), we proposed constraint approximations to yield formulae that fall into EPR. Moreover, validation can be guided by the designer in the form of initial (partial) model snapshots, which need to be included in valid instance models. We illustrated our approach on a running example extracted from an ongoing research project in the avionics domain.

Our future work is intended to be directed to improve the level of query feature coverage and raise the level of automation of our system. For instance, our current approach is restricted to handle attributes of enumeration values only, while real metamodels contain attributes of integers, strings, etc. For this purpose, we may build upon [16] where reasoning is provided for string attributes in OCL constraints or other decision procedures for numeric domains.

In our current framework, automation is restricted to forward mappings, while refinements are carried out manually by the domain engineer. It would be advantageous to shift our framework towards a black-box solution as much as possible, which immediately raises several challenges. On the tooling level, counter-examples derived by Z3 should be back-annotated to the DSL tooling (as model instances). On the validation level, an interesting direction is to develop counterexample guided refinement of approximations where false positive counterexamples obtained as a result of approximations can be filtered by instance-level validation techniques.

## References

1. The Eclipse Project: Eclipse Modeling Framework. <http://www.eclipse.org/emf>.
2. Willink, E.D.: An extensible OCL virtual machine and code generator. In: Proc. of the 12th Workshop on OCL and Textual Modelling, ACM (2012) 13–18
3. Bergmann, G., Horváth, Á., Ráth, I., Varró, D., Balogh, A., Balogh, Z., Ökrös, A.: Incremental Evaluation of Model Queries over EMF Models. In: MODELS'10. Volume 6395 of LNCS., Springer (2010)
4. ARINC - Aeronautical Radio, Incorporated: A653 - Avionics Application Software Standard Interface
5. AUTOSAR Consortium: The AUTOSAR Standard. <http://www.autosar.org/>.
6. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems. TACAS'08/ETAPS'08, Springer-Verlag (2008) 337–340
7. Ráth, I., Hegedüs, A., Varró, D.: Derived features for EMF by integrating advanced model queries. In Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D., eds.: Modelling Foundations and Applications. Volume 7349 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) 102–117
8. Hegedüs, A., Horváth, A., Ráth, I., Varró, D.: Query-driven soft interconnection of EMF models. In: Proc of the Int. Conf on Model Driven Engineering Languages and Systems. Volume LNCS 7590. (2012) 134–150
9. Piskac, R., de Moura, L., Bjorner, N.: Deciding effectively propositional logic with equality (2008) Microsoft Research, MSR-TR-2008-181 Technical Report.
10. Bergmann, G., Ujhelyi, Z., Ráth, I., Varró, D.: A graph query language for emf models. In Cabot, J., Visser, E., eds.: Fourth International Conference on Theory and Practice of Model Transformations. Volume 6707 of LNCS., Springer (June 2011) 167–182
11. SAE - Radio Technical Commission for Aeronautic: Architecture Analysis & Design Language (AADL) v2, AS-5506A, SAE International, 2009
12. Mathworks: Matlab Simulink - Simulation and Model-Based Design. <http://www.mathworks.com/products/simulink/>.
13. The Object Management Group: Object Constraint Language, v2.0. (May 2006) <http://www.omg.org/spec/OCL/2.0/>.
14. Sen, S., Mottu, J.M., Tisi, M., Cabot, J.: Using models of partial knowledge to test model transformations. In: 5th Int. Conf. on Theory and Practice of Model Transformations. Volume 7307 of LNCS. (2012) 24–39
15. Jackson, E.K., Levendovszky, T., Balasubramanian, D.: Reasoning about meta-modeling with formal specifications and automatic proofs. In: Proc. of the 14th Int. Conf. on Model Driven Engineering Languages and Systems. Volume 6981 of LNCS. (2011) 653–667
16. Büttner, F., Cabot, J.: Lightweight string reasoning for OCL. In Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D.S., eds.: Modelling Foundations and Applications - 8th European Conference, ECMFA 2012, Lyngby, Denmark, July 2-5, 2012. Proceedings. Volume 7349 of LNCS., Springer (2012) 244–258
17. Oszkár Semeráth: Validation of Domain Specific Languages (2013) Technical Report, <https://incquery.net/publications/dslvalid>.
18. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)



19. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL models in USE by automatic snapshot generation. *Softw. Syst. Model.* **4**(4) (2005) 386–398
20. Cabot, J., Clarisó, R., Riera, D.: UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In: *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, New York, NY, USA, ACM (2007) 547–548
21. Cabot, J., Clarisó, R., Riera, D.: First international conference on software testing verification and validation. In: *Verification of UML/OCL Class Diagrams using Constraint Programming*, IEEE (2008) 73–80
22. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On challenges of model transformation from UML to Alloy. *Softw. Syst. Model.* **9**(1) (2010) 69–86
23. Büttner, F., Egea, M., Cabot, J., Gogolla, M.: Verification of ATL transformations using transformation models and model finders. In: *14th International Conference on Formal Engineering Methods, ICFEM'12, LNCS 7635*, Springer (2012) 198–213
24. Kuhlmann, M., Hamann, L., Gogolla, M.: Extensive validation of OCL models by integrating SAT solving into use. In: *TOOLS'11 - Objects, Models, Components and Patterns*. Volume 6705 of LNCS. (2011) 290–306
25. Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., Drechsler, R.: Verifying UML/OCL models using boolean satisfiability. In: *Design, Automation and Test in Europe, (DATE'10)*, IEEE (2010) 1341–1344
26. Beckert, B., Keller, U., Schmitt, P.H.: Translating the Object Constraint Language into first-order predicate logic. In: *Proc of the VERIFY, Workshop at Federated Logic Conferences (FLoC)*, Copenhagen, Denmark. (2002)
27. Clavel, M., Egea, M., de Dios, M.A.G.: Checking unsatisfiability for OCL constraints. *ECEASST* **24** (2009)
28. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. *Data Knowl. Eng.* **73** (2012) 1–22
29. Brucker, A.D., Wolff, B.: The HOL-OCL tool (2007) <http://www.brucker.ch/>.
30. Grönniger, H., Ringert, J.O., Rumpe, B.: System model-based definition of modeling language semantics. In: *Formal Techniques for Distributed Systems*. Volume 5522 of LNCS., Springer (2009) 152–166
31. Clavel, M., Egea, M.: The ITP/OCL tool (2008) <http://maude.sip.ucm.es/itp/ocl/>.
32. Büttner, F., Egea, M., Cabot, J.: On verifying ATL transformations using 'off-the-shelf' SMT solvers. In: *Proc. of the 15th Int. Conf. on Model Driven Engineering Languages and Systems*. Volume 7590 of LNCS. (2012)
33. Winkelmann, J., Taentzer, G., Ehrig, K., Küster, J.M.: Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars. *ENTCS* **211**(0) (2008) 159 – 170 *Proc. of the 5th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'06)*.
34. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: A UML/OCL framework for the analysis of graph transformation rules. *Softw. Syst. Model.* **9**(3) (2010) 335–357
35. Lucio, L., Barroca, B., Amaral, V.: A technique for automatic validation of model transformations. In: *Proc. of the 13th Int. Conf. on Model Driven Engineering Languages and Systems*. Volume 6394 of LNCS. (2010) 136–150
36. Kuhlmann, M., Gogolla, M.: Strengthening SAT-based validation of UML/OCL models by representing collections as relations. In: *European Conf. on Modelling Foundations and Applications*. Volume 7349 of LNCS. (2012) 32–48
37. Jackson, E.K., Schulte, W., Bjørner, N.: Detecting specification errors in declarative languages with constraints. In: *Proc. of the 15th Int. Conf. on Model Driven Engineering Languages and Systems*. Volume 7590 of LNCS. (2012) 399–414