Towards a Model Transformation Intent Catalog

Moussa Amrani
University of Luxembourg
Luxembourg
Moussa.Amrani@uni.lu

Jürgen Dingel
Queen's University
Kingston ON, Canada
Dingel@cs.queensu.ca

Leen Lambers
Hasso Plattner Institute
Postdam, Germany
Leen.Lambers@hpi.uni-potsdam.de

Levi Lúcio
McGill University
Montreal QC, Canada
levi@cs.mcgill.ca

Rick Salay
University of Toronto
Toronto ON, Canada
rsalay@cs.toronto.edu

Gehan Selim
Queen's University
Kingston ON, Canada
Gehan@cs.queensu.ca

Eugene Syriani
University of Alabama
Tuscaloosa AL, USA
esyriani@cs.ua.edu

Manuel Wimmer
University of Malaga
Spain
mw@lcc.uma.es

ABSTRACT

We report on our ongoing effort to build a catalog of model transformation intents that describes common uses of model transformations in Model-Driven Engineering (MDE) and the properties they must or may possess. We present a preliminary list of intents and common properties. One intent (transformation for analysis) is described in more detail and the description is used to identify transformations with the same intent in a case study on the use of MDE techniques for the development of control software for a power window.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Design Tools and Techniques; Design Methodologies

Keywords

Model-Driven Engineering, Model Transformations, Classification

1. INTRODUCTION

While most model transformation languages are Turing-complete and, as such, can be used to solve any computable problem, most were developed to support Model-Driven Engineering (MDE). We identify a limited set of *model transformation intents* that appear repeatedly in most MDE efforts. Awareness of these intents is useful for developers of model transformations and model transformation languages. For instance, the intent of a model transformation can be to extract different views from a model (query), add or remove detail (refinement or abstraction), translate the model to another modeling language (translation), execute the model (simulation), restructure the model to improve certain quality attributes (refactoring), compose models (composition), or reconcile the information in different models (synchronization).

Each of these intents has its own set of attributes and properties. The effectiveness of a transformation in realising an intent depends on how well it respects the intent's attributes and properties. For instance, queries should produce information contained in the model in some form, translations and refactorings should preserve model semantics, and refinements should add information.

Influenced by the survey conducted in [1], this paper reports on our ongoing effort to build a *model transformation intent catalog* that identifies and describes transformations intents and the properties they may or must possess. This catalog has several potential uses:

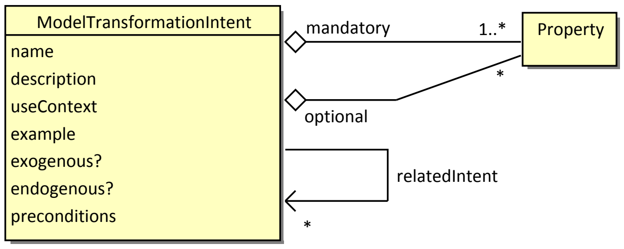
1. *Requirements analysis for transformations:* The catalog facilitates the description of transformation requirements, i.e., of what a transformation is supposed to do. Improved requirements can improve reuse, because they may make it easier to locate suitable transformations among a set of existing ones and reuse them.

2. *Identification of properties, certification methods, and languages:* The catalog may help transformation developers become aware of properties a transformation must possess, how these properties can be certified, and which transformation language is known to best support their needs (*i.e.,* if the used certification methods are language dependent).

3. *Model transformation language design:* The catalog may provide some useful input for designers of domain-specific transformation languages. For instance, it may be appropriate to design dedicated languages for specific intents for efficiency or readability. The properties and certification methods associated with an intent may provide useful information about requirements of a transformation language used for an intent.

Due to the space limitation, a subset of our current intent catalog and the supporting references is demonstrated in this paper. Besides illustrating our catalog and its uses

Figure 1: Portion of the intent domain metamodel with key classes `ModelTransformationIntent` and `Property`.



with different transformation examples from the literature, we use a case study on the use of transformations for the development of a car's power window software. It shows how important transformations are for the MDE of embedded software and how diverse their intents can be. In future work, we plan to complete the catalog and use it to classify and compare model transformation analysis approaches by extending the work in [1].

The paper is structured as follows: Section 2 presents a schema to describe an intent catalog, identifies common transformation intents, and lists some of their common properties; Section 3 instantiates this schema on a particular intent, namely the *translation for analysis*; Section 4 illustrates this particular intent with some of the matching transformations in the power window case study; Section 5 presents related work; and finally Section 6 concludes.

2. A MODEL TRANSFORMATION INTENT CATALOG

This section proposes a schema for a model transformation intent catalog: each intent has a set of attributes and properties, and a transformation with this intent should demonstrate such properties to be able to achieve its underlying goal. We then list a set of common transformation intents and model transformation properties from the literature.

2.1 Transformation Intent Catalog Scheme

In order to describe model transformations intents in a systematic way, we developed a schema for model transformations intents and their properties, which forms the basis of our intent catalog. Figure 1 shows a fragment of this schema as a metamodel: its root class is `ModelTransformationIntent`, whose attributes as described in Table 1. Adapted from the Gang Of Four [2], we define a transformation intent as follows:

Definition 1. *A* model transformation intent *is a description of the goal behind the model transformation and the reason for using it.*

The class `Property` encapsulates the properties of transformations that fall under a specific intent. We restrict our definition of a model transformation property as follows:

Definition 2. *A* model transformation property *is any verifiable characteristic inherent to a model transformation that depends on the internal details of its input/output types (i.e., definition of syntax and semantics) or the internal details of its implementation (i.e., the "code" of the transformation).*

Attributes	
`name`	Name used to identify the intent
`description`	Informal description of the intent's goal
`useContext`	Description of when to use a transformation with this intent (i.e. what problems can be solved?)
`example`	Examples of transformations with this intent
`exogeneous?`	True iff this intent can be expressed with an exogeneous transformation
`endogeneous?`	True iff this intent can be expressed with an endogeneous transformation
`preconditions`	Conditions that must hold before applying intent
Associations	
`mandatory`	Property required for a transformation to have this intent
`optional`	Property optional for a transformation to have this intent
`relatedIntent`	Other intent often associated with current one

Table 1: Attributes of `ModelTransformationIntent`

Definition 2 is used to justify cases where it is not clear whether to use an attribute or a property to express a concept related to an intent. For example, the attribute **`endogenous?`** could also be expressed as a transformation property called, and then associating this as an optional property to each intent that can have endogenous transformations. In this case, the concept of being endogeneous does not satisfy Definition 2 since it is dependent only on the fact of whether the input/output types are the same or different and not on those input/output types' internal details. Thus, we do not use a property to express this concept.

2.2 Common Transformation Intents

As defined in [3], "a model transformation is an automated manipulation of models according to a specific intent". The following list of model transformation intents extends previous work [3, 4, 5]. The proposed list is not meant to be complete, but it nevertheless covers a wide spectrum of common transformation intents. Moreover, transformation chains may combine multiple intents in separate phases.

Manipulation Simple atomic or bulk operations on a model such as adding, removing, updating, accessing, or navigating through model elements is considered a model transformation when the system is completely and explicitly modeled.

Restrictive Query requests for some information about a model by a proper sub-model a.k.a. a *view*. We consider any subsequent aggregation or restructuring of the sub-model as an abstraction.

Refinement produces a lower level specification (*e.g.,* a platform-specific model) from a higher level specification (*e.g.,* a platform-independent model) [6]. As defined in [7], a model m_1 refines another model m_2 if m_1 can answer all questions that m_2 can answer. For example, a non-deterministic finite state automaton (NFA) can be refined into a deterministic finite state automaton (DFA).

Abstraction is the inverse of refinement: if m_1 refines m_2 then m_2 is an abstraction of m_1. For example, an NFA is an abstraction of a DFA.

Synthesis produces a well-defined language format that can be stored, such as in *serialization*: e.g., *model-to-code generation* produces source code in a target programming language (like generating Java code from UML class diagrams).

Reverse engineering is the inverse of synthesis: it extracts higher level specifications from lower-level ones: e.g., generating UML class diagram models from Java code (using e.g. Fujaba [8]).

Approximation is a refinement with respect to negated properties, as defined in [7], i.e. m_1 approximates m_2 if m_1 negates the answer to all questions that m_1 negates. In practice, m_2 is an idealization of m_1 where an approximation is only extremely likely: e.g., a Fast Fourier Transform is an approximation of a Fourier Transform which is computationally very expensive.

Translational Semantics defines a language's semantics in terms of another formalism, by specifying the semantic mapping through a transformation from the original language to the target semantic domain with well-defined semantics: e.g., Causal Block Diagram's semantics expressed as Ordinary Differential Equations.

Analysis A model transformation can be used to map a modeling language to a formalism that can be analyzed more appropriately than the original language. The target language is typically a formal language with known analysis techniques. For example, a Petri net model is transformed into a reachability graph on which liveness properties can be evaluated.

Simulation defines the *operational semantics* of a modeling language, by defining a model transformation that updates the modeled system's state: e.g., a transformation can simulate a Petri net model and produces a trace of the transition firing.

Normalization aims to decrease the syntactic complexity of models by translating complex language constructs into more primitive constructs, which results in a canonical form of a model: e.g., a Statechart model is normalized into its flattened form by removing OR- and AND-states.

Rendering assigns one (or several) concrete representation to each abstract syntax element or group of elements, as long as a meta-model of the concrete syntax is defined explicitly.

Model Generation aims at automatically producing possible (correct) instances of a metamodel, e.g., [9].

Migration transforms a software model written in one language (or framework) into another one, keeping models at the same abstraction level [10]: e.g., migrating Enterprise Java Beans 2.0 (EJB2) code in such a way that resulting models conforms to into EJB3.

Optimization aims at improving operational qualities of models such as scalability and efficiency: e.g., replacing n-ary association with binary associations in a UML class diagram may optimize generated code.

Refactoring is a restructuring that changes the internal structure of the model to improve certain quality characteristics without changing its observable behavior [11]: e.g. Zhang *et al.* [12] proposed a generic model transformation engine that can be used to specify refactorings for domain-specific models.

Composition integrates models produced in isolation into a compound model; where each isolated model typically represents a concern which may overlap: *model merging* creates a new model such that every element from each model is present exactly once in the merged model; whereas *model weaving* creates correspondence links between overlapping entities.

Synchronization integrates models evolving in isolation but subject to global consistency constraints, by propagating changes to these integrated models.

2.3 Common Transformation Properties

We now identify a relevant set of model transformation properties for our purposes, based on the transformation properties classification presented in [1], and a literature survey we conducted for this paper. Very little work exists in the literature for classifying model transformation properties. Therefore, and due to space limitations, we present a set of properties that is tentative and obviously incomplete, and rather focus on the transformation properties required for Section 3. Note that the two last properties are *non-functional* whereas all previous ones are *functional* [13].

Termination A terminating transformation produces an output model from an input model in finite time. Termination directly refers to Turing's halting problem, which is known to be undecidable for Turing-complete, model transformation languages, which has been addressed extensively, e.g., [14, 15, 16, 17];

Determinism A deterministic transformation always produces the same output model for the same input model. Determinism has been addressed extensively in several studies, including [18, 15, 19];

Type Correctness A type correct transformation ensures that both input and output models conform to their respective metamodels. Usually, *structural* conformance, involving only the metamodel, is distinguished from conformance w.r.t. additional *well-formedness rules* (*e.g.,*[20]);

Property preservation A transformation can preserve the syntactic [21, 22] or semantic [23, 24] properties of a model. For exogenous transformations, a formal property of the input model needs to be transformed into an equivalent property of the output model as e.g. investigated by Varro and Pataricza [24];

Traceability Most transformation languages allow logging a transformation's traceability links between the transformation's input and output model elements. This mechanism is often used to alleviate the problem of tracing back the analysis results from the transformation's input model to its output model as done in [21, 25]. In cases where the traceability of the analysis results is simple (e.g. in [26] where the termination of a transformation is decided by simulating a Petri Net that abstracts the transformation's semantics and that always runs out of tokens in finite time), less costly means may be employed.

Readability A transformation is readable if it is comprehensible and amenable to be read by humans. Distinct parts in a transformation may be individually readable: the input model, the output model or the transformation specification itself. Mens and Van Gorp mention in [4] the readability property of model transformations. To the best of our knowledge, little work is done on the readability of software models, but metrics do exist for evaluating software readability [27].

Mathematical underpinning A transformation has a mathematical underpinning if the transformation language's semantics and/or the input and output metamodels' semantics are mathematically formalised. Mens and Van Gorp [4] refers to the mathematical properties of transformation languages. There is vast literature on the formalisation of programming and modeling languages (e.g. [28]).

3. OVERVIEW OF THE ANALYSIS INTENT

Due to space limitations, we demonstrate one intent, namely the *analysis intent*. First, we overview literature studies that fall under this intent. We then summarize the commonalities of these studies using the proposed intent scheme.

Several example transformations from the literature fall under the analysis intent. Kühne *et al.* [29] defined the semantics of Finite State Automata in terms of Petri Nets. de Lara and Taentzer [30] implemented a graph rewriting system to transform process interaction models to timed transition Petri Nets for analysis. The graph rewriting system was proven to be terminating, deterministic, type correct and behaviour preserving (i.e., property preserving). Varró *et al.* [26] transformed graph rewriting systems into Petri Nets to analyze them for termination. König and Kozioura [31] proposed a tool, Augur2, that approximates graph rewriting systems as Petri Nets and analyzes them for property preservation. A property of interest is specified as a graph pattern which is transformed by Augur2 to an equivalent Petri Net marking. Accordingly, Augur2 either verifies that the property is satisfied or produces a counter example. Narayanan and Karsai [21] implemented a graph rewriting system in GREAT to transform UML activity diagrams to communicating sequential process models. The graph rewriting system was then checked for preserving structural correspondences between input and output models (property preservation). Narayanan and Karsai [23] implemented a graph rewriting system in GREAT to transform state charts to Extended Hybrid Automata (EHA) models for analysis. The graph rewriting system was then checked to preserve bisimilarity (i.e. property preservation) between input and output models. Rivera *et al.* [32] mapped graph rewriting systems to Maude and used reachability analysis and LTL model checking in Maude to analyze the graph rewriting system for property preservation. Properties were expressed as invariants, safety properties and liveness properties. Cabot *et al.* [33] derive OCL invariants from declarative model-to-model transformations to enable their analysis. In particular, several levels of executability are analysed such as applicability (a valid input model exists), executability (a valid output model exists), and totality (for every valid input model an output model exists).

Table 2 instantiates the intent domain metamodel of Figure 1 for the analysis intent, summarizing our findings in the literature as formerly described. In the studied literature most preconditions and mandatory properties stated in Table 2 seem to be fulfilled, although they have not always been explicitly stated as such or even verified for the particular case studies used in the corresponding papers. Our work aims at identifying these gaps in order to allow for a more systematic engineering of model transformations with specific intents in the future.

Attributes	
`name`	Analysis
`description`	To indirectly analyse a property of the input model by running the analysis algorithm on the transformation's output model
`useContext`	Need to analyse models that are not analysable in the transformation's input language, or are more efficiently analysable in the transformation's output language
`example`	Analyse termination of graph rewriting systems with Petri Nets [26]
`exogeneous?`	True
`endogeneous?`	True (if transforming to a profile of input model)
`preconditions`	1. Access to intended semantics; 2. Definition of the property of interest; 3. Existence of a verification method on the target language for analyzing the property of interest; 4. Existence of a method to translate the property of interest onto the transformation's output language (if the transformation is exogenous)
Associations	
`mandatory`	1. Termination 2. Type correctness 3. Preservation of the property of interest (specialises *Property preservation*) 4. Analysis result can be mapped back onto the input model (specialises *Traceability*)
`optional`	1. Readability of the transformation's output for debugging purposes 2. Formal definition of input language's semantics (specialises *Mathematical underpinning*)
`relatedIntent`	Translational Semantics, Simulation

Table 2: Analysis Intent

4. APPLYING THE ANALYSIS INTENT TO THE POWER WINDOW CASE STUDY

The power window case study [34] is an industrially oriented study on the application of transformations to Mde of software. This study directly interests us because it describes those transformations in terms of metamodels, model transformations and Uml 2.0 activity diagrams chaining, the process of building software to control an automobile's power window. A power window is basically an electrically powered vehicle window. The development of control software for such devices is nowadays highly complex due to the set of functionalities required for the comfort and security of the vehicle's passengers. This case study is relevant because it exposes in a detailed fashion a large number of transformations that span many intents identified in Section 2.2.

The power window case study's authors provide in their text, using varying degrees of detail, the context where their transformations occur and the properties those transformations should satisfy. We use this information as a means to validate our work. Several transformations from the power window case study apparently fall under the analysis intent. In Table 3 we summarize these transformations according to the classification of the analysis intent in Table 2. We only provide brief descriptions of the case study's transformations, which is entirely described in [34].

Several interesting questions are raised by the transforma-

Transformation	Description	Precond.	Mandatory	Optional
`EnvToPN` `PlantToPN`	Build a Petri net representation of a specialised model of the passenger's interactions with the powerwindow (resp. of a specialised model of the powerwindow physical configuration). Check power window security requirements	(1),(2), (3)	(1),(2), (3)	
`ScToPN`	Build a Petri net representation of a specialised model of the powerwindow control software. Check power window security requirements.	(1),(2), (3)	(1),(2), (3)	(1),(2)
`ToBinPacking-Analysis`	Build an equational algebraic representation of the dynamic behavior of the involved hardware components from an AUTOSAR [35] specification. Check processor load distribution.	(1),(2), (3),(4)	(1),(2), (3),(4)	(1)
`ToSchedulability-Analysis`	Build an equational algebraic representation of the dynamic behavior of the involved hardware and software components from an AUTOSAR specification. Check software response times.	(1),(2), (3),(4)	(1),(2), (3),(4)	(1)
`ToDeployment-Simulation`	Build a DEVS representation of the deployment solution. Check latency times, deadlocks and lost messages.	(1),(2), (4)	(1),(2), (3),(4)	(1)

Table 3: Model transformation examples from the case study falling under the analysis intent

tions we describe in Table 3. First, only two of the transformations fulfill the four preconditions listed in Table 2. This may point to two issues: the transformation does indeed have the *analysis* intent, but has not been fully implemented; the transformation does not have the *analysis* intent. After looking at the detailed description of the transformations in Table 3, we found that transformations `EnvToPN`, `PlantToPN` and `ScToPN` are missing precondition (4) (property translation implementation) described in Table 2. Work to address that problem within the case study is foreseen. On the other hand, we found that the `ToDeploymentSimulation` transformation has the *simulation* rather than the *analysis* intent, which seems to be a good indicator of the discriminating power of Table 2.

Regarding the mandatory properties, the `EnvToPN`, `Plant-ToPN` and `ScToPN` transformations do not implement property (4) of Table 2. As previously, this is mainly due to the fact that the case study is not fully developed. In fact, traceability for interpreting the analysis result on the input model is yet to be implemented.

Finally, regarding the optional properties, the results in Table 3 are to be expected. Some of the transformations do exhibit the optional properties while others do not. This indicates that our choice for such properties is indeed correct, although most likely not complete.

5. RELATED WORK

The notion of *intent* in the software engineering discipline is not new. Yu and Mylopoulos [36] realized in 1994 that current research in this area was more focused on design and implementation—the *what* and the how for developing software—rather than on the requirements necessary to understand the software to improve the underlying production processes—the *why*. To a certain extent, MDE is following the same path: historically, research was more devoted to the management of different modelling and transformation activities instead of exploring the intents behind them.

Four contributions [4, 5, 37, 38] are related to our study; all aiming for a classification of different transformation aspects. Mens and Van Gorp [4] provide a multidimensional taxonomy of transformations exhibiting several syntactic classification dimensions. The dimensions are illustrated on transformations related to our intents. However, our catalog aims at reflecting known, documented *uses* of transformations and proposes, in addition to the seven intents presented in [4], ten additional intents whereas one of them is characterised by its properties.

In [37], design patterns for model transformations expressed in QVT Relations are presented. However, the intents behind the transformations are not discussed.

Tisi *et al.* [38] examined higher-order transformations (HOT), i.e., transformations manipulating transformations. They classify them based on whether source and/or target models are transformations or not. Our intents are more general in the sense that we do not distinguish between transformations and HOT allowing for a wider applicability of the intent catalog.

The goal of Czarnecki and Helsen [5] was to classify the features of transformations languages by establishing a feature model: they introduced five intended applications of transformations which are also reflected by our catalog.

A taxonomy of program transformations is presented by Visser [39]. Instead of proposing a taxonomy of multiple dimensions as in [4], Visser employs one discriminator for the taxonomy: out-place vs. in-place transformations (named as *translations* and *rephrasing*). Some of the leaf nodes in the taxonomy are program-specific, *e.g.,* (*de-*)*compilation*, *inlining*, and *desugaring*. However, other nodes in the taxonomy are covered by our intent catalog. Moreover, we present several intents specifically tailored to model transformations.

To sum up, the presented transformation intent catalog is more comprehensive than previous attempts. Besides providing a name and an example of each intent, this catalog propsoes comprehensive meta-information (e.g., the use context, preconditions, etc.) and properties of interest for a given intent. To the best of our knowledge, the latter has not been subject of research in previous work.

6. CONCLUSION AND DISCUSSION

In this paper, we have presented our ongoing work on using the notion of *intent* to help us understand the uses of model transformations in MDE and how they can be best supported. More concretely, we listed some common transformation intents and properties, presented a schema to describe intents, and briefly illustrated its use on a case study.

Future work includes making our catalog more comprehensive, as well as describing other intents, which is already started. We also plan on identifying certification methods that allow a given transformation property to be analyzed, together with suitable references to corresponding research efforts on the analysis and verification of transformations.

On the more abstract level, we hope to gain a better understanding of how potential uses of the catalog outlined in the introduction can best be realized, if at all. For instance, it is currently unclear how "crisply" intents and their properties can be described and how useful our descriptions are in practice, as transformations in practice may have overlapping intents and properties that span too large a spectrum. Also, it may be more useful to think of intents as a form of "requirements patterns" [40] for transformations. To support transformation analysis, a formal framework for the description of properties and a related extension of our intent metamodel will be developed. Finally, how intents and the certification of properties can be best supported by, possibly, dedicated transformation languages, is another topic for future work.

7. REFERENCES

[1] M. Amrani, L. Lúcio, G. Selim, B. Combemale, J. Dingel, H. Vangheluwe, Y. Le Traon, and J. R. Cordy, "A Tridimensional Approach for Studying the Formal Verification of Model Transformations," in VOLT *Workshop*, 2012.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994.

[3] E. Syriani, "A Multi-Paradigm Foundation for Model Transformation Language Engineering," Ph.D. Thesis, McGill University, 2011.

[4] T. Mens and P. Van Gorp, "A Taxonomy Of Model Transformation," ENTCS, vol. 152, pp. 125–142, 2006.

[5] K. Czarnecki and S. Helsen, "Feature-Based Survey of Model Transformation Approaches," IBM *Systems J.*, vol. **45**(3), pp. 621–645, 2006.

[6] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley, 2003.

[7] Holger Giese, Tihamer Levendovszky, and Hans Vangheluwe, "Summary of the Workshop on Multi-Paradigm Modeling: Concepts and Tools," in *Models in Software Engineering*, vol. 4364, 2007.

[8] T. Fischer, J. Niere, L. Turunski, and A. Zündorf, "Story Diagrams: A New Graph Rewrite Language Based on UML and Java," in *Theory and Application of Graph Transformations*, 2000, pp. 296–309.

[9] J. Winkelmann, G. Taentzer, K. Ehrig, and J. Küster, "Translation of Restricted OCL Constraints into Graph Constraints for Generating Meta Model Instances by Graph Grammars," ENTCS, vol. 211, pp. 159–170, 2008.

[10] P. Mc Brien and A. Poulovassi, "Automatic Migration and Wrapping of Database Applications - A Schema Transformation Approach," in *Conceptual Modeling ER*, vol. 1782, 1999, pp. 99–114.

[11] W. G. Griswold, "Program Restructuring as an Aid to Software Maintenance," Ph.D. dissertation, University of Washington, August 1991.

[12] J. Zhang, Y. Lin, and J. Gray, "Generic and Domain-Specific Model Refactoring Using a Model Transformation Engine," in *Research and Practice in Software Engineering (Vol. II)*, 2005, pp. 199–218.

[13] I. Sommerville, *Software Engineering.* Addison-Wesley.

[14] H.-K. Ehrig, G. Taentzer, J. de Lara, D. Varró, and S. Varró Gyapai, "Termination Criteria for Model Transformation," in FASE, 2005.

[15] J. M. Küster, "Definition and Validation of Model Transformations," SoSyM, vol. **5**(3), pp. 233–259, 2006.

[16] H. S. Bruggink, "Towards a Systematic Method for Proving Termination of Graph Transformation Systems," ENTCS, vol. **213**(1), 2008.

[17] F. Spoto, P. M. Hill, and E. Payet, "Path-Length Analysis of Object-Oriented Programs," in EAAI, 2006.

[18] R. Heckel, J. M. Küster, and G. Taentzer, "Confluence of Typed Attributed Graph Transformation Systems," in ICGT, 2002.

[19] L. Lambers, H. Ehrig, and F. Orejas, "Efficient Detection of Conflicts in Graph-based Model Transformation," ENTCS, vol. 152, 2006.

[20] A. Boronat, "MoMENT: A Formal Framework for Model manageMENT," Ph.D. dissertation, University of Valencia, 2007.

[21] A. Narayanan and G. Karsai, "Verifying Model Transformations by Structural Correspondence," ECEASST, vol. 10, 2008.

[22] L. Lúcio, B. Barroca, and V. Amaral, "A Technique for Automatic Validation of Model Transformations," in MO*DELS*, 2010, pp. 136–150.

[23] A. Narayanan and G. Karsai, "Towards Verifying Model Transformations," ENTCS, vol. 211, pp. 191–200, 2008.

[24] Dániel Varró and András Pataricza, "Automated Formal Verification of Model Transformations," in CSDUML *Workshop*, 2003, pp. 63–78.

[25] L. Lúcio, Q. Zhang, V. Sousa, and Y. Le Traon, "Verifying Access Control in Statecharts," ECEASST, 2012.

[26] D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer, "Termination Analysis of Model Transformations by Petri Nets," *International Conference on Graph Transformations*, pp. 260–274, 2006.

[27] R. P. Buse and W. R. Weimer, "A metric for software readability," in *Proceedings of ISSTA '08.* NY, USA: ACM, 2008, pp. 121–130.

[28] D. Harel and B. Rumpe, "Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff," Israel, Tech. Rep., 2000.

[29] T. Kühne, G. Mezei, E. Syriani, H. Vangheluwe, and M. Wimmer, "Systematic Transformation Development," ECEASST, vol. 21, 2009.

[30] J. de Lara and G. Taentzer, "Automated Model Transformation and its Validation Using AToM3 and AGG," in *Diagrams*, 2004, pp. 182–198.

[31] B. König and V. Kozioura, "Augur 2–A New Version of a Tool for the Analysis of Graph Transformation Systems," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 211, pp. 201–210, 2008.

[32] J. Rivera, E. Guerra, J. de Lara, and A. Vallecillo, "Analyzing Rule-Based Behavioral Semantics of Visual Modeling Languages with Maude," *Software Language Engineering*, pp. 54–73, 2009.

[33] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara, "Verification and Validation of Declarative Model-to-Model Transformations Through Invariants," *JSS*, vol. **83**(2), pp. 283–302, 2010.

[34] L. Lúcio, J. Denil, and H. Vangheluwe, "An Overview of Model Transformations for a Simple Automotive Power Window," McGill University, Tech. Rep. SOCS-TR-2012.2, 2012, http://msdl.cs.mcgill.ca/people/levi/AMT/material/.

[35] AUTOSAR, "http://www.autosar.org," 2010.

[36] E. S. Yu and J. Mylopoulos, "Understanding "Why" in Software Process Modelling, Analysis, and Design," in ICSE, 1994, pp. 159–168.

[37] M.-E. Iacob, M. W. A. Steen, and L. Heerink, "Reusable Model Transformation Patterns," in *EDOCW'08*, 2008, pp. 1–10.

[38] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the Use of Higher-Order Model Transformations," in ECMDA-FA, 2009, pp. 18–33.

[39] Eelco Visser, "A Survey of Strategies in Rule-Based Program Transformation Systems," *J. Symbolic Computation*, vol. **40**(1), pp. 831–873, 2005.

[40] S. Withall, *Software Requirement Patterns.* Microsoft Press, 2007.