Journal of Engineering and Technology for Industrial Applications

ISSN ONLINE: 2447-0228

ITEGAM-JETIA

Manaus, v.12 n.58, p. 538-548. March/April, 2026.
DOI: https://doi.org/10.5935/jetia. v12i58.3176

RESEARCH ARTICLE OPEN ACCESS

CONSISTENCY VALIDATION OF TRANSFORMING UML STATECHART MODELS TO FLAT STATE MACHINES MODELS WITH USE APPROACH

Ali Lalouci*[1,2]

[1]Department of Computer Science, Abdelhafid Boussouf University of Mila, 43000 Mila, Algeria .
[2]Laboratoire de Modélisation Statistique, Sciences des Données et Intelligence Artificielle (MS2DIA), Mohammed Seddik Ben Yahia University of Jijel, 18000, Jijel, Algeria.

[1]https://orcid.org/0009-0001-7228-1206

Email:*ali.lalouci@centre-univ-mila.dz

ARTICLE INFO

Article History
Received: December 20, 2025
Reviewed: January 9, 2026
Accepted: January 16, 2026
Published: March 31, 2026

Keywords:
Model-Driven Engineering,
Software Engineering,
Model Transformation,
Transformation Model,
Consistency Validation,
USE Approach.

ABSTRACT

Model-Driven Engineering (MDE) supports software development through the systematic use of models, meta-models, and model transformations, making transformation validation a critical concern. This paper addresses the consistency validation of a transformation from UML statechart diagrams to Flat State Machines (FSMs). The transformation is formalized as a transformation model using a UML class diagram enriched with Object Constraint Language (OCL) invariants. The UML-based Specification Environment (USE) model validator is used to automatically verify transformation consistency and its implied properties, including weak consistency, class instantiability, and class and association instantiability. A case study based on an Automated Teller Machine (ATM) system demonstrates the effectiveness of the proposed approach in supporting reliable software development within Model-Driven Engineering.

I. INTRODUCTION

Engineering systems are becoming increasingly complex due to the growing number of components and functions, as well as the involvement of multiple engineering disciplines. Managing this complexity has become a significant challenge, and MDE has emerged as a promising approach to address it [1]. Rather than relying primarily on document-based descriptions, MDE emphasizes the use of models as central engineering artifacts throughout the development process. These models capture and maintain relevant system information, are shared among stakeholders, and are manipulated using Computer-Aided Software Engineering (CASE) tools to support automation, for instance through model transformations. By enabling models to be processed by machines, MDE supports more informed decision-making across the system development lifecycle [2].

Model transformation is an important approach in MDE. It consists of a set of rules allowing the mapping of one model to another, by defining for each element of the source model its equivalent among the elements of the target. These rules are carried out by a transformation engine that reads the source model which must conform to the source meta-model, and applies the rules defined in the model transformation to produce the target model which will be itself conform to the target meta-model. From the MDE point of view, a transformation can be seen as a *transformation model* or *a model transformation* [3]. A transformation model can denote a computation, whose expression relies on a particular model of computation embedded in a transformation language. Model transformation focuses on the particular artifacts manipulated by a transformation (namely, meta-models for its specification and models for its execution) [4].

The computational nature will require that the underlying language guarantees some desirable properties, like termination and determinism, needed for the purpose of the transformations, independently of what a particular transformation expresses. On the other hand, manipulating models implies that specific properties of interest will be directly related to the involved models as well as the intrinsic intent behind the transformation: e.g., one may be interested in always producing conforming models by construction, or ensuring that target models still conserve their semantics [4]. Thus, model transformation provides important information about the relation between the elements connected by the transformation. Different verification approaches have been applied to model transformation in the literature with the two previous views.

In [5], [6], the authors propose a transformation from UML statechart diagrams to Flat State Machines (FSMs). Statechart diagrams are typically used to model the behavior of complex systems using a relatively informal notation, whereas FSMs constitute one of the most widely adopted behavioral models in computer programming and are defined using a more formal notation. This transformation represents a key stage in an automated methodology and supporting tool environment that aims to formally convert the dynamic behavior of systems specified in UML models into their equivalent Colored Petri Net representations, thereby enabling formal analysis and verification. Although numerous studies in the literature refer to and build upon this transformation (see, for instance, [7-10]), a formal validation is still lacking, primarily due to the complexity of the validation process. This paper addresses this gap by representing the transformation as a transformation model and assessing its consistency and implied properties through the application of the USE approach. The main objectives of this study are as follows:

1. Formally model the transformation from UML statechart diagrams to Flat State Machines as a transformation model using UML class diagrams enriched with OCL invariants.
2. Define and analyze the consistency property of the proposed transformation model and its implied properties, including weak consistency, class instantiability, and class and association instantiability.
3. Apply the USE model validator to automatically verify the consistency of the transformation model.
4. Illustrate the feasibility and effectiveness of the proposed approach through a realistic case study based on an Automated Teller Machine (ATM) system.

The remainder of this paper is organized as follows: Section II reviews related work on model transformation validation and the USE approach. Section III introduces the background and context of this study, detailing the transformation from UML statechart diagrams to Flat State Machines, including their respective meta-models, the underlying graph grammar rules, and an overview of the USE model validator and its supported verification properties. Section IV describes the proposed transformation model, which formalizes the statechart-to-FSM transformation using a UML class diagram enriched with OCL invariants to capture structural and semantic constraints. Section V is devoted to the validation of the proposed transformation model, where transformation consistency and its implied properties are defined and evaluated through a practical case study using the USE tool. Finally, Section VII concludes the paper by summarizing the main contributions and discussing potential directions for future research.

II. RELATED WORKS

Model transformation analysis and validation have been extensively studied, particularly in the context of UML, OCL, and database-related transformations. Several works leverage the USE (UML-based Specification Environment) tool to formally analyze transformation correctness and properties. Büttner and Bauerdick [11] introduce an early and influential approach for realizing UML model transformations using USE. Their work primarily focuses on the handling of OCL invariants and expressions attached to UML class diagrams. A key contribution is the demonstration of how invariants must be systematically adapted when structural modifications occur during transformations, such as changes in association multiplicities or class structures.

This work establishes USE as a practical environment for validating transformation-related constraints. [12], [13] further advance this line of research by studying transformations between Entity–Relationship (ER) schemata and relational database schemata. They represent transformations as explicit transformation models that connect source and target meta-models. Using USE, the authors analyze essential transformation properties, including consistency and the presence of implied properties. Their approach enables the detection of semantic mismatches between source and target models and provides formal guarantees about transformation correctness. Building upon this foundation, [14] focus on invariant independence in transformation models.

Their work addresses the problem of redundancy in invariant sets by checking whether invariants can be derived from others. By applying USE-based analysis techniques, they show how invariant independence can be formally verified, thereby improving model clarity, maintainability, and correctness in UML and OCL-based transformations. Beyond structural and invariant-based analysis, [15] extend UML model validation to the temporal dimension. They propose a methodological framework and tool, TPV, that allows designers to specify and validate temporal properties in UML class diagrams.

This work broadens the scope of model analysis by enabling the verification of time-related constraints, which are increasingly important in complex and dynamic systems. Finally, [16] address the testing aspect of model transformations by proposing a specification-driven testing framework. Their approach systematically integrates the main phases of model transformation testing—test design, execution, and evaluation—into a unified framework. This work complements formal analysis approaches by emphasizing systematic testing as a means to increase confidence in transformation correctness.

III. BACKGROUND AND CONTEXT

This section provides the background and context of the study. It introduces the transformation considered in this work and describes the approach adopted to validate the proposed transformation**.**

III.1 STATECHART MODEL TO FLAT STATE MACHINE MODEL TRANSFORMATION

In this subsection, we describe the transformation process from a UML statechart diagram to a Flat State Machine (FSM), as presented in [6].

III.1.1 Statechart Diagram Meta-Model

A statechart diagram consists of states, both simple and composite, and transitions that are associated with events and actions. Each state is characterized by several elements, including a name, which is a textual identifier that may be omitted, as well as entry and exit actions that are executed upon entering and leaving the state, respectively.

Composite states exhibit a nested structure of substates, which may be organized either sequentially or concurrently. Statechart transitions are represented by standard finite state machine arcs, specifying a change from a source state to a target (successor) state in response to an event and the execution of associated actions. Figure 1 illustrates the metamodel for UML statechart diagrams as proposed by the authors in [6].

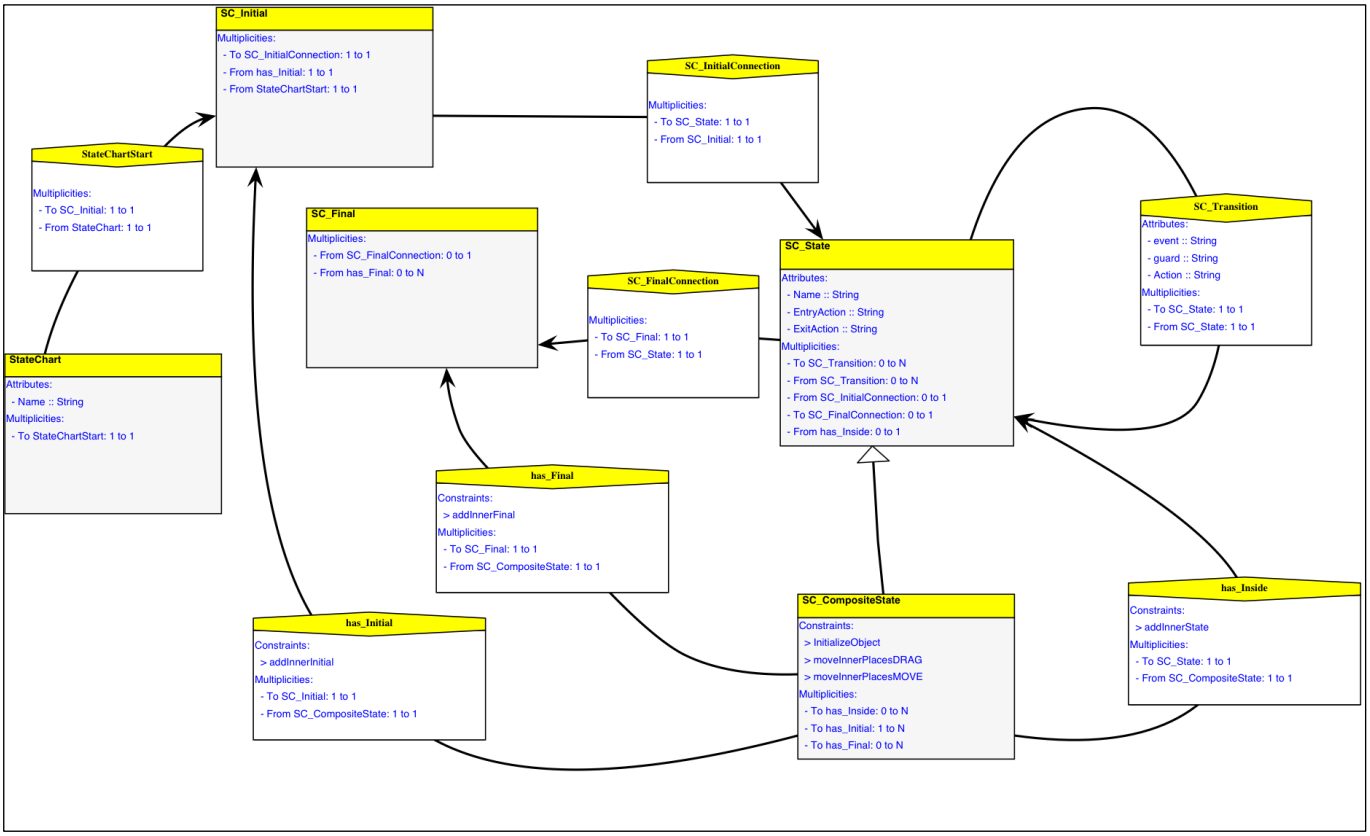

Figure 1: Meta-model of statecharts.
Source: [6].

III.1.2 Flat State Machines Meta-Model

A Flat State Machine (FSM) is a state machine that does not contain composite states. In an FSM, states are represented by rounded boxes, while transitions between states are depicted as directed arcs. Figure 2 illustrates the meta-model of the FSM formalism as presented in [6].

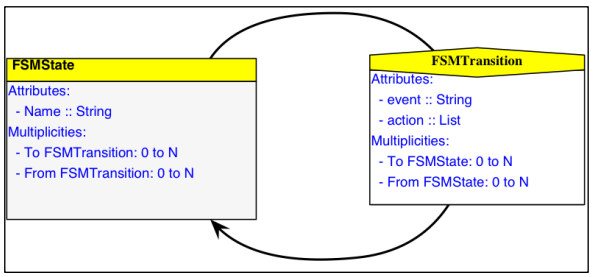

Figure 2: Meta-model of Flat State Machines (FSM).
Source: [6].

III.1.3 Graph Grammar of the Transformation

In the following, we present the considered transformation as described in [6]. The authors refer to this graph grammar as Statechart2FSM. To transform a statechart into its equivalent finite state machine (FSM) model using the Statechart2FSM grammar, they propose twenty-three transformation rules (see Figures 3 and 4), which are applied in ascending order of priority. Each rule is associated with a specific priority level. The overall transformation process implemented by the Statechart2FSM grammar can be summarized by the following main steps:

1. **Selection and initialization**

- A statechart is first selected from the set of available statechart diagrams.
- The initial state of the selected statechart is converted into an FSM state:
  - Rule #22 is applied for a simple initial state.
  - Rule #23 is applied for a composite initial state.

2. **Statechart traversal and transition conversion**

- The transformation then traverses the selected statechart starting from its initial state, which is processed in the previous step.
- For each processed statechart state:
  - Every outgoing (exit) transition is converted into a corresponding FSM transition.
  - The destination statechart state is also converted into an FSM state if it has not been previously processed (rules #5 and #6).

3. **Handling of composite states**

- For composite states, the graph grammar uses flag attributes to manage the traversal and processing status:
  - T = 1 indicates that the composite state has been traversed.
  - P = 1 indicates that the composite state has been fully processed.
- Examples of such transformations are illustrated by rules #7, #8, and #9.

4. **Conversion of composite states**

- In the third step, all composite states present in the statechart (if any) are converted.
- The conversion process follows the same principles applied to simple statechart diagrams, as defined by rules #1 and #2.

5. **Integration into the global FSM model**

- Finally, the FSM segment corresponding to each composite state is connected to the overall FSM model of the statechart.
- For instance, rule #14 transforms an exit transition from a composite state to a simple state into its equivalent transition in the FSM model.

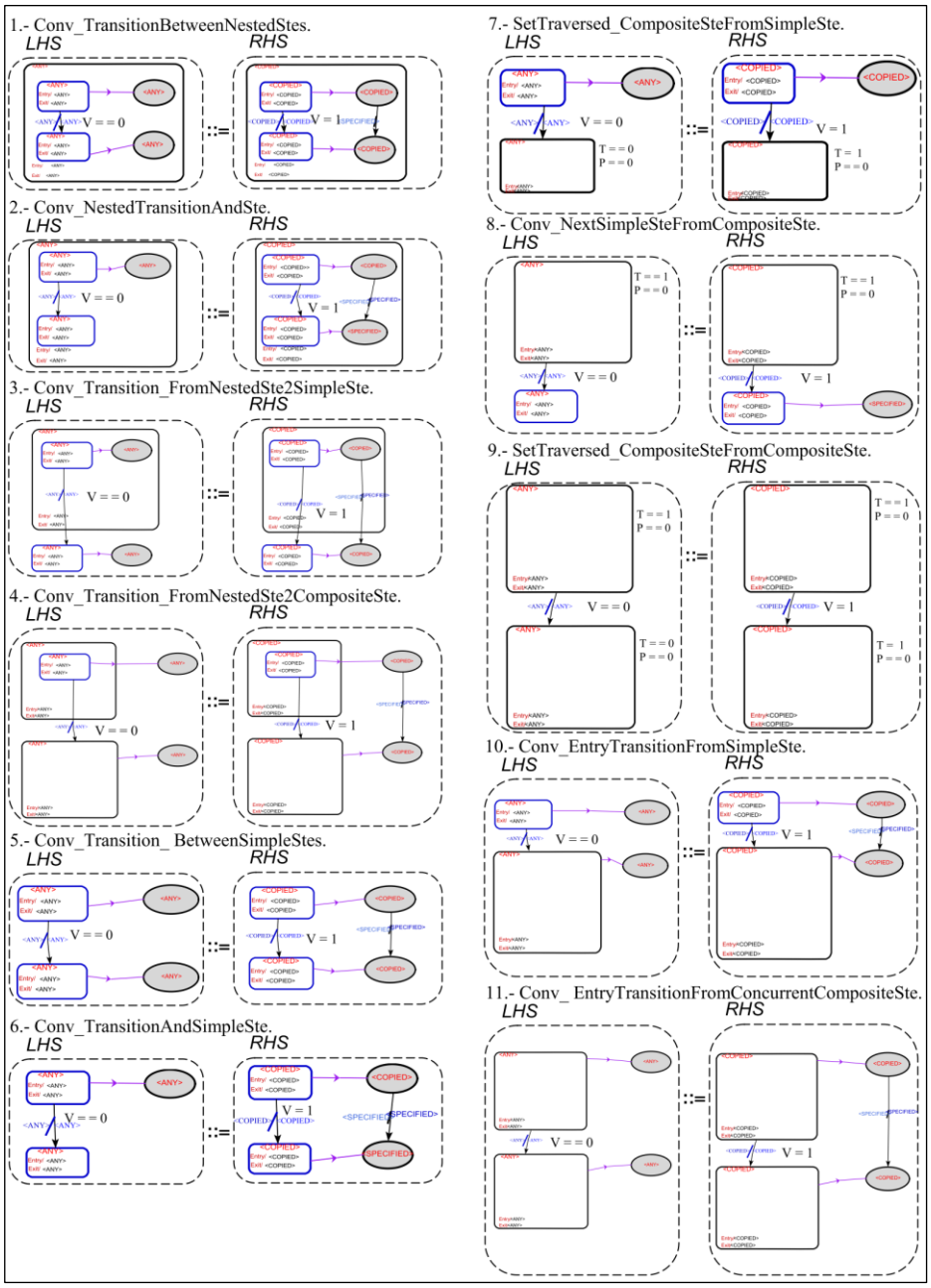

Figure 3: Graph Grammar Rules 1-11.
Source: [6].

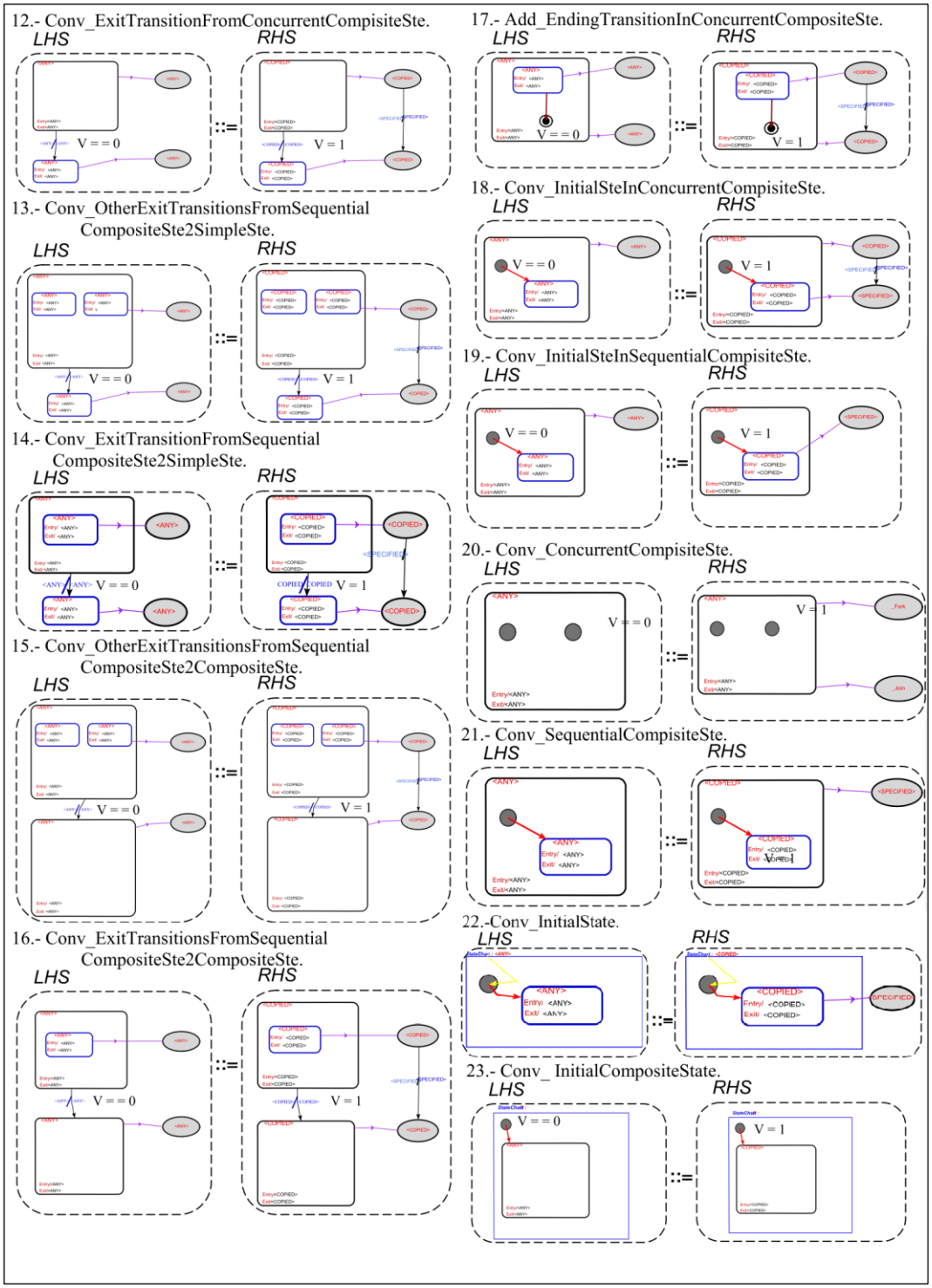

Figure 4: Graph Grammar Rules 12-23.
Source: [6].

III.2 USE APPROACH

USE is an acronym for UML-based Specification Environment, designed to support the specification of information systems and software, as well as the validation of models [3], [7]. One of the main applications of the USE tool lies in verification and validation. From its inception, a strong OCL evaluator has been a core component of USE, continuously enhanced and adapted to support new features and successive updates of the OCL standard. In addition, a variety of validation and model-finding approaches have been developed on top of this foundation, enabling the tool to remain a relevant and competitive solution in the field of model checking.

The importance of model validation is widely recognized, and numerous approaches have been proposed that rely on diverse tools and model transformation techniques to address the associated challenges [17]. A USE specification includes a textual description of a model, utilizing elements found in UML class diagrams, such as classes, associations, and other features. Additional integrity constraints on the model are specified through expressions written in the Object Constraint Language (OCL). The model can be animated to validate the specification against informal requirements. Figure 5 presents an overview of the various properties that can be verified within the USE model validation framework.

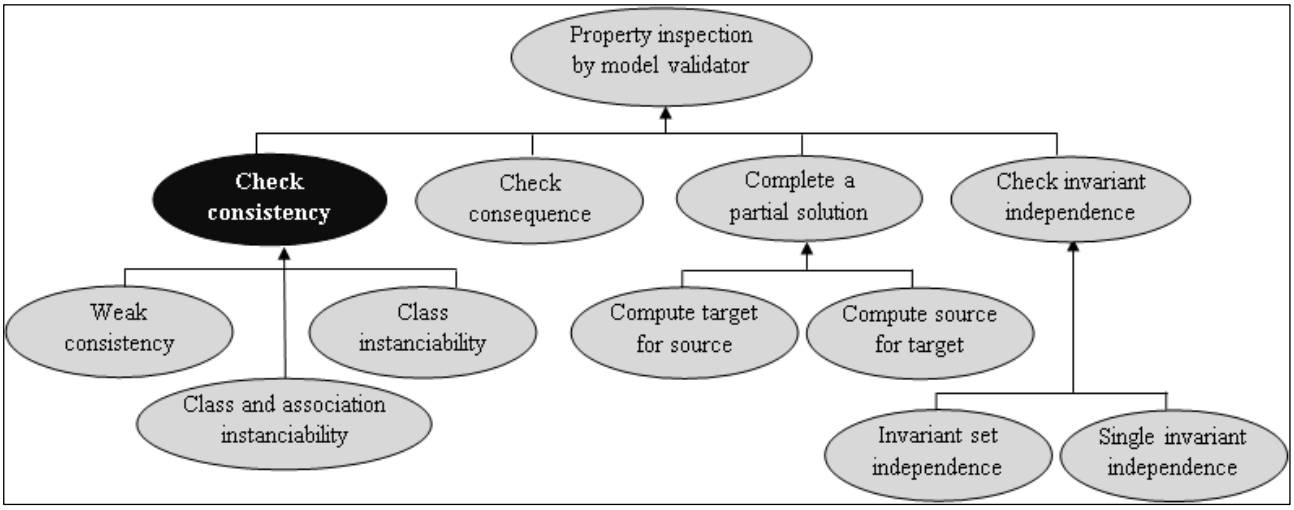

Figure 5: Properties verified by USE Model validator.
Source: Authors, (2026).

In the context of this work, we focus specifically on the *consistency property* of the considered transformation. Consistency constitutes a fundamental prerequisite for any model transformation, as it ensures that the set of structural and semantic constraints defined over the transformation model is non-contradictory and jointly satisfiable. By selecting consistency as the primary validation property, we aim to establish the existence of at least one valid instance of the transformation model that simultaneously satisfies all specified OCL invariants. This choice allows us to provide a solid foundation for further analysis and to guarantee that the transformation from UML statechart models to Flat State Machine models is well-defined within the USE framework.

IV. THE PROPOSED TRANSFORMATION MODEL

In this section, we adopt the USE-based approach proposed by [12]. Accordingly, we begin by modeling the transformation presented in Subsection III.1 as a transformation model specified by a UML class diagram enriched with OCL invariants. The resulting transformation model is illustrated in Figure 6.

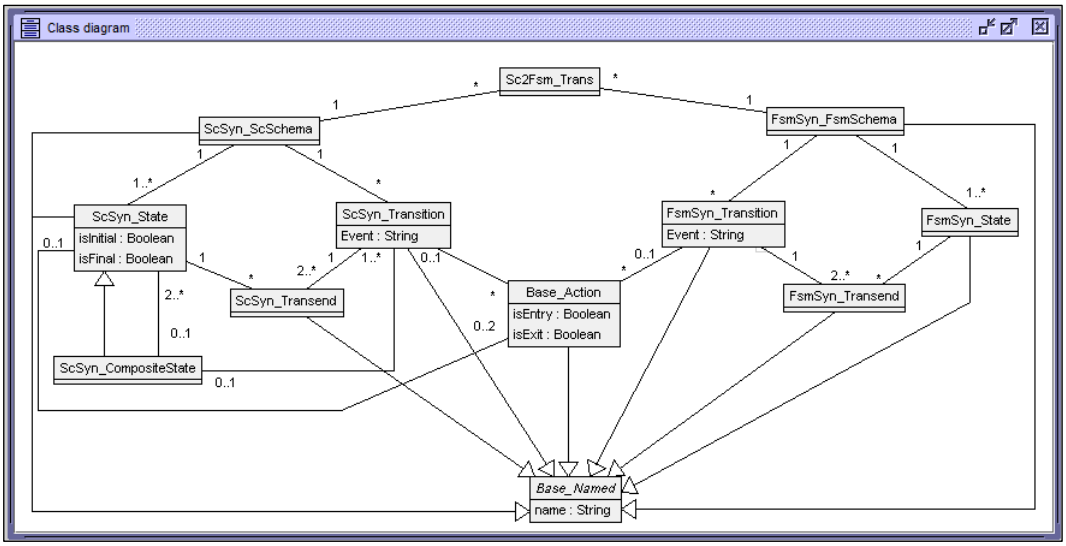

Figure 6: Class diagram for transformation model.
Source: Authors, (2026).

Our proposed transformation model comprises twelve classes interconnected through associations. In line with the approach presented in [12], the model includes an anchor class for the StateChart (SC) model, an anchor class for the Flat State Machine (FSM) model, and an additional connecting class that represents the transformation object. The class diagram is structured into four distinct parts. The first part defines a base layer that captures action-related concepts commonly employed in both the SC and FSM models. The second part corresponds to the StateChart schema (*ScSchema*), encompassing the concepts of *State*, *Transition*, and *TransEnd* (transition end). The third part represents the Flat State Machine schema (*FsmSchema*), which similarly includes the concepts of *State*, *Transition*, and *TransEnd*. Finally, the fourth part is dedicated to the transformation itself (*Trans*). Figure 7 depicts the invariant names associated with the transformation model. Constraints expressed in the Object Constraint Language (OCL) are employed to specify additional integrity conditions on our transformation model. As shown, all class invariants defined for the proposed transformation model are satisfied. Note that these OCL invariants are satisfied when instances are constructed for model transformation. It is used to restrict the source meta-model, for the target meta-model, and for the transformation. In the following, we present the details of two selected constraints.

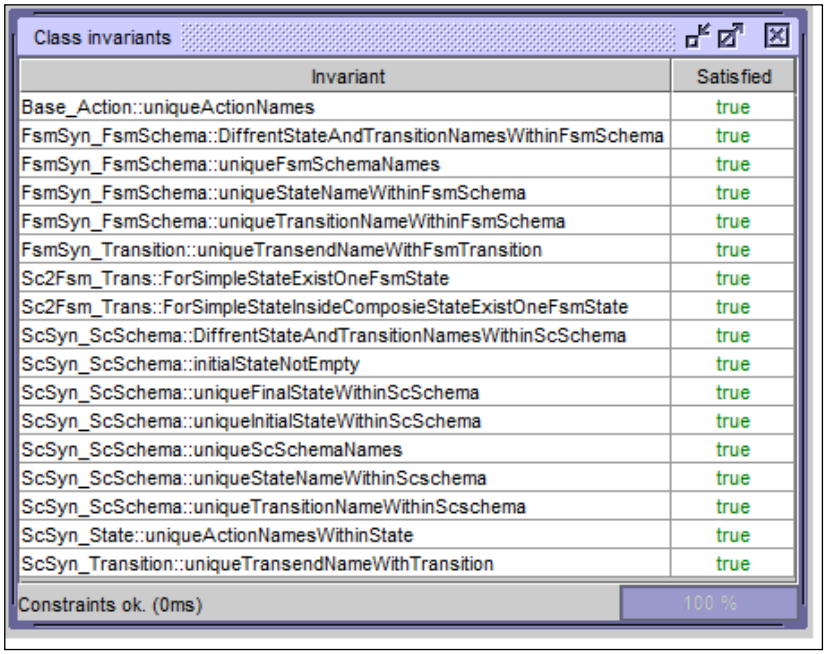

Figure 7: Class invariants for transformation model.
Source: Authors, (2026).

`ScSyn_ScSchema::uniqueStateNameWithinScschema:` Within one Statechart schema, different States have different names, i.e., two distinct States have distinct names.

```
Contextself:ScSyn_ScSchemainvuniqueStateNameWithinScschema:self.state
->forAll(s1,s2|s1.name=s2.name implies s1=s2)
```

`ScSyn_ScSchema::uniqueInitialStateWithinScSchema:` within one Statechart schema, one initial State exists.

```
Contextself : ScSyn_ScSchemainvuniqueInitialStateWithinScSchema :
self.state ->forAll( s1,s2 | s1.isInitial=true and s2.isInitial=true
                    implies s1=s2 )
Contextself : ScSyn_ScSchemainvinitialStateNotEmpty:
self.state ->select(s|s.isInitial)->notEmpty
```

V. CHECKING TRANSFORMATION MODEL CONSISTENCY

This section presents a practical case study designed to illustrate the transformation model validation process. To this end, the validation procedure is conducted in three main steps. First, we formally define the consistency property along with its associated implied properties. Second, we introduce a representative and non-trivial example. Finally, we instantiate the transformation model and verify its consistency.

V.1 CONSISTENCY PROPERTY

Within the USE approach, model consistency is established by the existence of a valid object diagram. In this work, we automatically assess the consistency of the transformation model by constructing valid model instances. The successful construction of such instances demonstrates that the specified constraints are non-contradictory [12]. According to the USE framework, three levels of consistency must be considered:

V.1.1 Weak Consistency

Weak consistency indicates that at least one valid object diagram can be generated, possibly including empty populations for one or more classes, or even no objects at all, provided that such configurations are permitted by the model and its settings [12].

V.1.2 Class Instantiability

Class instantiability requires that each class in the model is instantiated with a non-empty population [12].

V.1.3 Class and Association Instantiability

Class and association instantiability ensures that all classes and all associations in the model are instantiated with non-empty populations [12].

V.2 PRESENTATION OF THE EXAMPLE

We consider an example of an Automated Teller Machine (ATM) that dispenses cash to a user, originally presented in [18], [8] and implemented in [5], [6] using a graph transformation–based approach. The AToM³ meta-modeling tool is employed for this purpose. The execution of the *Statechart2FSM* graph grammar on the Statechart diagram shown in Figure 8 produces the corresponding FSM model illustrated in Figure 9.

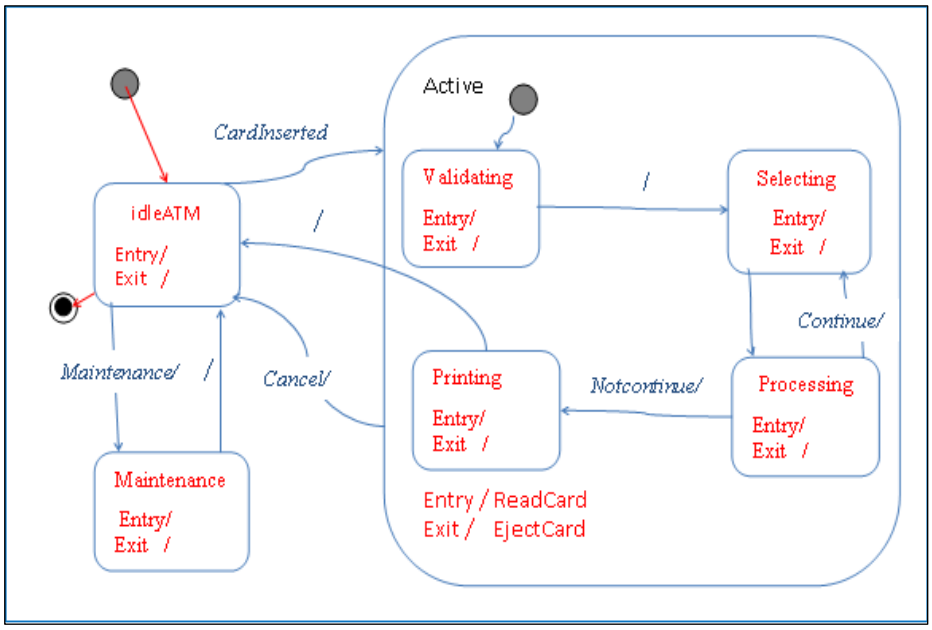

Figure 8: Statechart diagram for ATM_Machine (source model).
Source: Authors, (2026).

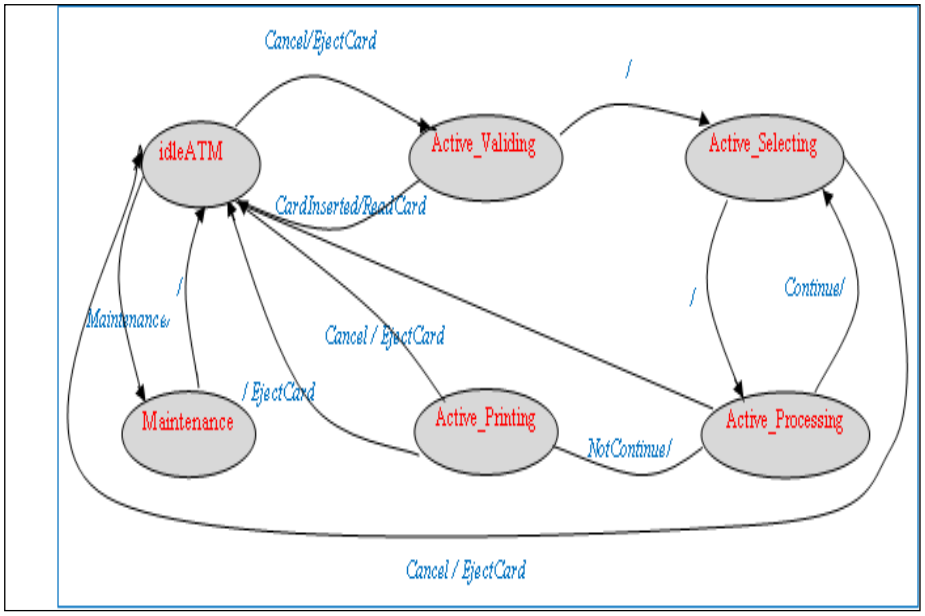

Figure 9: Flat State Machine model for ATM_Machine (target model)
Source: Authors, (2026).

V.3 CHECKING TRANSFORMATION MODEL CONSISTENCY

To validate our proposed transformation model, we employed the USE model validator on the source and target models of the case study presented in Figures 8 and 9. The transformation was encoded in the textual specification language of USE, and the model validator was then used to automatically construct valid object diagrams that satisfy all defined OCL invariants. The resulting object diagrams, shown in Figures 10, 11, and 12, represent a single instance-level realization of the transformation model, displayed in three parts for readability. In USE terminology, such an object diagram corresponds to an instance specification of the UML class diagram and explicitly depicts instantiated classes (objects), attribute values (slots), and instantiated associations (links). The existence of this object diagram constitutes a constructive proof, by example, of the consistency of the transformation model.

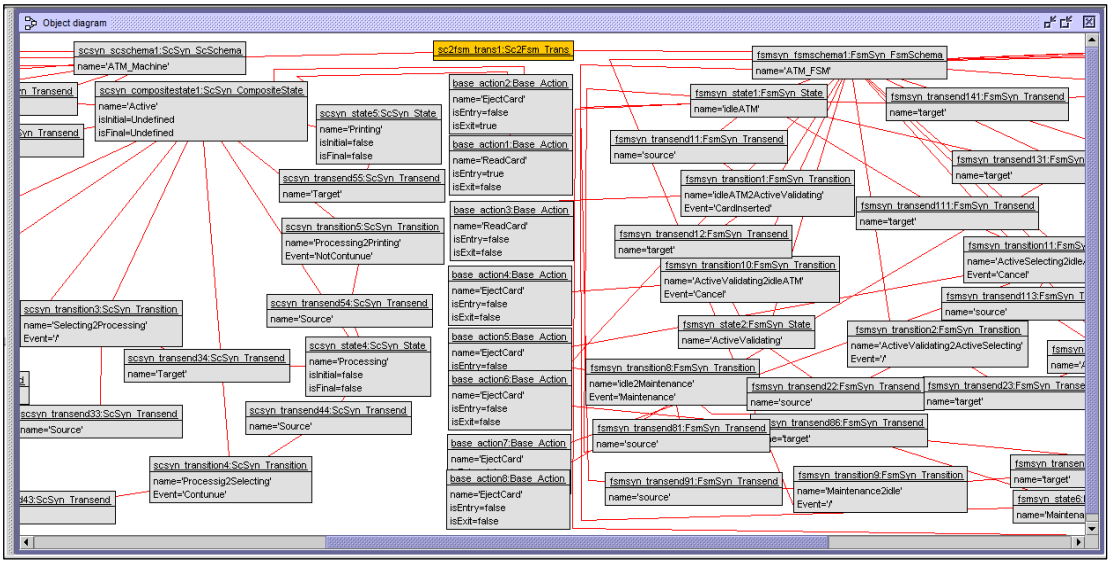

Figure 10: Generated proof example for weak consistency, class, class and association instantiability (Sc2FSM_Trans part).
Source: Authors, (2026).

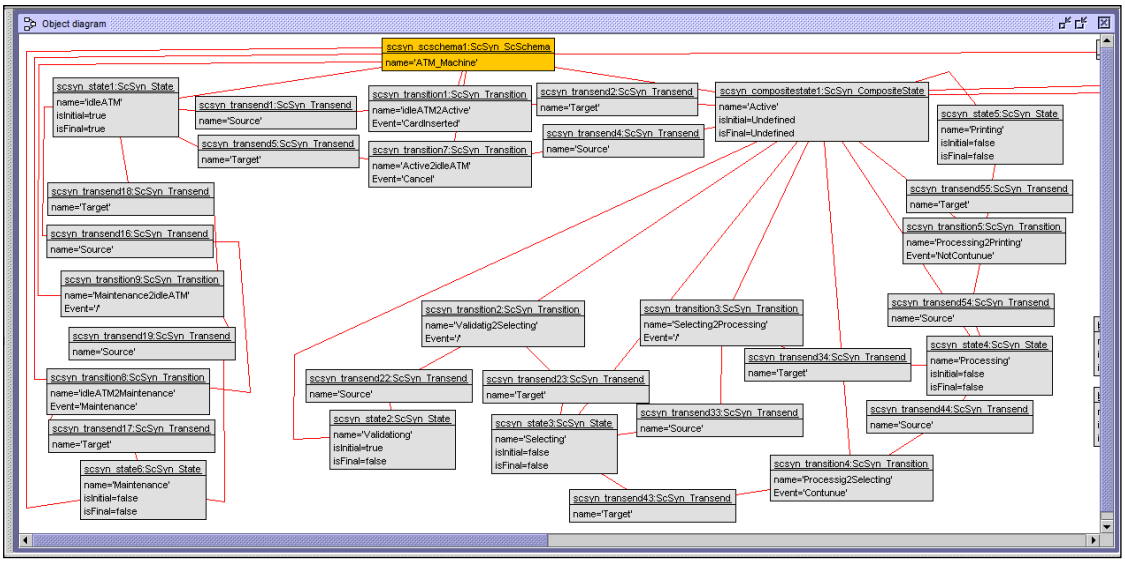

Figure 11: Generated proof example for weak consistency, class , class and association instantiability (ScSny_SsShema part).
Source: Authors, (2026).

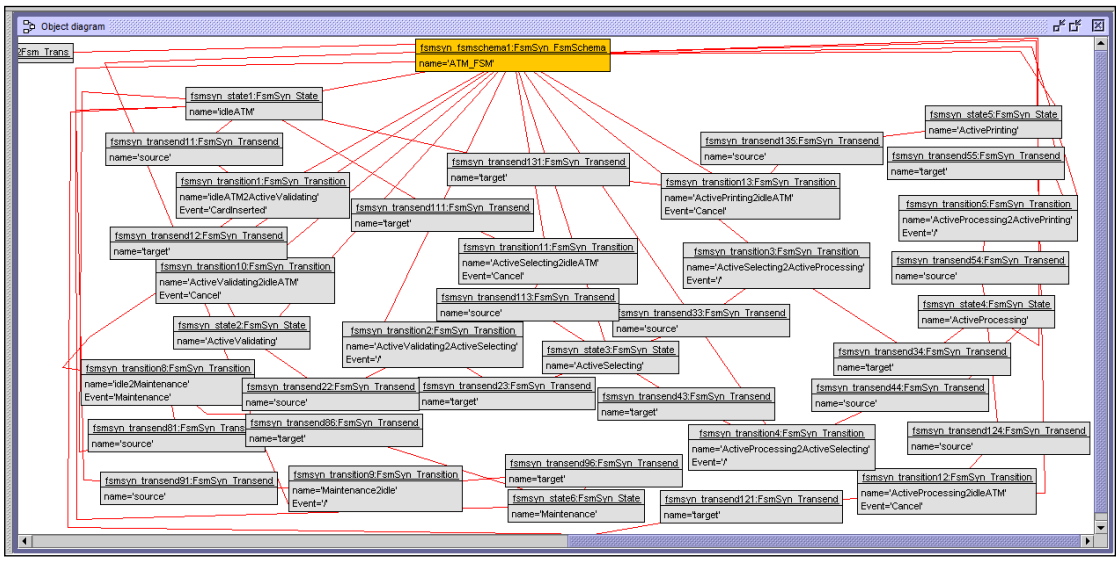

Figure 12: Generated proof example for weak consistency, class, class and association instantiability (FsmSyn_FsmSchema part).
Source: Authors, (2026).

Taken together with the instantiated Statechart and Flat State Machine schemata, these object diagrams demonstrate that the transformation model satisfies the required consistency properties defined in the USE framework, namely weak consistency, class instantiability, and class and association instantiability. In particular, all classes and associations of the transformation model are populated with at least one instance, confirming that the specified constraints are non-contradictory and jointly satisfiable.

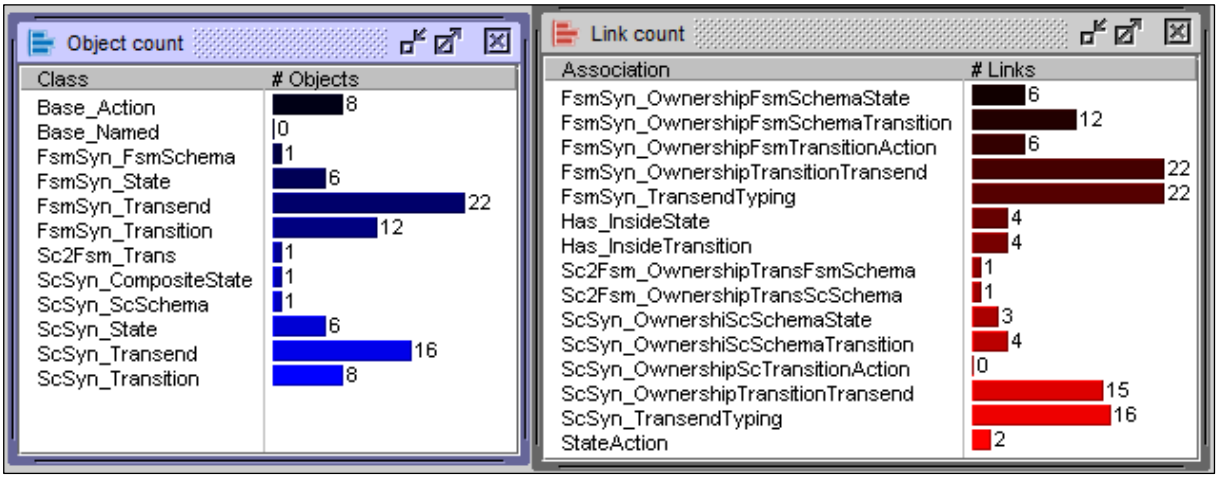

Figure 13: Check class and association population.
Source: Authors, (2026).

To further support this analysis, the Object count and Link count views provided by USE (Figure 13) were used to inspect class and association populations in a concise manner. For instance, the generated instance model contains six FSM State objects, corresponding directly to the six State objects of the source Statechart model, thereby illustrating the structural correspondence enforced by the transformation. Moreover, no instances of the abstract superclass Base_Named were generated, which is consistent with its role as a common abstraction inherited by concrete classes rather than as an instantiable modeling element. Overall, this USE-based validation confirms that the proposed transformation model is consistent and structurally well-founded, and that it faithfully captures the intended relationships between UML statechart models and Flat State Machine models.

VI. CONCLUSIONS

This paper presented a formal approach for validating a model transformation by representing it as a transformation model. Specifically, we modeled the transformation from UML statechart diagrams to Flat State Machines using a UML class diagram enriched with OCL invariants. The proposed transformation model explicitly captures both the structural and semantic relationships between the source and target models. By applying the USE model validator, we automatically verified the consistency of the transformation through the construction of valid object diagrams, thereby demonstrating weak consistency, class instantiability, and class and association instantiability. As future work, we plan to extend this analysis to additional transformation properties, including invariant independence, invariant consequence, termination, and confluence. Investigating these properties will further strengthen the formal guarantees provided by the transformation model and support more robust and scalable model transformation validation.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Ali Lalouci.
Methodology:. Ali Lalouci.
Investigation: Ali Lalouci.
Discussion of results: Ali Lalouci.
Writing – Original Draft: Ali Lalouci.
Writing – Review and Editing: Ali Lalouci.
Resources: Ali Lalouci.
Supervision: Ali Lalouci.
Approval of the final text: Ali Lalouci.

VIII. REFERENCES

[1] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (MBSE): evidence from the literature", Syst. Eng., vol. 24, no. 1, pp. 51–66, 2021.

[2] S. Rädler, L. Berardinelli, K. Winter, A. Rahimi, and S. Rinderle-Ma, "Bridging MDE and AI: a systematic review of domain-specific languages and model-driven practices in AI software systems engineering.", Software and Systems Modeling, vol. 24, no.2, pp. 445-469, 2025.

[3] J.Bezivin, , F. Buttner, M. Gogolla, F. Jouault, , I. Kurtev, and A. Lindow, "Model Transformations? Transformation Models! ", In :Nierstrasz, O., Whittle, J., Harel, D.,Reggio, G., eds.: Proc. 9th Int. Conf. Model Driven Engineering Languages andSystems (MoDELS'2006), LNCS, vol. 4199, pp. 440–453, 2006.

[4] M. Amrani, B. Combemale, L. Lúcio, M. K. Selim, J.,Dingel, J.,Traon, Y.L., Vangheluwe, H., and R. Cordy, J., "Formal Verification Techniques for Model Transformations: A Tridimensional Classification", Journal of Object Technology, vol. 14, no. 3, pp. 1-43, August, 2015, doi:10.5381/jot.2015.14.3.a1.

[5] E. Kerkouche, A. Chaoui, E.B. Bourennane, and O. Labbani, "A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation", Journal of Object Technology, vol. 9, no. 4, pp. 25-43, July, 2010.

[6] E. Kerkouche, A. Chaoui, E.B. Bourennane, and O. Labbani, "On the Use of Graph Transformation in the Modeling and Verification of Dynamic Behavior in UML Models", Journal of Software, Vol. 5, no. 11, pp. 1279-1291, Nov, 2010.

[7] L. Vidal, F. Benito, J.E. Fernandes, "Feasibility Study of the Use of Petri Nets in the Verification of UML Diagrams.", In: Rocha, Á., Fajardo-Toro, C.H., Riola, J.M. (eds) Developments and Advances in Defense and Security. Smart Innovation, Systems and Technologies, vol 328, 2023. https://doi.org/10.1007/978-981-19-7689-6_3.

[8] M. KEZAI and A. KHABABA, "Generating Maude specifications from M-UML Statechart diagrams.", Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 26, no 1, p. 8-16, 2022.

[9] Y. Rahmoune and A. Chaoui, "Automatic bridge between BPMN models and UML activity diagrams based on graph transformation.", Computer Science, vol. 23, 2022.

[10] H. Hachichi, "A graph transformation approach for modeling UML diagrams.", International Journal of Systems and Service-Oriented Engineering (IJSSOE), vol. 12, no 1, p. 1-17, 2022.

[11] F. Büttner and H. Bauerdick, "Realizing UML model transformations with USE.", In : *UML/MoDELS Workshop on OCL (OCLApps' 2006)*, p. 96-110, 2006.

[12] M. Gogolla, L. Hamann, and F. Hilken, "Checking Transformation Model Properties with a UML and OCL Model Validator.", In: Amrani, M., Syriani, E.,Wimmer, M. editors, Proc. 3rd Int. Workshop on Verification of Model Transformation (VOLT'2014), Vol. 1325, pp. 16-25, 2014.

[13] M. Gogolla, "Tales of ER and RE Syntax and Semantics.", In Cordy, J.R., L¨ammel,R., Winter, A., eds.: Transformation Techniques in Software Engineering, IBFI, SchlossDagstuhl, Dagstuhl Seminar Proceedings vol. 51, pp 05161, 2005.

[14] M. Gogolla And F. Hilken, "UML and OCL Transformation Model Analysis: Checking Invariant Independence.", In : *VOLT@ STAF*, p. 20-27, 2015.

[15] M. Al Lail, A. Viesca, H. Cardenas, et al., " Tpv: A tool for validating temporal properties in uml class diagrams.", In : Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, p. 114-118, 2024.

[16] Thi-Hanh. Nguyen And Duc-Hanh. Dang, "Tc4mt: A specification-driven testing framework for model transformations.", International Journal of Software Engineering and Knowledge Engineering, vol. 33, no 06, p. 953-991, 2023.

[17] F. Hilken and H. Lars, "History of the USE Tool 20 Years of UML/OCL Modeling Made in Germany." J. Object Technol., vol. 19, n. 3, pp. 3-1, 2020.

[18] G. Booch, I. Rumbaugh, and J. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1999.