The Journal of Systems and Software 230 (2025) 112508

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Model transformation and property preservation in rigorous software development: A systematic literature review☆

Gullelala Jadoon [a,b,d,*], Maurice H. ter Beek [b], Alessio Ferrari [c,b]

[a] Department of Information Engineering, University of Florence, Italy
[b] FMT Lab, CNR-ISTI, Pisa, Italy
[c] University College Dublin (UCD), Dublin, Ireland
[d] Department of Information Technology, University of Haripur, Pakistan

ARTICLE INFO

Dataset link: https://zenodo.org/records/15227409

Keywords:
Model transformation
Software modeling
Property preservation
Non-functional properties
Survey study

ABSTRACT

Rigorous software development involves using highly structured methods and processes in software and system engineering to ensure that the developed products are correct, reliable, and robust. In this context, model-driven development (MDD) has emerged as a development paradigm that emphasizes designing software systems by means of graphical or textual models at different levels of abstraction, which capture different aspects or dimensions of the system-to-be. At the core of MDD is *model transformation*, which is the process of translating one model into another, according to specific rules. *Property preservation* in MDD refers to maintaining specific properties of the system model during transformations, including structural, behavioral, and domain-specific constraints.

Over the past decades, research on model transformation and property preservation has seen several contributions. In this paper, we present a systematic literature review (SLR) to compile information on study demographics, model properties considered, techniques to ensure property preservation, and other aspects. In addition, through thematic analysis, we highlight significant challenges and benefits associated with model transformation and property preservation.

We analyze 202 research studies published between 2000 and 2024. Most of the studies concern case studies (62, 31%) and rigorous analysis (49, 24%), while experimental studies using human subjects are limited (1). Formal logic is the most commonly used transformation language, used in 42 studies (21%), while the Unified Modeling Language (UML) is also used for source (58, 29%) and target (25, 12%) modeling. A total of 100 of the studies (50%) performed system testing on models, while 44 of the studies (22%) used transformation rules to verify transformation properties. Among the verified model properties, 66 studies (33%) focused on consistency management, while 4 (2%) are related to model maintainability and reuse.

We conclude from our SLR that property preservation could be improved by using model-specific verification methods and strategies based on the considered model artifacts. Our research also provides a relevant contribution by identifying the major challenges in MDD and proposing relevant solutions.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

Contents

☆ Editor: Christoph Treude.

* Corresponding author at: FMT Lab, CNR-ISTI, Pisa, Italy.
E-mail addresses: gullelala.jadoon@isti.cnr.it (G. Jadoon), maurice.terbeek@isti.cnr.it (M.H. ter Beek), alessio.ferrari@ucd.ie, alessio.ferrari@isti.cnr.it (A. Ferrari).

https://doi.org/10.1016/j.jss.2025.112508
Received 26 September 2024; Received in revised form 18 April 2025; Accepted 21 May 2025
Available online 16 June 2025

1. Introduction

In software and system development, models serve as abstractions, conveying the fundamental aspects of a problem or solution while managing complexity (Sommerville, 2015). Model-driven development (MDD) has emerged as a development paradigm in which models are at the center of the process. Models enable reasoning on different system aspects, and ultimately facilitate automated code generation and verification, while ensuring both inter-model and intra-model consistency between different models and within a model by means of rigorous model transformations. A necessary aspect of model transformations lies in specifying the characteristics of the source models and target models—i.e., the input and output of the transformation—, achieved by defining the meta-models to which the models must conform. A meta-model is a model that describes the structure, constraints, and semantics of the models within a specific domain, as shown in Fig. 1 inspired by Czarnecki and Helsen (2006). It describes the types of elements, relationships, and properties that can exist in the considered models. They also encapsulate transformation semantics, establishing how models can be changed or manipulated. Essentially, meta-models ensure that transformations follow the intended rules and meanings of the models involved in MDD (Magalhães et al., 2020b; Arrassen et al., 2012; Meyma et al., 2023). The focus of this SLR is on property preservation and synchronization across models, which is essential when managing multiple representations of a system, such as design models, analysis models, and code models. Property preservation in MDD means ensuring that certain important characteristics of a model, such as its structure, behavior, or domain-specific features, remain unchanged and consistent throughout the different stages of model transformation (Tariq and Cheema, 2021).

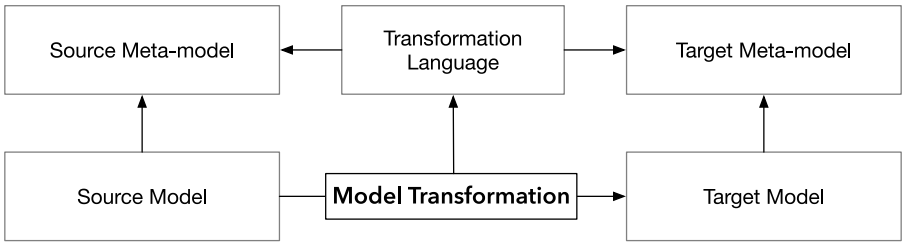

Fig. 1. Model Transformation (inspired by Czarnecki and Helsen, 2006).

Modeling languages, like the Unified Modeling Language (UML) and SysML (Systems Modeling Language), provide the syntax and semantics for expressing system designs, while transformation languages, like the Atlas Transformation Language (ATL) (Cuadrado et al., 2022) and Query/View/Transformation (QVT) (Arrassen et al., 2012), automate the process of transforming models from one form to another. To ensure that model properties are preserved throughout the transformation process, i.e., that these transformations align with the intended design without introducing errors by maintaining consistency across transformations, different techniques are applied (Ciancone et al., 2010; Hutchesson and McDermid, 2011; Giraldo Velásquez, 2017; Meng et al., 2021; Dyck et al., 2018), which typically go under the name of property preservation techniques. Property preservation is essential for upholding correctness, as properties like functional behavior, safety, and security remain intact during transformations, reducing the likelihood of defects. Additionally, property preservation facilitates verification and validation by providing a consistent foundation for assessing model quality. It enhances system reliability by ensuring that non-functional requirements such as performance and scalability are preserved, while also improving modularity and reusability by enabling components to be reused without extensive re-verification. Automation, a key aspect of MDD supported by robust transformation languages, relies on property preservation to deliver predictable and reliable results, making development processes more efficient. Furthermore, property preservation aids in managing the complexity of large systems, ensures traceability by linking implementation changes to high-level requirements, and mitigates risks associated with faults, inconsistencies, or non-compliance. Various property preservation techniques exist, such as logical inference, model checking, manual testing, static analysis (Cuadrado et al., 2022; Mottu et al., 2012; Alshalalfah et al., 2023), white/black box techniques (Schonbock et al., 2013), machine-learning approaches (Karimi et al., 2024; Sick et al., 2021; Jadhav et al., 2024; Jadoon et al., 2024), and verification of construction properties (He et al., 2016). However, a comprehensive study that provides critical analysis and a complete overview of the topic is lacking (Gogolla and Vallecillo, 2011; Kessentini et al., 2011; Baudry et al., 2010).

The study presented in this paper fills this gap, offering a systematic literature review (SLR) of 202 studies published between 2000 and 2024. The paper analyzes different dimensions of the primary studies considered, namely: the demographic of the studies—in terms of year, venue, and empirical evaluation—and also including languages used and transformation types; the properties considered; and the property preservation techniques proposed with associated tools. Furthermore, the paper includes an in-depth thematic analysis of the paper to identify the benefits and challenges of performing model transformation and ensuring property preservation.

The takeaway message concludes that model transformation specifications are as crucial as the models, yet they often lack the formal analysis needed to ensure reliability and accuracy. Effective verification of model transformations through static analysis or logical theorem proving is essential, but challenges with clarity and rigidity persist. Furthermore, while simulations and regression testing offer ways to validate transformations, a comprehensive framework for property preservation remains necessary. As models grow in complexity, they become more susceptible to bugs, emphasizing the need for robust, automated tools—particularly those tailored to domain-specific contexts. Although solutions like variability management and feature modeling show potential, they need further exploration, especially in terms of scalability, performance, and privacy. Ultimately, while the benefits of property preservation—such as consistency, reliability, and traceability—are recognized, achieving these outcomes demands more focused research and refinement.

This paper provides the following contributions to the literature on MDD:

1. Statistics on the publication venues and type of empirical strategies adopted by the studies, which indicate the field's degree of maturity and, thus, the transferability to practice of the proposed techniques.
2. Statistics on the source, target, and transformation languages considered in the primary studies, together with properties addressed, and associated techniques. This information is useful to point the attention of researchers towards areas that have received limited attention and that require more in-depth studies. The statistics also indicate to practitioners the solutions that are currently more mature, such as the ATL language in the context of declarative model transformation.
3. A list of support tools, which can be used by practitioners and researchers to perform case studies in industrial environments.
4. A set of benefits of model transformation of property preservation, which can trigger practitioners to experiment with these techniques in their companies.
5. A set of challenges that indicate areas of improvement that need to be addressed by researchers.

The rest of the paper is composed as follows. Section 2 presents some necessary background on modeling and transformation languages and properties, as well as property preservation. Section 3 discusses related reviews of model transformation and property preservation techniques. Section 4 describes the review methodology. Section 5 presents the results and Section 6 discusses the statistical findings. Section 7 reports threats to validity. Section 8 concludes the paper. An appendix complements Section 5.

2. Background

Model transformations are typically defined by syntactic and semantic rules for transforming models expressed in a certain transformation language. It refers to the automated process of converting one model into another within the same domain or across domains while preserving the intended semantics and specific properties.

Bidirectional transformations, comprising forward and backward transformations, are fundamental for maintaining consistency and synchronization between source and target models. Forward transformations derive target models from source models, aligning system views or implementation details. Conversely, backward transformations propagate changes from target to source models, preserving alignment and preventing inconsistencies.

Property preservation is fundamental to MDD, playing a vital role in maintaining the integrity, reliability, and usability of models. Techniques such as logical inference, model checking, and static analysis are essential for maintaining critical model properties throughout the transformation (Ciancone et al., 2010; Hutchesson and McDermid, 2011; Giraldo Velásquez, 2017). These approaches collectively form the foundation of property preservation methods, addressing key challenges in transformation quality.

Model transformations have a broad range of application areas in software and systems engineering like reverse engineering, digital twins, simulation, etc. One application area where model transformations are vital is Software Product Lines (SPLs) (Apel et al., 2013). SPLs represent a systematic approach to managing a family of related software systems, enabling the efficient development of configurable products by leveraging shared core assets. Variability modeling, a key aspect of SPLs, captures the commonalities and variabilities among product families, facilitating the specification, configuration, and customization of individual products (ter Beek et al., 2019; Benavides et al., 2025). Property preservation within variability modeling ensures functional correctness, adherence to non-functional constraints, and behavioral consistency during product derivation (Jadoon, 2024). These attributes are essential for delivering reliable and scalable systems, ensuring the derived products meet desired quality and performance standards.

Modeling language and transformation language. A modeling language and a transformation language serve distinct but complementary roles in MDD. A modeling language is primarily used to define and describe models, which are abstractions representing systems or processes. It focuses on specifying the structure, behavior, and interactions of a system at various levels of abstraction. Examples include UML, SysML, and BPMN (Business Process Model and Notation). These languages provide the foundation for creating source and target models essential for system design, analysis, and documentation.

In contrast, a transformation language is specifically designed to define and execute transformations between models. Its main purpose is to convert one model into another while ensuring properties like consistency, correctness, or scalability. Transformation languages specify the rules and mappings required to automate these conversions, which may involve changes to the language, structure, or abstraction level of the models. Examples include ATL, QVT, and ETL (Epsilon Transformation Language).

On the other hand, a Domain-Specific Language (DSL) is a high-level programming language tailored to express the concepts and constructs of a particular domain or application area (Fowler, 2010; Wąsowski and Berger, 2023). It provides abstractions and syntax specific to the domain, making it easier to develop solutions in that domain. Think of SQL for databases or Simulink for control systems.

A model transformation language is indeed a type of DSL because it is designed with a limited scope, specifically targeting the domain of transforming models. A transformation language supports the significance of domain-specific tools in facilitating precise and reusable multi-model transformations, by defining system properties and semantics for multiple source and target models, often involving heterogeneity and complex interdependencies. A domain-specific tool refers to a software application explicitly developed to assist certain tasks within a defined domain or problem space. These tools are crafted to address the distinctive requirements and complexities inherent in a particular application area, ensuring consistency for modeling, transformation, and analysis tasks relevant to that domain (Macías et al., 2019). Consistency, in this context, refers to the property of maintaining coherence and correctness within a model and across different models within the transformation development process (Guerra and Soeken, 2013; Uzun and Tekinerdogan, 2018; Tsigkanos et al., 2020). It involves ensuring that the transformed models accurately reflect the intended behavior and structure specified by the original models, thus preserving inter-model and intra-model consistency throughout the transformation process (Oakes et al., 2015).

Transformation properties. The properties considered span from language-related properties (Rodriguez-Echeverria et al., 2021) (e.g., termination, determinism) to transformation-related properties (e.g., source/target model conformity, syntactic and semantic relation, functional behavior) (Combemale et al., 2007; Ko et al., 2013; Sahin et al., 2015). Similar to model transformation, property preservation requires advanced verification tools to detect and address both functional and non-functional issues that could have occurred in the transformation (Ciancone et al., 2010; Hutchesson and McDermid, 2011; Giraldo Velásquez, 2017).

Our classification of focused properties is based on an extensive review of the literature, where the primary objectives, challenges, and evaluation criteria of model transformations were analyzed. Each study was examined for its emphasis on key transformation aspects, such as ensuring correctness, managing consistency, and optimizing performance. For example, properties like consistency management were classified based on studies addressing the alignment and synchronization between source and target models. Transformation verification was categorized from works focused on validating the correctness and accuracy of transformation processes, often using formal methods or testing frameworks. Non-functional behavior was identified in studies prioritizing quality attributes like scalability, reliability, or performance, while syntax and semantics reflected efforts to ensure structural and interpretative accuracy in transformations. Functional and non-functional behavior was classified from studies that explicitly combined both aspects for holistic evaluation. Finally, categories like maintainability and reuse emerged from research emphasizing the long-term usability and adaptability of transformation artifacts. This categorization ensures that the focused properties align with the explicit goals and methodologies detailed in the reviewed studies.

3. Related reviews

MDD has been an area of interest for researchers over the past two decades. Sommerville (2015) provides a detailed introductory review of MDD focusing on model quality attributes. We collected related reviews in the field of model transformation and property preservation, which is the focus of our research. In the following, we summarize the content of these reviews and illustrate our contribution.

The existing reviews can be divided into four main groups. The focuses of the different groups are: (1) rigorous model transformation approaches; (2) support tools; (3) specific applications, such as IoT or mobile ones; and (4) security aspects.

Rigorous model transformation. Troya et al. (2022) surveyed 140 papers on model transformation in MDD, highlighting challenges in model transformation testing and debugging and presenting a comprehensive overview of current trends, advances, and open research challenges. Troya et al. (2022) reviewed the state of the art in testing and debugging model transformations. Analyzing 140 papers, they explored trends, advances, and the evolution of these research areas. They also introduced a conceptual framework to categorize various approaches, and highlighted significant research challenges. Batot et al. (2016) explored the transformation of concrete models in MDD, with a focus on new directions in the modeling of domain-specific systems. According to them, concrete models mainly emphasize the code of transformed models instead of the abstract meta-models used before code generation.

Support tools. Kahani et al. (2019) presented a catalog of 60 model transformation tools, organizing them based on transformation approaches and comparing them according to six categories. Their study provides valuable details on the results and insights for stakeholders in the modeling community, including practitioners, researchers, and tool developers. Rashid et al. (2015) presented a comprehensive study on tool analysis, which discussed recent approaches to developing and verifying embedded systems using model-based software engineering; 39 tools are identified that provide support for all phases of development in MDD. Madlener et al. (2010) also focused on the same subject using transformation rules.

Application-specific. Ameller et al. (2015) presented a survey in which characteristics of MDD are discussed that can be applied to the development of service-oriented systems. Ameller et al. (2021) also presented an interview-based survey on how to manage non-functional requirements in MDD, considering 18 IT companies from 6 countries. They concluded that more advanced tools and techniques are needed to improve the management of non-functional requirements like productivity and maintainability. A systematic mapping study by Santos et al. (2015) revealed deficiencies in current consistency management practices in SPL engineering, including tool support, tracking, evaluation, and validation mechanisms. Although some studies address property management for identified inconsistencies, there is a lack of discussion about the handling and management of these inconsistencies. Shamsujjoha et al. (2021) conducted a literature review focusing on MDD for mobile applications, noting that MDD is becoming essential to manage the complexity of developing application models dependent on devices, platforms, and system requirements. They also observed that some property preservation mechanisms are more responsive and ad-hoc for specific applications, yet not generic enough to handle heterogeneity. Erazo-Garzón et al. (2022) conducted a systematic review of 51 relevant articles on model-driven user interface engineering, studying how MDD is utilized to build and maintain software user interfaces, addressing key research questions and identifying challenges, gaps, and research opportunities. Szvetits and Zdun (2016) performed an SLR on MDD, focusing on the design of models at run-time and addressing changing requirements, accuracy, error handling, and human involvement. Neis et al. (2019) systematically reviewed tools to design and develop power systems, concluding that MDD approaches are well suited to develop these systems, as step-by-step transformation is necessary to build verified executable systems.

Security aspects. Finally, some of the reviews focus on security aspects in MDD. In 2015, Nguyen et al. (2015) performed a systematic review of MDD for secure software systems. They provided an overview of the state-of-the-art in model-driven security and identifies challenges in related model transformations. They highlighted that domain-specific systems are less prone to security issues than generic applications due to predefined semantics and suggests future directions for security solutions. Geismann and Bodden (2020) performed an SLR targeting security in IoT-based models of cyber–physical systems. These systems are vulnerable to various cyber-attacks that compromise data security and confidentiality. Their review discussed 17 approaches and analyzed their limitations, noting that MDD can help identify such attacks in the early development phases.

Comparison and contribution. Compared to the reviews on rigorous model transformation, the review presented in this paper has a broader focus, as it considers a larger set of property preservation techniques, and not solely transformation testing and debugging, as Javier et al. and Troya et al. did. In addition, we provide an in-depth thematic analysis of the primary studies to illustrate the benefits and challenges in the field. Batot et al. also had a narrower—yet relevant—scope, as they focused on the practical dimension of model transformation. Similar to Kahani et al. and Rashid et al. we also survey existing tools. However, compared with Kahani et al. we consider only tools that offer property preservation support. Compared to Rashid et al. our review does not exclusively consider embedded systems. Finally, compared to work on application-specific cases and security properties, our scope differs, as we do not focus on single application types or properties. Overall, this is the first study that focuses on model transformation and property preservation in a broad sense, providing a general overview of the field, and outlining research challenges grounded on existing empirical studies.

4. Review method

This section provides insight into the literature review process that follows the guidelines provided by Kitchenham and Charters (2007). In addition to these established guidelines, we also adhered to the PRISMA 2020 (Page et al., 2021) guidelines for SLRs, which offer a comprehensive checklist to enhance the rigor, transparency, and reproducibility of the review process. By integrating these frameworks, we ensured a robust methodological approach to identifying, selecting, and analyzing the relevant literature. The primary search strategy involved systematic keyword-based queries across selected digital libraries using well-defined search strings. The secondary search strategy concerned snowballing—both backward and forward citation tracking—which was conducted to ensure broader coverage and to identify additional studies not captured through the initial search. More in detail, we first identified the research questions (Section 4.1) and composed the search string (Section 4.2) accordingly. This was followed by a proper search and selection strategy (Section 4.3), and by data extraction and synthesis procedures (Section 4.4).

4.1. Research questions

Our research goal is to survey existing empirical studies applying model transformation and property preservation techniques, and identify benefits and challenges of the surveyed approaches. Based on this goal, we define the following research questions (RQs):

RQ 1: What are the demographic aspects of the primary studies in the field of model transformation?

- **RQ 1.1:** What is the yearly distribution?
- **RQ 1.2:** What is the venue distribution?
- **RQ 1.3:** What are the main languages used in the transformation?[1]
- **RQ 1.4:** What are the transformation types considered?
- **RQ 1.5:** Which type of evaluation has been conducted?

RQ 2: What are the model properties considered?

RQ 3: What are the techniques used to ensure that model properties are preserved in the transformation?

RQ 4: What are the different automated tools used to ensure that model properties are preserved in the transformation?

RQ 5: What are the challenges of model transformation and property preservation?

RQ 6: What are the benefits of model transformation and property preservation?

4.2. Databases and search string

The databases we selected for our search are Springer, IEEE Xplore, ACM Digital Library, and Elsevier Science Direct. These databases belong to the main publishers of reputable conferences and journals in software engineering. Google Scholar has also been used for complementary searches.[2] The search string that we used is as follows:

(“model transformation” OR “model synchronization”)
AND

[1] Here we consider transformation languages as well as languages used in the source and target models.

[2] To this end, we analyzed the results solely in the first 25 pages returned by Google Scholar, as further pages did not appear to include relevant papers.

Table 1
Inclusion and exclusion criteria.

	Inclusion Criteria
1	Relevant studies published in journals, conferences, or book chapters.
2	Papers that provide a clear methodology, sufficient detail, and evidence supporting their claims.
3	In similar studies of model transformation and property preservation, the extended version of the article is considered.
4	Studies applying property verification and property preservation in models are considered.
	Exclusion Criteria
1	Duplicate and irrelevant studies.
2	Papers in the form of tutorials or posters are excluded.
3	Publications or sources that lack academic rigor or credibility.
4	Studies where full text is not available.
5	Papers not in the English language.
6	Papers published before the year 2000.

("verif*" OR "propert*")

For Science Direct, we used the following string, as * is not allowed:

("model transformation" OR "model synchronization")

AND

("verification" OR "property preservation")

We decided to use "model transformation" rather than "model*" AND "transform*" because the latter resulted in 3,226,423 papers. Analyzing this number of papers was considered unfeasible. The possible limitations resulting from this choice are mitigated by our secondary search strategy (snowballing, cf. Section 4.3). To guarantee the relevance of the papers while maintaining feasibility, we applied a general filter of publication dates from 2000–2024 across all databases. Additionally, we tailored the search strategy for specific databases to maximize the precision of the results.

In the ACM Digital Library, the search was conducted within the abstract field, as abstracts concisely convey the key contributions and objectives of papers, facilitating the identification of studies closely aligned with our focus. For ScienceDirect, the search was limited to the Title and Abstract fields, targeting papers explicitly addressing model transformation topics while avoiding peripheral mentions. In SpringerLink, additional filters were applied, including restricting results to research articles and conference papers, selecting only papers in the English language, and narrowing the subject areas to "Model Building and Simulation" and "Models of Computation" to maintain a clear alignment with the domain of MDD.

4.3. Search strategy and study selection procedures

We selected the most relevant studies based on the inclusion and exclusion criteria presented in Table 1. The inclusion and exclusion criteria are designed to refine the dataset for the SLR by selecting studies that contribute directly and substantially to the research questions. These criteria balance the breadth and depth of the review, ensuring coverage of relevant topics while maintaining academic rigor and relevance. By adhering to these criteria, the SLR achieves a focused, high-quality, and reliable dataset for comprehensive analysis. The quality checklist we used to evaluate our search results and remove low-quality studies is listed in Table 2.

The selection process consisted of the following stages:

1. Retrieval: Search is performed on the abstract and title, using the described search string in each database. Where necessary for more detail, a full-text search is performed as well.

2. Screening: Read the title and abstract of each study, and following the inclusion criteria, each study is added or discarded in the Excel sheet.

3. Full-text Reading: Read the full text of the study and follow the exclusion criteria and quality checklist. Discard irrelevant studies. A ternary scale (Yes=1, Partial=0.5, and No=0) is used to check each research study against each question of the quality checklist. So, the quality score of each research study is calculated, and if the total score is equal to or greater than 6 (out of 10), then the study is included; otherwise, it is excluded from the SLR. The selected papers are then assigned a unique ID for data extraction.

The secondary search strategy consisted of forward and backward snowballing, in which the references of any selected study from the primary search were screened and selected.

After pilot searches, the search string was defined, and the primary search was performed on 2 April 2025.[3] The search interval was 2000–2024. The additional search (snowballing) was performed on the studies extracted after the primary examination. A total of 21,942 studies were retrieved during the initial search, of which 295 studies were included.[4] Upon a thorough quality check, 120 studies were discarded. In addition to the remaining 175 studies selected, a further 27 studies were selected by snowballing as part of the additional search. This also led to the inclusion of studies from additional venues such as ECMFA. Duplicates were discarded during the database searches, which were performed incrementally (ACM Digital Library, IEEE eXplore, Science Direct, Springer Link, Google Scholar), as well as during snowballing. The included studies already account for the discarded duplicates. We did not keep track of the number of duplicates, but we believe this does not impact the soundness of the overall search and selection protocol. Fig. 2 provides more details on the number of studies. The search, paper selection, and data extraction for the SLR were conducted by the first author, with the second and third authors reviewing and validating the selection process and extracted data.

4.4. Data extraction and synthesis procedures

Data extraction was performed according to a set of classification schemes, which will be illustrated within the specific sections of the results section (Section 5). All the information has been recorded by the first author in a Spreadsheet, after piloting the schemes together with the other two authors on a subset of the papers retrieved during the pilot searches. The Spreadsheet was used to collect statistical information from the classified data, which is used to answer our research questions RQ 1–RQ 4. For RQ 5 (challenges) and RQ 6 (benefits), we performed thematic analysis (Braun and Clarke, 2021; Bano and Zowghi, 2015) to classify the benefits and challenges of model transformation and property preservation. We created another Spreadsheet categorizing the main themes of benefits and challenges. All the Spreadsheets produced are available in our permanent repository (Jadoon, 2024–2025).

Based on the Spreadsheets associated to the classification schemes, we provide visual analytics using pivot tables and graphical diagrams to represent different aspects of the extracted data entered by the extractor, such as transformation type, transformation language, source and target languages used, properties, etc. The other authors reviewed and crosschecked the data.

5. Results

5.1. RQ 1.1: Studies by year

Our analysis begins by exploring the temporal distribution of studies that span from 2000 to 2024, highlighting the evolutionary trajectory of the field.

Fig. 3 answers RQ 1.1, illustrating the annual distribution of relevant studies. It reveals a notable increase in research and application from 2010 onwards, indicating that the topic has seen a steady interest across the years.

[3] The search string was updated for this revision and the numbers reported here concern a search performed on 2 April 2025.

[4] Please recall that of the 20,700 studies returned by Google Scholar, we only analyzed the first 25 pages (cf. Footnote 2).

Table 2
Quality checklist.

	Quality checklist	Scale
1	Are the aims clearly stated?	Yes/No/Partial
2	Are the proposed techniques adequately described?	Yes/No/Partial
3	Was the study design appropriate concerning the research aims?	Yes/No/Partial
4	Are the data collection methods adequately described?	Yes/No/Partial
5	Are the statistical methods justified by the author?	Yes/No/Partial
6	Are negative findings presented?	Yes/No/Partial
7	Are all the study questions answered?	Yes/No/Partial
8	Do the researchers explain future implications?	Yes/No/Partial
9	Is the study published in a high-quality venue?	Yes/No/Partial
10	Are statistical/analytical methods properly used to describe the data?	Yes/No/Partial
11	Are considerations for formal or theoretical approaches included?	Yes/No/Partial
12	Does the study account for practical implementation challenges?	Yes/No/Partial

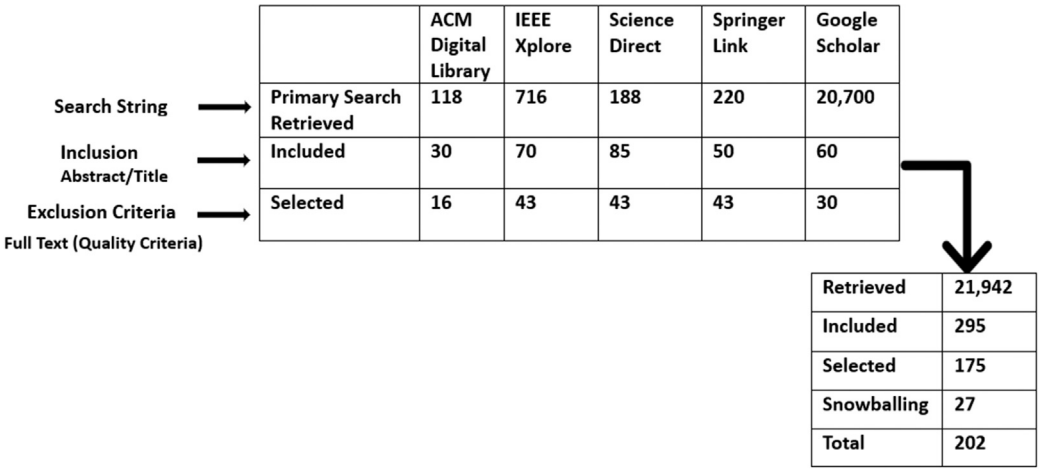

Fig. 2. Numerical results of the database search and snowballing.

Recent spikes in activity, especially evident in 2019 and 2021, can be attributed to influential conferences such as MoDELS (Ege and Tichy, 2019; Klare and Gleitze, 2019) and significant publications in high-impact journals such as IST (Information and Software Technology) (Magalhaes et al., 2019), TSE (IEEE Transactions on Software Engineering) (Ameller et al., 2021), and SoSyM (Software and Systems Modeling) (Kahani et al., 2019), suggesting that the technologies developed in the earlier years had acquired the maturity needed to boost more interest and solid publications.

In addition, the integration of MDD into the development of mobile applications (Shamsujjoha et al., 2021) and IoT-based systems (Geismann and Bodden, 2020) further fueled research interest, contributing to the peaks observed. This trend stresses the growing recognition of the field to streamline the development of emerging technologies.

5.2. RQ 1.2: Venue

Fig. 4 presents a breakdown of the publication venues, illustrating a slight trend towards conference presentations, which account for 105 (52%) of all studies. This highlights the dynamic and rapidly evolving nature of the field, as conferences typically facilitate the immediate dissemination and discussion of cutting-edge research. However, journals also play a crucial role, contributing 93 (46%) of the publications and providing rigorous peer-reviewed platforms for in-depth research articles. Book articles have also been included in the review process.

In particular, SoSyM emerged as a significant publication venue, hosting 17 (18%) of the journal articles. This reflects its status as a leading forum for scholarly discourse in model transformation and MDD in general.

In addition to conferences and journals, book chapters constitute a smaller portion of the literature, comprising only 4 (2%) of the total. However, the presence of book chapters indicates that the topic is sufficiently consolidated in the MDD field, suggesting it has matured enough to warrant inclusion in comprehensive texts that provide overviews, detailed discussions, or specialized knowledge on the subject.

Among the conferences, MoDELS stands out significantly, contributing 15 (14%) of the research studies. This conference shapes the discourse and direction of research in MDD, including research on practical and theoretical aspects of model transformation and property preservation. The acronyms of journals and conferences are shown in Tables 3 and 4 for information, extracted according to Table 5.

The distribution and contribution by publication venues allows researchers and practitioners to better identify relevant sources of knowledge in the field, aligning their work with the most influential forums, and capitalizing on the collective expertise and insights shared within these communities.

5.3. RQ 1.3: Transformation language

Fig. 5 illustrates the frequencies of the transformation languages. It indicates that many studies do not explicitly identify the transformation

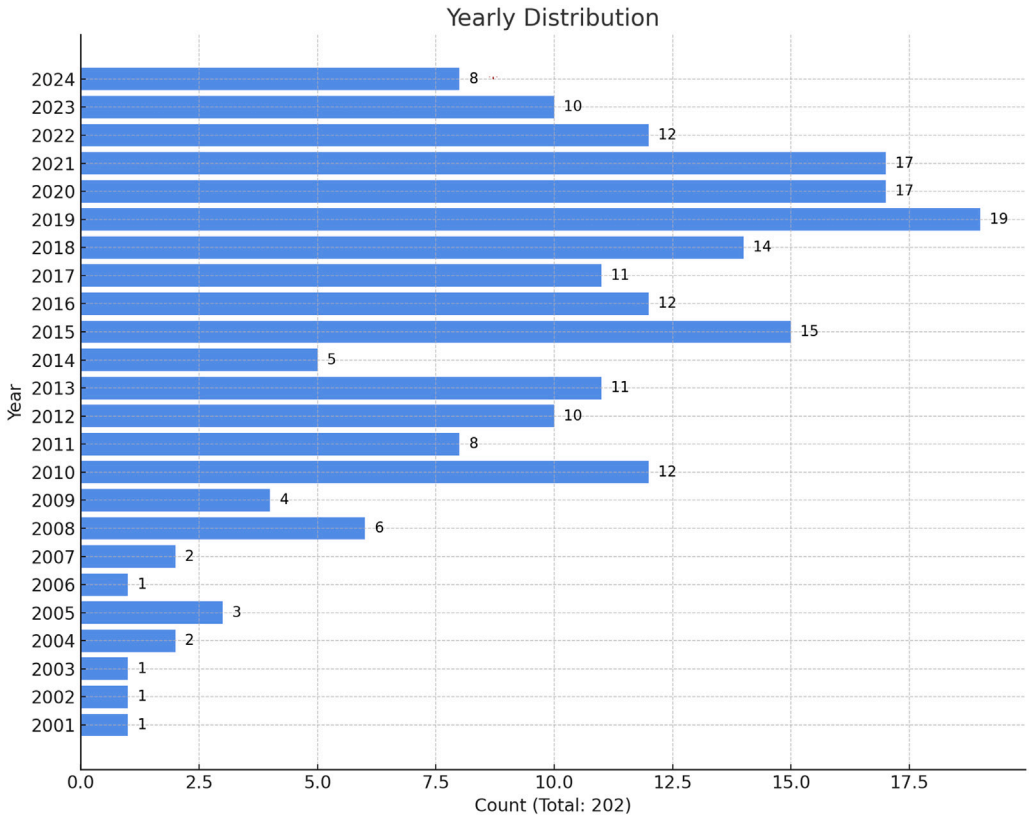

Fig. 3. Distribution of studies by year.

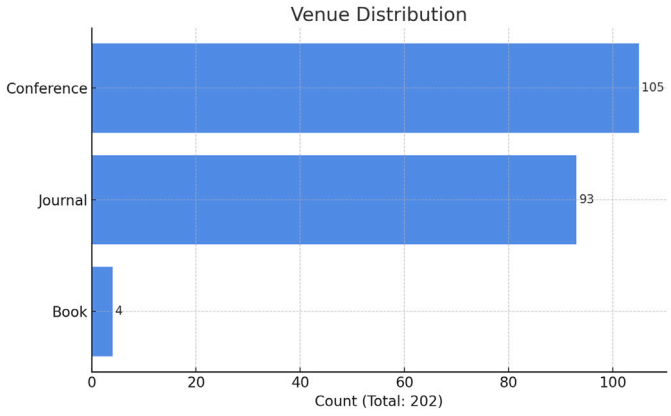

Fig. 4. Statistics on publication venues.

Table 3
Journal acronyms.

Acronym	Journal
SoSyM	Software and Systems Modeling (17)
JLAMP	Journal of Logical and Algebraic Methods in Programming (1)
TOSEM	ACM Transactions on Software Engineering and Methodology (3)
ELS	Engineering Languages and Systems (1)
TSE	IEEE Transactions on Software Engineering (5)
ENTCS	Electronic Notes in Theoretical Computer Science (2)
ECEASST	Electronic Communications of the EASST (1)
JSS	Journal of Systems and Software (8)
FAC	Formal Aspects of Computing (5)
IST	Information and Software Technology (5)

Table 4
Conference acronyms.

Acronym	Conference
MoDELS	Model Driven Engineering Languages and Systems (15)
FASE	Fundamental Approaches to Software Engineering (2)
SEAA	Euromicro Software Engineering and Advanced Applications (2)
ICSE	Int. Conf. on Software Engineering (2)
ICSR	Int. Conf. on Software Reuse (2)
ICEIS	Int. Conf. on Enterprise Information Systems (4)
MODELS-WARD	Model-Based Software and Systems Engineering (3)
IFM	International Conference on Integrated Formal Methods (1)
VaMoS	Variability Modeling and Simulation (2)
ICCIS	Int. Conf. on Computational Intelligence and Security (3)
ECBS	Engineering Computer-Based Systems (1)
ITCC	Information Technology: Coding and Computing (1)
ICMT	Int. Conf. on Model Transformation (1)
ICFEM	Int. Conf. on Formal Engineering Methods (1)
ECMFA	European Conf. on Modeling Foundations and Applications (3)

language used. This lack of specification could suggest either a focus on modeling concepts rather than specific tools or a reliance on custom or implicit approaches to transformation that are not formally defined.

Formal Logic emerges as the second most frequently cited category, with 42 mentions (21%), highlighting its significant role in underpinning model transformation techniques. Its popularity suggests a strong reliance on logical frameworks to ensure precision, correctness, and verifiability in transformation tasks.

ATL (Atlas Transformation Language), cited 23 times (11%), is widely recognized for its declarative approach to model transformation. Its prominence indicates its versatility and efficiency in practical applications, particularly in the context of uni-directional transformations.

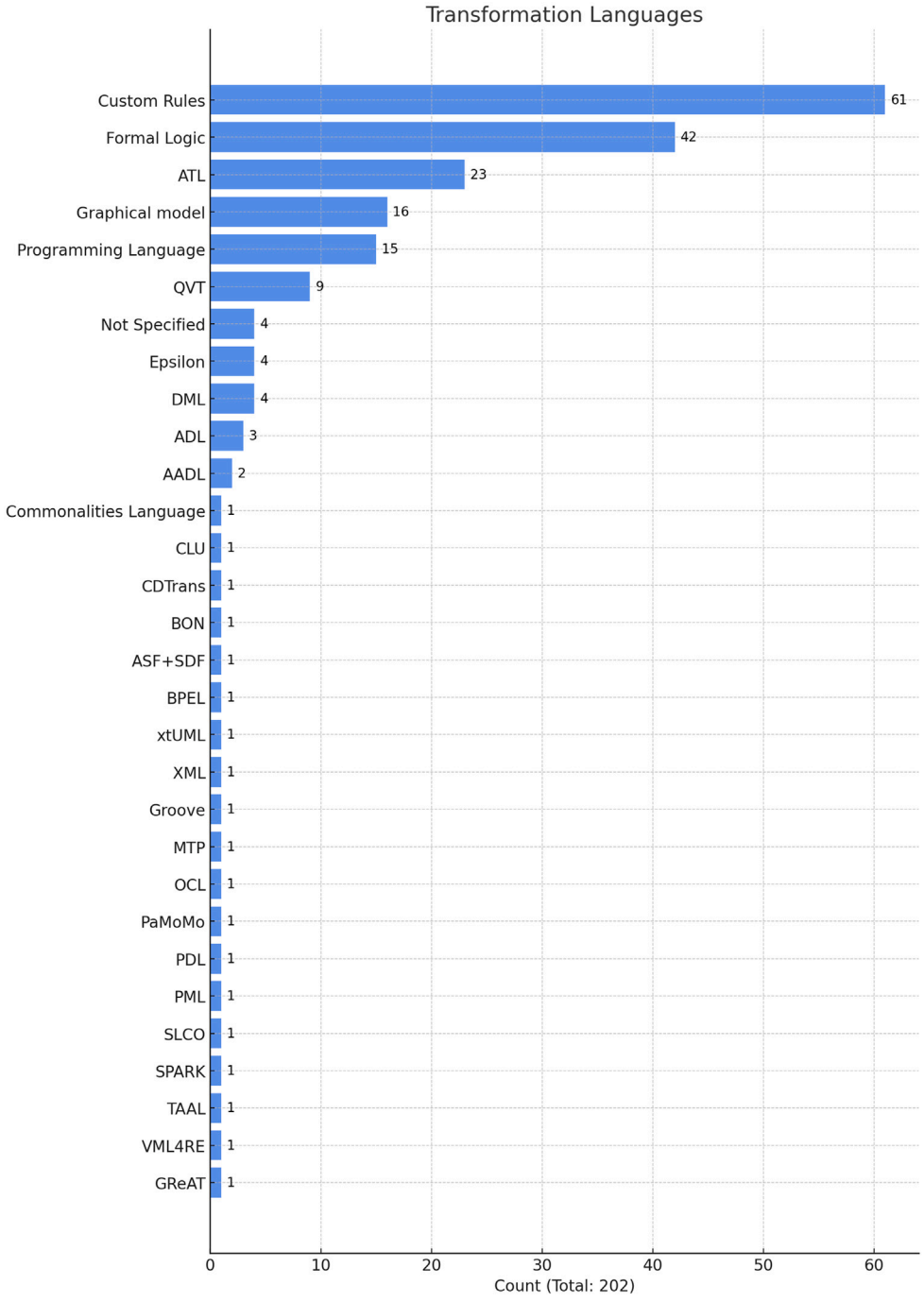

Fig. 5. Transformation languages.

Table 5
Data extraction categories for RQ 1.2.

Venue	RQ 1.2
Conference name	The conference to which a research study belongs.
Journal name	The journal to which a research study belongs.

Graph model is referenced 16 times (8%), reflecting its popularity in visual modeling scenarios. This category highlights the importance of user-friendly, visual representation techniques of attributed graph transformation (AGGs) and triple graph grammars (TGGs) for defining transformations, making it accessible for practitioners working on complex models.

Programming Languages (15 mentions, 7%) and QVT (9 mentions, 4%) represent general-purpose and standard transformation tools, respectively. Meanwhile, Epsilon and DML (Domain Modeling Language), each cited 4 times (2%), show their relevance in niche contexts or specific domains.

Despite being inherently unidirectional and lacking built-in verification mechanisms, ATL still emerged as one of the most widely used model transformation languages in the surveyed literature. This widespread adoption can arguably be attributed to ATL's mature tooling support, extensive documentation, and large user community, which makes it accessible and practical for a wide range of transformation tasks. Moreover, practitioners often address the unidirectional nature of ATL by implementing paired transformations—one for the forward and one for the backward direction—to maintain consistency

Table 6
Data extraction categories related to RQ 1.3.

Modeling Language	RQ 1.3
Source Model Language	Modeling language of the source model like UML (Troya et al., 2018), OCL (Object Constraint Language) (Wijs and Engelen, 2012), AADL (Architecture Analysis and Design Language) (Qiuyan et al., 2011; Wei and Shuyu, 2013), and Programming Languages (Beyer and Lemberger, 2017; Abate et al., 2019).
Target Model Language	Modeling language of the target model like B (Arrassen et al., 2012), BPMN (Business Process Modeling Notation) (Aït Ameur and Aït-Sadoune, 2012; Dechsupa et al., 2019), MTP (Model Transformation Profile) (Magalhães et al., 2016), and Ecore (Greiner et al., 2017).

Table 7
Data extraction categories related to RQ 1.4.

Type of Model Transformation	RQ 1.4
Endogenous Transformation	Model Transformation resulting in dynamic system modeling referring any functional change (Macías et al., 2019; Antignac et al., 2018).
Exogenous Transformation	Model Transformation resulting in static system modeling referring structural change (Troya et al., 2018; Hilken et al., 2018).
Horizontal Transformation	Transformation resulting in high-level abstract system models (Wijs and Engelen, 2012; Magalhães et al., 2016).
Vertical Transformation	Transformation resulting in low-level basic system models (Klare and Gleitze, 2019; Esbai and Erramdani, 2015).
Syntactic Transformation	Transformation resulting in low-level basic system models (Iqbal and Al-Azzoni, 2022; Asztalos et al., 2010).
Semantic Transformation	Transformation resulting in low-level basic system models (Reder and Egyed, 2012; Jilani et al., 2014).

between models. This approach enables ATL to be employed effectively in scenarios that demand synchronization-like behavior, despite not being a native bidirectional language.

In contrast, several other transformation languages and tools offer native support for bidirectional transformations and formal verification, thus aligning with the goals of model synchronization and property preservation (Fritsche et al., 2021). QVT—in particular its Relations variant—supports declarative specifications and bidirectionality, making it suitable for ensuring round-trip consistency. Henshin (Arendt et al., 2010), designed for in-place transformations, supports graphical syntax and leverages critical pair analysis to enable formal verification of transformation rules. eMoflon (Weidmann, 2018) employs TGGs to support bidirectional transformations with formal consistency guarantees. VIATRA (Visual Automated model Transformations) (Mkaouar et al., 2022) offers solver-based approaches for consistency maintenance and has seen significant use in formal verification contexts.

Figs. 6 and 7 further detail the source and target languages used in the transformations, also listed in Table 6. When comparing source and target modeling languages, notable differences emerge. UML dominates the source modeling landscape, with 58 mentions (29%), but is significantly less common as a target modeling language, where DSLs take the lead with 35 mentions (17%). This suggests that UML often serves as a general-purpose input for transformations, while DSLs are favored for generating specialized outputs.Graph model maintains a strong presence in both source (26 mentions, 13%) and target (25 mentions, 12%) modeling, reflecting its versatility and usability across both ends of the transformation process. Similarly, Formal Logic appears prominently in target languages (25 mentions, 12%) but is less common in source models (13 mentions, 6%), highlighting its role in ensuring rigor and correctness at the output stage. The "Not specified" category indicates either endogenous transformations within the same transformation language or the absence of any discussion on the modeling language (Babaei and Dingel, 2023; Cheng and Tisi, 2017). Additionally, there were instances where the main emphasis of the study was on transformation techniques, verification approaches, or semantic mapping strategies, without explicitly stating the source or target modeling languages (Meyma et al., 2023; Batot and Sahraoui, 2016). These patterns underline the complementary nature of these modeling languages, with UML and DSLs frequently shaping source and target models, respectively, and Graphical Syntax bridging both domains.

Fig. 8 shows a model representation graph that reveals key trends in the adoption of various modeling paradigms. UML metamodels dominate with 41 occurrences (20%), emphasizing their role as a standard in modeling due to their versatility and extensive tool support. Graph Models follow with 26 mentions (13%), showing their importance in representing structural relationships and dependencies within models. Class Model ranks third, with 22 mentions (11%), highlighting their frequent use in modeling system behavior and dynamic transitions. Feature Models, cited 11 times (5%), are notable for their application in variability modeling, particularly in SPL. These dominant representations showcase a balance between general-purpose and specialized models, each serving distinct needs, while less frequently mentioned representations cater to specific applications.

5.4. RQ 1.4: Transformation type

Model transformations can be classified into distinct types based on their scope and purpose (Czarnecki and Helsen, 2006; Mens and Van Gorp, 2006) (cf. Table 7).

Endogenous and Exogenous Transformations: *Endogenous* transformations modify models within the same language, focusing on refinement or improvement (Macías et al., 2019; Antignac et al., 2018; El-Sharkawy et al., 2019; Bachmann and Clements, 2005). *Exogenous* transformations, on the other hand, translate models between different languages, enabling interoperability across tools and domains (Troya et al., 2018; Hilken et al., 2018).

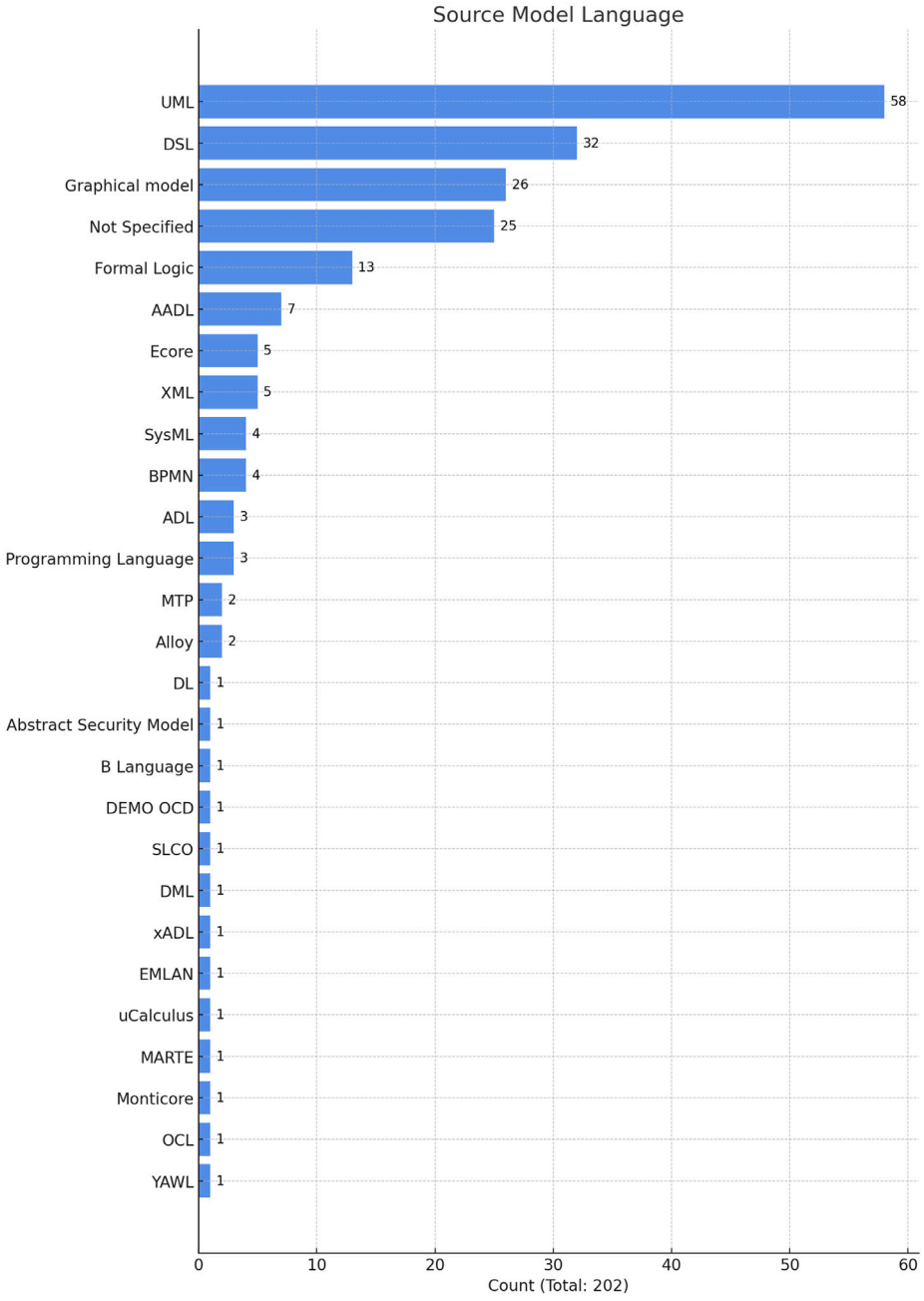

Fig. 6. Source languages.

Horizontal and Vertical Transformations: *Horizontal* transformations operate at the same abstraction level (van Amstel et al., 2008; Jadoon et al., 2019), whereas *vertical* transformations bridge different levels, such as from abstract designs to concrete implementations or vice versa (Klare and Gleitze, 2019; Esbai and Erramdani, 2015).

Syntactic and Semantic Transformations: *Syntactic* transformations change the format or structure of a model without altering its meaning (Yang et al., 2024; Krzywicki et al., 2021). In contrast, *semantic* transformations ensure that the essential behavior or properties of the model are preserved during the transformation (Cuadrado et al., 2017; Daniel et al., 2021; Moon et al., 2005).

Fig. 9 showcases the distribution of various transformation approaches used in the reviewed studies. Semantic transformations dominate with 51 occurrences (25%), highlighting their significant role in capturing meaning and intent during transformations, particularly in ensuring the accuracy and consistency of the output models. Vertical transformations, cited 44 times (22%), reflect their importance in refining or deriving detailed representations from abstract models. Horizontal transformations appear 22 times (11%), depicting their relevance in translating between models at the same abstraction level, often for interoperability or system integration.

Exogenous transformations are noted 31 times (15%), indicating their frequent use in mapping between models of different domains or languages. In contrast, Endogenous transformations, with 38 mentions (19%), point to their more specialized application in transformations within the same modeling language or domain. Syntactic transformations, cited 16 times (8%), showcase their role in structural or rule-based modifications. This distribution emphasizes the varying applicability of transformation types, with semantic and vertical approaches being the most prevalent.

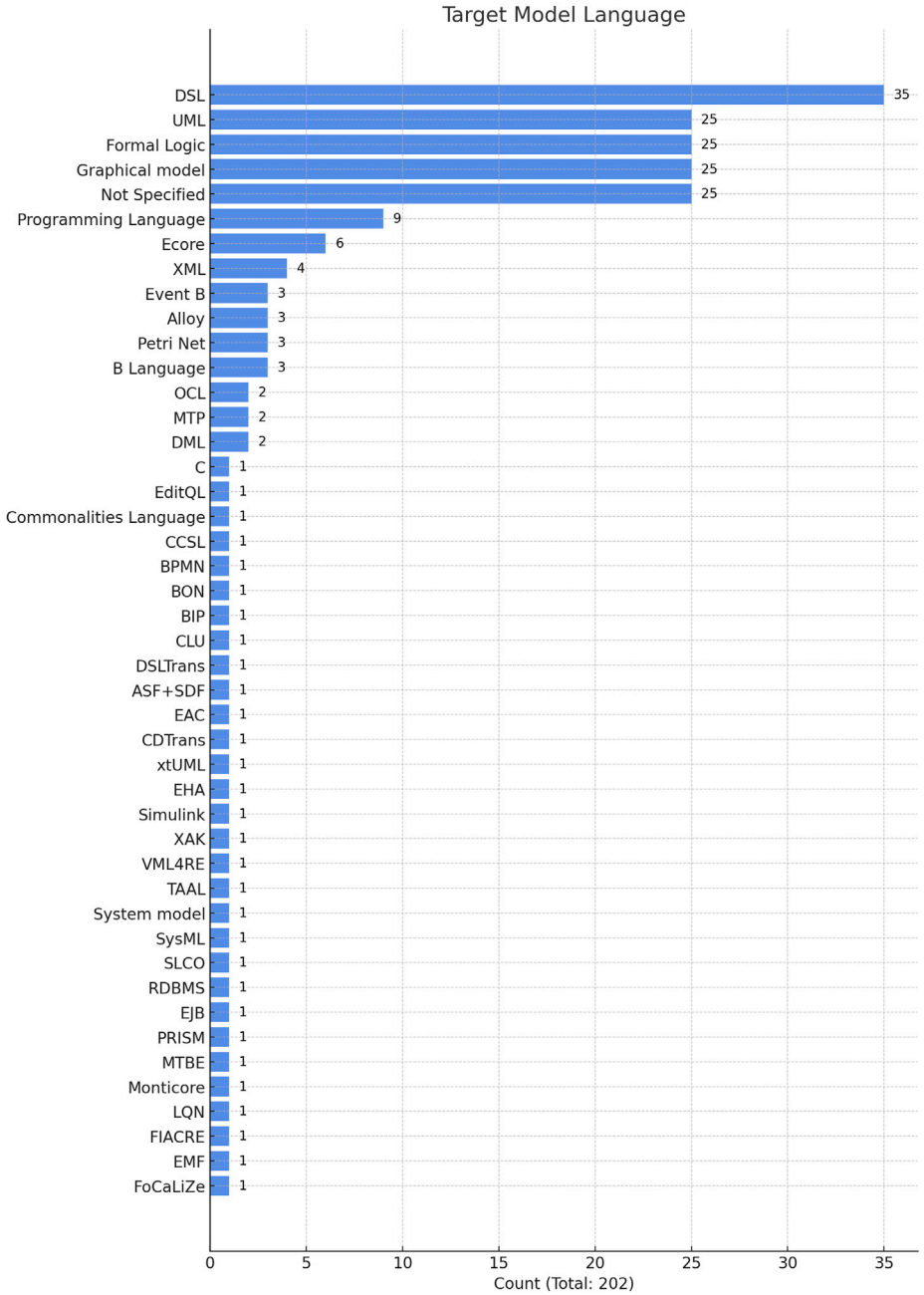

Fig. 7. Target languages.

5.5. RQ 1.5: Evaluation type

This subsection emphasizes the methodologies used to evaluate studies within the domain of model transformation and property preservation, highlighting the latest trends in research evaluation.

Fig. 10 provides an overview of the evaluation methodologies used in the reviewed literature. Case Studies dominate with 62 occurrences (31%), highlighting their vital role in demonstrating the practical application and effectiveness of proposed methods or frameworks in real-world scenarios. Rigorous Analysis, with 49 mentions (24%), is the second most prevalent type, reflecting the emphasis on thorough, theoretical evaluation to ensure the validity and reliability of the findings.

Discussions, cited 34 times (17%), highlight their importance in synthesizing and analyzing existing knowledge, offering valuable insights, or proposing future directions for research. Example Studies, appearing 26 times (13%), showcase their utility to illustrate specific use cases or simplified scenarios to explain methodologies. Meanwhile, Experimental Studies using Software Subjects, with 11 mentions (5%), show a smaller but notable focus on empirical evaluation via software simulations or prototypes. Lastly, Experimental Studies using Human Subjects are rare, with just 1 occurrence, suggesting limited engagement with user studies or human-centric evaluations in the reviewed work. This distribution indicates a strong preference for theoretical and case-driven evaluations, with limited emphasis on experimental or empirical studies.

Awareness of the various evaluation methods used in the studies, as shown in Table 8, allows for researchers to better understand the depth and rigor of the research on model transformation. It also helps to identify which methods are most effective in different contexts, guiding future studies towards adopting the most appropriate evaluation strategies to validate their findings.

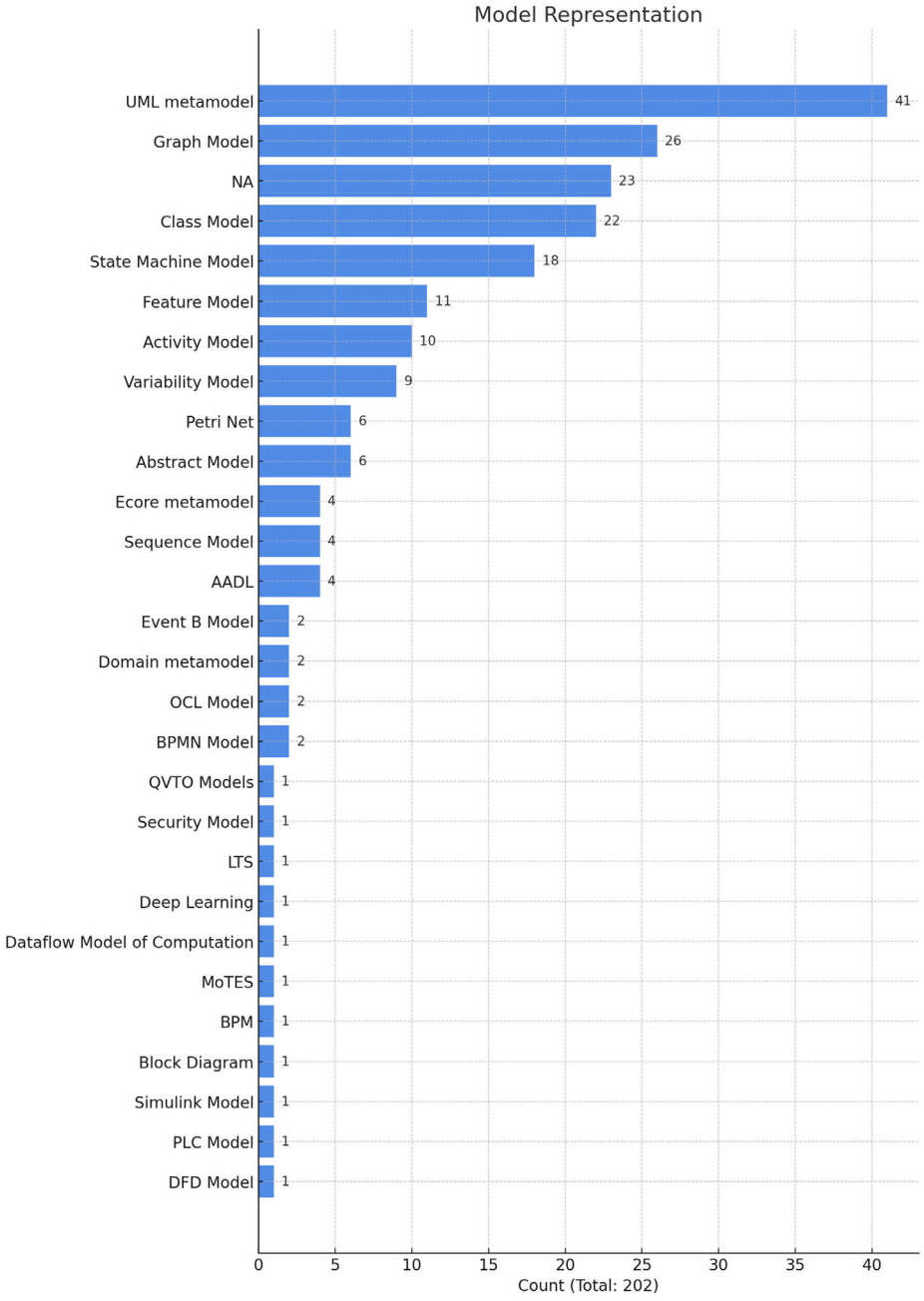

Fig. 8. Model Representation.

5.6. RQ 2: Model properties

The considered model properties include model consistency, syntax and semantics, functional and non-functional behavior, maintainability and reusability, and transformation verification (cf. Table 9). These properties are classified based on their quality scope defined by ISO/IEC 25010 such as functional quality, structural quality (model consistency, syntax, and semantics), non-functional quality (non-functional behavior, reliability, model verification), maintainability, and portability (reuse) (Estdale and Georgiadou, 2018).

Consistency Management is the most frequently addressed property, with 66 occurrences (33%) as shown in Fig. 11, reflecting its key role in ensuring alignment and coherence between models during and after transformations. This prominence shows the importance of maintaining system integrity in model transformations. Transformation Verification, with 50 mentions (25%), is the second most focused property, indicating a strong emphasis on validating the correctness and reliability of transformation processes.

Non-functional behavior, cited 49 times (24%), highlights its importance in addressing performance, scalability, and other quality attributes of systems. Syntax and Semantics, appearing 19 times (9%), point to the necessity of ensuring structural correctness and meaningful interpretation of transformations. Functional and Non-Functional Behavior, with 14 mentions (7%), indicates efforts to balance these two dimensions for comprehensive evaluations. Finally, Maintainability and Reuse, with only 4 mentions (2%), suggests limited attention in this area, pointing to potential opportunities for further research.

Overall, the data reveals a strong focus on ensuring correctness, quality, and consistency, and less emphasis on aspects like maintainability and reuse.

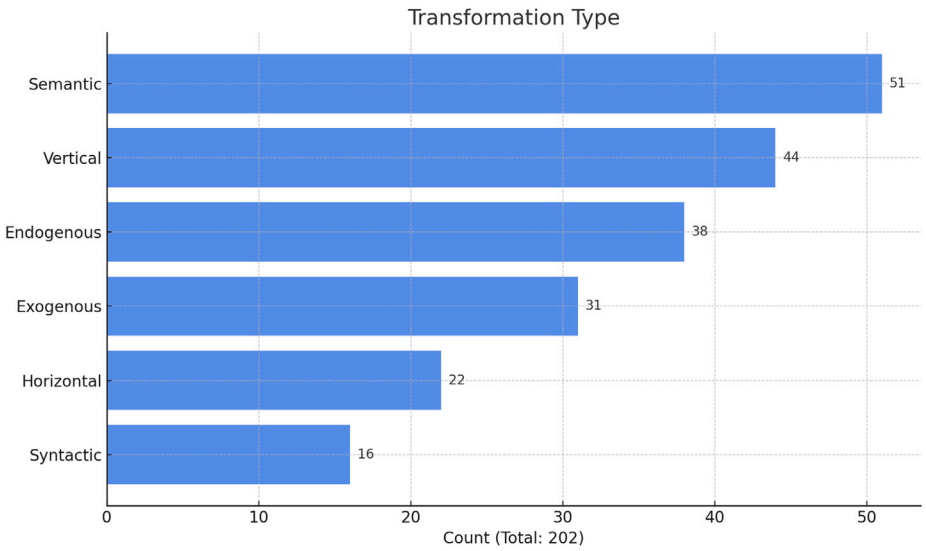

Fig. 9. Transformation types.

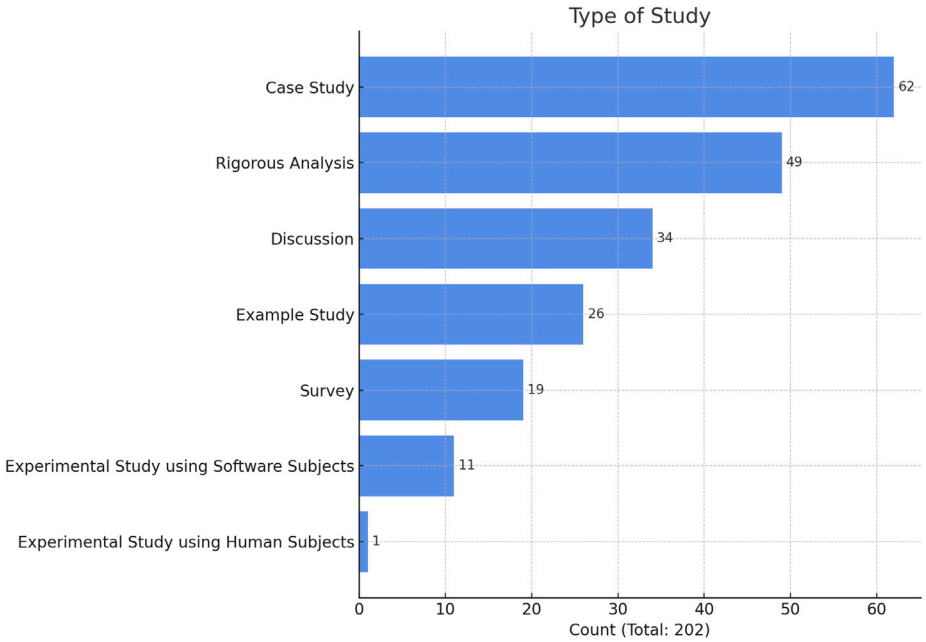

Fig. 10. Type of study.

5.7. RQ 3: Techniques used for property preservation and verification

Table 10 lists the different types of techniques identified. We categorize them into five major classes based on a comprehensive review and synthesis of the methodologies commonly employed in model transformation research. These classes, including formal verification, regression analysis, system testing, module testing, and unit testing, were derived by examining the primary objectives and scope of the verification methods reported in the literature.

Fig. 12 provides an overview of the methodologies used to ensure the accuracy and correctness of model transformations. System Testing is the most prevalent approach, with 100 occurrences (50%), emphasizing its importance in validating the overall functionality and behavior of the system under realistic conditions. Module Testing, cited 51 times (25%), follows as the second most common method, reflecting its role in verifying the correctness of individual components or modules in isolation before integration.

Formal Verification, appearing 34 times (17%), highlights the use of rigorous mathematical methods to prove the accuracy of transformations, particularly in safety-critical or high-assurance domains. Unit Testing, with 10 mentions (5%), focuses on testing smaller, specific parts of the transformation logic, ensuring localized correctness. Regression Analysis, cited 7 times (3%), is employed to verify that changes or updates do not introduce new defects.

The distribution of these approaches (cf. Table 11) shows a strong preference for techniques like system and module testing, while formal methods and lower-level testing are applied more selectively in specific contexts.

5.8. RQ 4: Verification tools

To address RQ 4, we explore the tools used for the verification of models in the context of MDD. The diversity in tool usage shows the varying requirements based on the model type and verification process's specific needs.

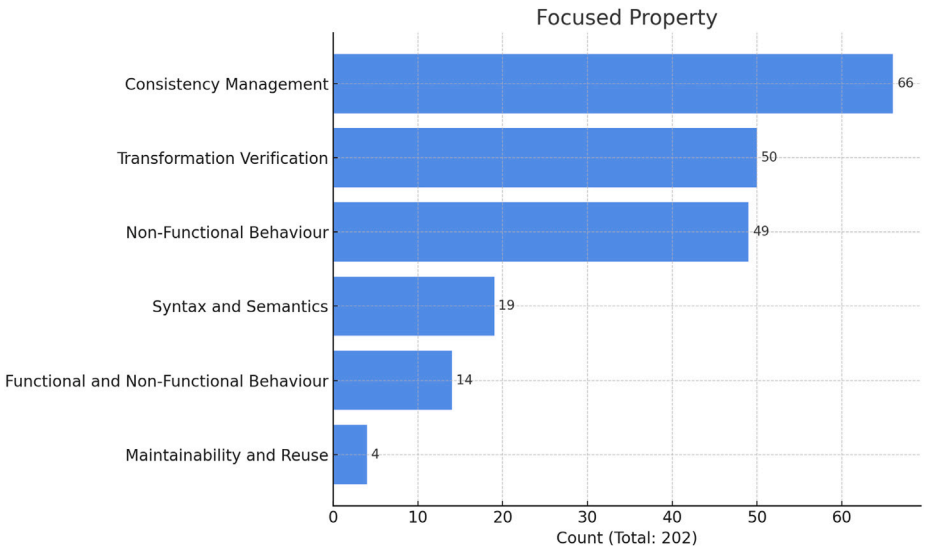

Fig. 11. Model properties.

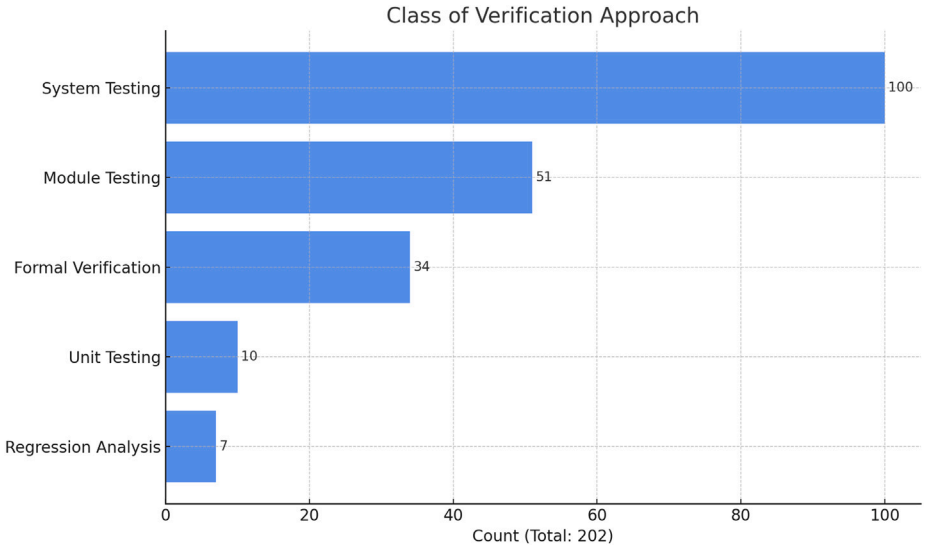

Fig. 12. Property verification and preservation classes.

This broad spectrum of tools, as shown in Table 12, highlights the multidimensional nature of tools, catering to a range of complexity levels and specific requirements. The choice of a verification tool is usually influenced by the expertise of the verification team, the complexity of the model, and the specific attributes of the model that require verification.

Our results indicate that a significant number of the studies rely on manual approaches to verify the model, indicating that a significant portion of the research does not specify any tool support, and suggesting that several approaches are still at the early proposal stage, and did not achieve the level of maturity required to implement them in a tool.

The category *Modeling Tools* includes various tools for the creation and visualization of data and models. These tools often support specific modeling languages like UML, which is the primary underlying representation for tools like UML RT (Babaei and Dingel, 2023), StarUML (Jadoon et al., 2019), and IBM Rational Rose (Egyed, 2010).

Transformation Tools are specifically designed to perform model transformations in MDD. These transformations are essential in automating various stages of software development, such as converting high-level design models into lower-level implementation models or generating executable code from models. Tools such as eMoflon (Weidmann, 2018), AADL Verification Tool (Hu et al., 2015), CDTrans (Hölldobler et al., 2015), and Groove (Ege and Tichy, 2019) automate the transformation process, reducing the potential for human error and speeding up the development process.

Among *Verification and Validation Tools*, model checkers such as UPPAAL (Basile et al., 2019) and NuSMV (Daw et al., 2014) are prevalent, mainly due to their robustness in handling complex verification scenarios involving temporal and logical properties. Tools such as AnAtlyzer (Lukács and Bartha, 2022) and Vitruvius (Klare and Gleitze, 2019) emphasize rigorous verification methods, often incorporating logical theorems that prove to ensure the correctness of model transformations. These tools are vital for applications requiring high levels of assurance, such as safety-critical systems. Other verification tools, including USE ModelValidator (Hilken et al., 2018) and MTB (Troya et al., 2018), provide interactive environments for property verification, allowing more flexibility and user control during the verification process.

Domain-specific Tools are particularly designed to cater to the unique needs of a particular domain. These tools are typically built around

Table 8
Data extraction categories related to RQ 1.5.

Type of Study & Evaluation (Chen and Babar, 2011)	RQ 1.5
Case Study	An empirical inquiry that investigates a phenomenon within a real-life context (Reder and Egyed, 2012; Jilani et al., 2014).
Survey	A list of questions aiming to solve a specific problem (Ameller et al., 2021).
Discussion	Provided some qualitative, textual opinion (van Amstel et al., 2008; Zhu and Liu, 2009).
Rigorous Analysis	Rigorous derivation and proof for formal models (Abbas et al., 2021; Somogyi and Mezei, 2023).
Action Research	The researcher solves a real-world problem while having experience solving it.
Example Study	Authors describe an application and provide an example to assist the description, but the example is used to 'validate' or 'evaluate' as far as the authors suggest (Jadoon, 2024; Bilic et al., 2020).

Table 9
Data extraction categories.

Model properties	RQ 2
Maintenance and Reuse	Study related to transformation maintenance and model portability (Le Calvar et al., 2019; Castellanos et al., 2015).
Consistency Checking	Study related to model property preservation such as consistency checking (Reder and Egyed, 2012; Braga et al., 2014).
Functional and Non-functional Properties	Study related to functional and non-functional aspects of model transformation (Diskin et al., 2018; Wada et al., 2008).
Syntax and Semantics	Study related to model language syntax and semantic properties (Liu et al., 2010; Le et al., 2013).
Transformation Verification	Study related to property preservation and model correctness properties (Schneider et al., 2024; Hidaka and Tisi, 2024).
Non-Functional Behavior	Study related to properties like performance, privacy, traceability, scalability and model conformance (Esbai and Erramdani, 2015; Le Calvar et al., 2019; Narayanan and Karsai, 2008; Liu et al., 2005).

DSLs, which are specialized languages tailored for specific application areas and are easier for domain experts to use compared to general-purpose programming languages (e.g., TypeChef Le et al., 2013, DSLTrans Oakes et al., 2015, Simcan2Cloud Bernal et al., 2019, etc.). Among these tools, DOPLER (Vierhauser et al., 2010b; Feichtinger and Rabiser, 2020), FAMA Tool (Lopez-Herrejon and Egyed, 2012), and FeatureIDE (Guo et al., 2012) are utilized by the researchers to manage variability models.

5.9. RQ 5: Challenges in property preservation

We now highlight the challenges associated with the preservation of properties in MDD, drawing on classified and analyzed data using thematic analysis (Braun and Clarke, 2021; Bano and Zowghi, 2015). The challenges are presented in Table B.14 in Appendix A.

Some of the challenges pertain *System Inconsistency*, which is the risk of consistency issues among models, or within models. These can be due to the ambiguity of model artifacts caused by the absence of formal semantics for the languages adopted, but also due to other factors, like the use of too many languages that can make it difficult to ensure consistency or issues associated with model changes that can make it difficult to maintain consistency.

Other challenges are associated with *Error Detection*. The generation of test data when adopting transformation testing can lead to errors that are hard to detect. In addition, errors have been observed to remain undetected in the models even after several iterations, which indicates the poor ability of the techniques to identify all existing errors.

Challenges in *Modeling Languages* are mainly due to insufficiencies of the different languages used for source and target models. In particular, some languages do not support different levels of semantics, and very frequently, the semantics is not explicitly defined due to the use of semi-formal or non-formal languages. Table B.14 depicts some broader categories of these challenges and associated subcategories based on the thematic analysis. This table is presented in the appendix.

Non-functional quality attributes such as performance, security, reliability, scalability, and maintainability present significant preservation challenges. These challenges often arise from the inherent trade-offs between competing properties, which can impact the overall design of the system and the user experience.

5.10. RQ 6: Benefits of property preservation

Finally, we explore the advantages of property preservation in MDD, detailing how these benefits enhance the transformation process and contribute to overall system quality. By subjecting the analyzed data to thematic analysis (Braun and Clarke, 2021; Bano and Zowghi, 2015), we identified several key benefits that are instrumental in the development and reuse of transformation models.

The basic edges of property preservation identified after thematic analysis are transformation abstractions, conceptual models, intensive views, transformation chains, and intents. These artifacts play an important role in transformation development and reuse. The predefined constraints help the developers to maintain integrity and standards among the model transformations. Various formal approaches are also embedded in model transformations to maintain their accuracy and reliability, such as formal languages, formal refinements, and formal defect detection and prevention.

The role of DSLs is explicitly defined as a subset of transformation languages, specifically tailored to the domain of model transformations, providing domain-specific syntax and semantics for precise and reusable operations. Automated tools are also available that efficiently validate and verify specific transformations. These model transformations provide consistent models that conform to the source and target meta-models and support maintenance and reuse.

Variability management in transformations also benefits from feature modeling by providing a systematic and structured way to represent, manage, and implement the configurable aspects of a product family in SPLs. Feature models serve as a blueprint for product configuration, allowing stakeholders to select or deselect features to create specific product variants. They ensure that each variant adheres to the desired functional and non-functional properties by linking features to corresponding attributes. This linkage helps in validating that important properties like performance, security, and reliability are not compromised during product derivation.

The main benefits are presented in Table B.15 in Appendix B. These benefits are classified into subclasses based on the themes extracted for each category. These benefits collectively improve the correctness, efficiency, and adaptability of MDD practices, emphasizing the crucial role of property preservation in ensuring the longevity and relevance of models in evolving technological landscapes.

6. Summary and discussion

In this section, we focus on the empirical findings of our study and their importance within the MDD domain. We begin with a brief summary of the results corresponding to each RQ. Subsequently, we explore

Table 10
Data extraction categories related to RQ 3.

Class	Verification Approach	Description
Module Testing	Transformation Rules (Antignac et al., 2018)	Model evaluation using flexible and generic transformation rules as graph chains.
	OCL Constraints (Reder and Egyed, 2012; Boronat, 2017)	Using object constraint language rules to verify the output behavior of the model.
	Semantic Translation (Combemale et al., 2007)	Using language's semantic translation for translating target to source constraints.
	Rule Iteration Algorithm (Liu et al., 2005)	Using rule iteration and schedule algorithm for transformation definition meta-model.
	Consistency Constraints (Le Calvar et al., 2019; Stünkel et al., 2021)	Using scalable consistency management concepts for variability models in SPLs.
	Feature Constraints (Gwasem et al., 2023)	Using feature modeling rules in SPL engineering.
Formal Verification	Mathematical Analysis (Paige et al., 2007)	Using any formal verification methodology such as propositional logic to verify the system model.
	Propositional Logic (Asztalos et al., 2010)	Using propositional and predicate logical inferences for model verification.
	Theorem Proving (Aït Ameur and Aït-Sadoune, 2012)	Using formal theorem proving to verify model properties.
	Systematic Reification (Almeida et al., 2019)	Using conventional 2-level modeling techniques to preserve multi-level semantics.
	Test-based falsification (Beyer and Lemberger, 2017)	Using statistical tests to validate a model's output behavior by drawing inferences.
Unit Testing	Manual Inspection (Hinkel et al., 2019)	Using manual static or dynamic analysis to verify the system model.
System Testing	Two-level Inspection (Wang et al., 2008)	Using model inspection by passing composite and complex models into already defined model patterns.
	Trace-based Analysis (Westfechtel and Greiner, 2018)	Using trace propagation for mapping alterations in multi-variant models.
	Test case generation (Hilken et al., 2018)	Building test suites for identification of model errors.
Regression Testing	AADL Error Model (Qiuyan et al., 2011)	Comparison with AADL error model to implement verified transformation from AADL to UML models.
	Mutation Technique (Selim et al., 2012)	Using various alteration techniques to verify the model's behavior.
	Spectrum Fault Localization (Troya et al., 2018)	Fault localization based on spectrum analysis using test cases and the corresponding code coverage.

the broader implications of these findings by discussing representative studies in each category linked to the RQs. These example studies were selected based on their superior quality scores derived from our evaluation process. This not only highlights the robustness of our study but also improves our understanding of the evolving landscape in MDD.

6.1. RQ 1. What are the demographic aspects of the primary studies in the field of model transformation?

Yearly Distribution: Research in model transformation emerged in the past two decades, showing significant growth, particularly since 2015, highlighting the increasing relevance of MDD in modern technological contexts.

Venue Distribution: The field is prominently featured in conferences such as MoDELS, ICEIS, VaMoS, and SETIT, and in journals including SoSyM, TOSEM, and TSE. A predominant share of the work (105 studies, 52%), is published in conferences, whereas a significant almost equal share (93 studies, 46%) appears in journals. As conferences have historically been the predominant venues for publishing research in software engineering, including MDD however, we have also observed a gradual shift towards journal publications in recent years, consistent with trends in other software engineering sub-disciplines. This shift can be attributed to the growing emphasis on rigorous evaluations, methodologies, and reproducibility, which journals are better equipped to accommodate.

Transformation Languages: Formal Logic, with 42 mentions (21%), emerges as a significant category, reflecting its essential role in ensuring precision and correctness in transformation tasks. The visibility of Formal Logic suggests a strong reliance on rigorous, theoretical frameworks to support transformation techniques, particularly in contexts demanding verifiability and consistency. Similarly, ATL, cited 23 times (11%), demonstrates its importance as a widely used language for model transformations. Its efficiency and versatility make it a good choice for practical applications in MDD. Graph model, referenced 16 times (8%), highlights the growing preference for intuitive, visual techniques that make complex transformations more accessible to practitioners. Additionally, QVT, BPEL, B, Ecore, Java, and C++ are also utilized, albeit less frequently.

A substantial portion of studies fail to explicitly specify the transformation language used, as evidenced by the "Custom Rules" category

Table 11
Data extraction categories.

Classes of verification approaches	RQ 3
Unit Testing	Verification of individual model units (Hutchesson and McDermid, 2011; Stevens, 2010).
Module Testing	Verification of model components and artifacts (Wijs and Engelen, 2012; Fleurey et al., 2004; Chen et al., 2013).
System Testing	Verification of functional and non-functional aspects of model (Hülsbusch et al., 2010; Gerpheide et al., 2016).
Regression Analysis	Verification of model properties after incorporating changes (Babaei and Dingel, 2023; Qiuyan et al., 2011; Iqbal and Al-Azzoni, 2022).
Formal Verification	Verification of model properties using formal techniques such as theorem proving and model checking (Alshalalfah et al., 2023; Aït Ameur and Aït-Sadoune, 2012; Daw et al., 2014; Andraus and Sakallah, 2004; Fernandez Adiego et al., 2015).

dominating the results. This category is used to classify studies where the transformation logic is defined using ad hoc or implementation-specific rule sets that do not conform to a standardized model transformation language, such as ATL, QVT, or TGG. This trend highlights a need for greater clarity and standardization in reporting the tools and languages used in model transformations.

Transformation Types: The distribution of transformation types shows that semantic transformations (51 occurrences, 25%) and vertical transformations (44 occurrences, 22%) are among the most frequently applied approaches. This reflects a strong emphasis on refining abstractions and ensuring semantic preservation across different levels of model representation.

The use of endogenous transformations (38 occurrences, 19%)—where source and target models belong to the same language—and exogenous transformations (31 occurrences, 15%)—which involve mappings between different languages—demonstrates the ongoing relevance of both intra-language evolution and cross-language interoperability in MDD.

These results emphasize the importance of selecting the appropriate transformation type based on the task's requirements. While semantic transformations ensure consistency and accuracy, vertical transformations enable abstraction refinement, and horizontal and exogenous transformations address integration challenges. Future work could explore underutilized approaches, like horizontal transformations, to address intra-domain refinement needs or enhance the structural validity of models through syntactic transformations.

Evaluation Types: The distribution of evaluation types reflects a strong preference for theoretical and case-driven evaluations, with case studies included 62 times (31%) and rigorous analysis appearing 49 times (24%). This highlights the importance of demonstrating practical relevance through real-world scenarios and ensuring rigorous theoretical validation in model transformation research. Discussions, with 34 mentions (17%), further emphasize their role in synthesizing knowledge and proposing future directions, adding depth to the research landscape. The relatively smaller use of example studies (26 occurrences, 13%) highlights their utility in showcasing simplified use cases or specific methodologies.

However, the limited use of experimental studies points to a gap in empirical validation. Experimental studies using software subjects, with 11 mentions (5%), provide some empirical evidence through simulations or prototypes. In contrast, experimental studies involving human subjects, with just 1 occurrence, indicate a significant underrepresentation of human-centric evaluations, which could offer valuable insights into real-world usability and adaptability of transformation techniques.

To advance the field, future studies should focus on experimental evaluations with software subjects to assess performance, scalability, and transformation accuracy under controlled conditions. Also, incorporating human-centric evaluations can provide valuable insights into usability, developer experience, and real-world adaptability of transformation tools and methods.

Significance of Model Transformation: Model transformation and property preservation are foundational aspects of MDD, supported by significant advancements in methodologies for property verification and preservation (Braga et al., 2014; Brambilla et al., 2017; Samimi-Dehkordi et al., 2018; Meghzili et al., 2016). Research on model requirements demonstrates the feasibility of adopting certain methodologies akin to model specifications to address system modeling challenges (Guerra and Soeken, 2013; Magalhaes et al., 2019; Jadoon et al., 2019; Moon et al., 2005; Liu et al., 2005; Lano et al., 2019). Debnath et al. (2011), Flake and Mueller (2003) and Muliawan (2008) also advocate for enriching modeling specifications with semantic extensions to enhance the refinement and maintainability of models.

Nguyen and Dang (2021) and Weidmann et al. (2019) utilize linear programming techniques and graph grammars to ensure consistency and conformance between meta-models and contract models, addressing key challenges in model coherence.

Lano et al. (2018, 2021) propose automated methods for recording and analyzing model-to-meta-model mappings during model specifications, enabling more efficient and consistent MDD processes. Similarly, Lúcio et al. (2016) introduce a framework defining six common transformation intentions, offering a comprehensive modeling structure. This framework is essential for identifying significant transformation properties and supports the effective management of transformations in industrial model-based systems.

6.2. RQ 2. What are the model properties considered?

The results provide insights into the key properties considered in the domain of MDD. These properties play a vital role in guiding the development, transformation, and evaluation of models. Below, we discuss the significance of each property, its implications for the field, and potential areas for improvement.

6.2.1. Model consistency: A key priority

Consistency is a fundamental property of model transformation, with a significant portion of research dedicated to it, representing 66 occurrences (33%) on the topic (Neis et al., 2019; Reder and Egyed, 2012; Braga et al., 2014; Le et al., 2013; Lopez-Herrejon and Egyed, 2012; Guo et al., 2012; Lopez-Herrejon et al., 2010; Spanoudakis and Zisman, 2001; Rose et al., 2009). It ensures that model transformation ons preserve the necessary features as independent functionalities of the system particularly in bi-directional model transformations, where inconsistencies can compromise system integrity (Neis et al., 2019). The prominence of consistency management reflects the maturity of the research community in recognizing the challenges of maintaining synchronized models across various abstraction levels and domains. Braga et al. (2014) emphasize the use of DSLs to build consistent models that comprehensively represent system complexities. Prior to the adoption of DSLs, OCL structural properties were primarily used, though they lacked coverage of behavioral aspects. Early error detection through consistent model transformations is necessary, but challenges such as model refactoring and merging demand scalable and validated design rules (Reder and Egyed, 2012; Mohagheghi et al., 2009). Rose et al. (2009) propose a method for tracking inconsistencies within the Eclipse Modeling Framework, enhancing automated consistency checks. Despite this strong emphasis, there is room for further innovation, particularly in handling large-scale, complex systems where consistency issues become more pronounced.

Table 12
Verification tools (unless indicated differently, tools occur only once).

Category	Tools RQ 4
Modeling Tools	UML RT, StarUML (2), IBM Rational Rose (2), MagicDraw, REFINER, SysML Tool, Ark, Pluto, DREAM, Eclipse Process Framework (EPF), SPSS, DEMO ADO Tool, USE Tool, MARTE, Matlab, RTL, MS Visio, WebML, Prefuse, and xtUML
Transformation Tools	eMoflon (6), Henshin (3), TGG Tool (4), CDTrans, Groove, Simulink, BPEL2EventB, Eclipse Agile UML Toolset, and Multecore
Verification and Validation Tools	UPPAAL (5), AnAtlyzer (3), Vitruvius (2), USEModelValidator (2), NuSMV (3), MTB, DMOSES, ProB (3), SAT Solver, HADAS, PVS Toolkit, SPIN, MuJava, EclEmma, Motter, Vapor Tool, Test Generator Tools, TINA, EMF Plugin, mCRL2, SAT4j, Totem, TestGenerator Tool, and MODEST
Domain-specific Tools	TypeChef, DSLTrans, Simcan2Cloud, Monticore, ML2-Editor, OIDE, Acceleo, AADL2TASM (2), HADAS, MARTE, Petri net Tool, BIP Tool, Simulink, Matlab, Ground Tram, Pluto, ViTAL, DOPLER (3), FAMA Tool, and FeatureIDE

Summary

A total of **105** studies (52%) are published in conferences; **93** studies (46%) are published in journals. The majority of the studies are evaluated through case studies (**62** studies, 31%) and rigorous analysis (**49** studies, 24%), while there is a limited number of experimental studies and surveys. Dedicated venues (like MoDELS) allowed researchers to focus on core model-based artifacts.

Take-away messages

1. Semantic and vertical transformations dominate the field, but gaps in horizontal and syntactic approaches present opportunities for further research.
2. Most studies prioritize case studies and rigorous analysis, with limited experimental validation, particularly involving human subjects, highlighting a need for more empirical evaluations.
3. Established tools like ATL, Formal Logic, and UML are prevalent, but inconsistent reporting of transformation tools and languages limits reproducibility and comparability.

6.2.2. *Transformation verification: Emphasis on correctness*

With 50 mentions (25%), transformation verification is the second most addressed property. Verification is vital in modeling to ensure that transformations produce correct and reliable outputs that adhere to the desired specifications (Wijs and Engelen, 2012; Antignac et al., 2018; Le Calvar et al., 2019; Narayanan and Karsai, 2008; Liu et al., 2005; Stevens, 2010; Paige et al., 2007; Cabot et al., 2010). This focus highlights the community's commitment to establishing trust in automated processes and mitigating risks associated with faulty transformations.

Le Calvar et al. (2019) propose a performance evaluation framework that assess the efficiency of model transformations by measuring the number of input models processed and the processing time. This framework offers a systematic approach to evaluating transformation performance, providing a systematic method for selecting effective transformation techniques.

Narayanan and Karsai (2008) and Stünkel et al. (2020) emphasize the need to verify property equivalence in graph models, using bi-simulation techniques to check consistency between the input and output models. This approach allows for accurate comparison and evaluation of model properties.

Liu et al. (2005) address the challenge of maintaining the coherence of source/target models, advocating for the use of the ATLyzer tool to evaluate the integrity constraints, thereby ensuring conformance among models.

Stevens (2010), Giese and Wagner (2008) and Terayawan and Vatanawood (2019) support the concept of bi-directional transformations, to maintain consistency between source and target models, particularly when translating abstract transformations into more specific ones.

Wijs and Engelen (2012) suggest an iterative verification approach to maintain model coherence, especially when features or functionalities are modified within the system. Paige et al. (2007) advocate for a model refinement approach that not only supports conformance but also aids in managing consistency between source and target models.

Techniques for verification often involve formal methods, test cases, or runtime checks (Chen et al., 2022; Mouakher et al., 2022; Ghaedi Heidari and Ajoudanian, 2023). The relatively high number of mentions indicates significant research activity aimed at developing robust and scalable verification mechanisms. However, achieving comprehensive verification remains a challenge, especially for transformations involving multiple interacting models or those with non-deterministic elements. Future research could explore lightweight yet effective verification techniques to make them accessible and applicable in diverse MDD scenarios.

6.2.3. *Non-functional behavior: Addressing system quality*

While functional correctness is foundational, non-functional requirements determine the practicality and efficiency of model-driven solutions in real-world scenarios. The relatively high focus on non-functional behavior suggests that researchers are paying attention to system attributes that go beyond correctness. However, achieving a balance between functional and non-functional goals remains a challenge.

Non-functional behavior, cited 49 times (24%), highlights the growing recognition of performance, scalability, and other quality attributes such as privacy, safety, traceability, variability, and reliability (van Amstel et al., 2008; Jadoon et al., 2019; Zhu and Liu, 2009; Bilic et al., 2020; Wada et al., 2008; Guo et al., 2012; Spanoudakis and Zisman, 2001; Fatwanto and Boughton, 2008; Munoz et al., 2021;

Syriani and Gray, 2012; Drave et al., 2022). Bilic et al. (2020) discuss the use of specific variability modeling languages for managing variable properties in SPLs, highlighting that annotated variability models, though comprehensive, are prone to errors and can lead to model inconsistency.

To address potential inconsistencies, feature constraints extracted from variability models are analyzed and compared with the overall product line models, to detect and rectify inconsistencies or ambiguities (Spanoudakis and Zisman, 2001; Ayala et al., 2023; Daniel et al., 2021). Guo et al. (2012) emphasize the importance of quality management within product lines through effective feature modeling.

Diskin et al. (2018), Drawel et al. (2022) and Krystosik and Turlej (2006) highlight the role of vertical model transformations to ensure that the functional and non-functional quality attributes of a system, such as safety and security, are preserved. They use mathematical techniques for rigorous model checking to maintain these properties. Wada et al. (2008) further elaborate on the non-functional aspects of web models, discussing elements like privacy, usability, availability, and performance, and how integrating service architecture can enhance the reliability of models.

Although certain bidirectional transformation languages, like TGGs (Stünkel et al., 2021) and VIATRA (Mkaouar et al., 2022), embed mechanisms to maintain consistency, other quality attributes—including scalability, performance, maintainability, and usability—also need to be preserved during transformations. Achieving consistency and coherence may introduce trade-offs with other non-functional quality attributes, requiring careful management to balance these competing priorities.

6.2.4. Functional and non-functional behavior: A balanced approach

The combined consideration of functional and non-functional behavior, with 14 mentions (7%), indicates efforts to address both aspects in an integrated manner. Balancing these dimensions is essential for thorough evaluations in MDD. While functional correctness ensures the system operates as intended, non-functional considerations often determine the system's quality and success in real-world applications (Meyma et al., 2023; Jadhav et al., 2024; Bernal et al., 2019; Dávid et al., 2016; Demuth et al., 2016).

The limited attention to this context highlights an opportunity for more holistic approaches that do not treat functional and non-functional aspects in isolation. Future studies could explore frameworks that incorporate trade-off analyses to achieve optimized solutions.

6.2.5. Model syntax and semantics: Ensuring structural and interpretive validity

With 19 occurrences (9%), syntax and semantics represent the necessity of ensuring both structural correctness and meaningful interpretations of model transformations. Syntax checks validate adherence to modeling notations and standards, while semantic analysis ensure that transformations preserve the intended meaning of the models (Le et al., 2013; Almeida et al., 2019; Hudson and Sprinkle, 2011; Klare et al., 2021).

Klare et al. (2021) propose a view-based semantic development approach as a solution for constructing well-defined and structured semantics. This method facilitates the definition of semantic equivalence across models within a domain, aiding in consistent transformations.

Le et al. (2013) emphasize the essential role of semantics in ensuring that a model fulfills its intended functionality. Hudson and Sprinkle (2011) point out that semantic properties are often embedded into models using complex domain-specific objects, which are integrated into automated transformation patterns. These patterns allow developers to customize transformations, with adjustable attributes embedded in the models.

Almeida et al. (2019) highlight the challenges of maintaining semantic integrity when translating multi-level meta-models into more conventional two-level models. They recommend a systematic approach to modeling class types and their instances through reification, ensuring that the deep semantics of multi-level models are preserved during this simplification process.

Antignac et al. (2018) propose translating the semantics of source language constraints into target constraints based on the transformation principles to verify the integrity and correctness of the models.

Although this property is less frequently discussed compared to consistency and verification, its importance cannot be understated. Enhanced research on semantic equivalence and interoperability between modeling languages could address current gaps.

6.2.6. Model maintainability and reuse: An underexplored area

Maintainability and reusability are essential attributes in modeling, yet only 4 studies (2%) have mentioned these aspects in MDD (Macías et al., 2019; Castellanos et al., 2015; de Lara et al., 2017; Boronat, 2022). Maintainability ensures that models and transformations can be easily updated or extended, while reuse promotes the leveraging of existing assets to save time and effort.

Castellanos et al. (2015) introduce the concept of optimized reuse, where careful selection of model units for reuse ensures error minimization and efficiency. The model units must meet specific pre- and post-conditions related to the source and target models to prevent complexities and potential errors, especially in scenarios involving extensive model transformations.

de Lara et al. (2017) highlight the significance of reusability in model transformation by proposing a framework that encapsulates transformations into reusable components. This approach enhances efficiency, thus reducing the effort, time, and cost associated with performing transformations for similar tasks. Additionally, they suggest segmenting system meta-models into smaller, more manageable parts, to improve maintainability and integration.

The lack of focus on these aspects might be due to the predominant emphasis on correctness and consistency, which are more immediate concerns. However, neglecting maintainability and reuse can lead to increased costs and reduced agility over time. Future research could aim to develop methodologies, tools, and metrics to evaluate and enhance these properties, especially in domains where the long-term evolution of systems is needed.

6.3. RQ 3. What are the techniques used to ensure that model properties are preserved in the transformation?

The study of techniques used to preserve model properties during transformations reveals various methods categorized into five major classes: formal verification, regression analysis, system testing, module testing, and unit testing. Each technique brings unique strengths and caters to specific requirements, reflecting the diverse nature of model transformation verification in MDD. Below is a detailed discussion of these approaches, their applications, and their implications.

6.3.1. System testing: Validating overall functionality

With 100 occurrences (50%), system testing is the most widely used technique for ensuring model properties are preserved. This technique validates the overall functionality and behavior of the system under realistic conditions. The emphasis on system testing reflects the need to ensure that transformed models meet both functional and non-functional requirements when integrated into larger systems.

Key techniques such as two-level inspection (Wang et al., 2008), trace-based analysis (Westfechtel and Greiner, 2018; Bessifi et al., 2021), and test case generation (Selim et al., 2012) allow practitioners to simulate real-world scenarios and identify errors across the entire model.

Test case generation is a useful methodology in verification, particularly in test-driven development environments. It requires rigorous verification based on generated test data to ensure model integrity. Selim et al. (2012) and Olajubu et al. (2017) explore the

Summary

The results demonstrate that most studies focus on consistency management methods and the problems associated with inconsistent models, namely **66** (33%).
A total of **19** studies (9%) are related to the model's syntax and semantic attributes.
A total of **63** studies (31%) are related to functional and non-functional property management in which functional aspects and various other attributes are considered, such as performance, traceability, verifiability, security, and changeability.
In contrast, only **4** (2%) of the studies are related to post-development properties, such as maintainability and reuse.

Take-away messages

1. Consistency management is vital in MDD, ensuring alignment and coherence between models, especially in complex bi-directional transformations.
2. Transformation verification and non-functional behavior highlight a strong emphasis on correctness, reliability, and performance, reinforcing the importance of quality assurance.
3. Maintainability and reuse receive limited attention, presenting opportunities for research to improve system evolution and efficiency.

generation of search-based automated test cases, which focus on measuring the branch distance of model transformations to identify the optimal path. This verification approach is versatile and applicable to various transformation languages, including ATL, B, and various DSLs.

Equivalence partitioning is another technique that is used to categorize test cases into classes based on their characteristics. Hilken et al. (2018) discuss the application of this method by deriving test cases from two equivalence classes defined by source and target models using OCL constraints. Each class contains equivalent models, and test cases are generated through a model validator tool, enabling bidirectional testing of model transformations.

Jilani et al. (2014) focus on black box testing for source meta-model, which complements white box testing used to cover the transformation implementation. This dual approach ensures complete verification across both the structure and behavior of the model.

These methods are particularly effective in large-scale systems with interconnected models, where localized testing may miss integration issues. Despite its popularity, system testing can be resource-intensive, requiring significant computational power and effort to develop exhaustive test cases.

6.3.2. Formal verification: Rigorous mathematical approaches

Formal verification, found 34 times (17%), employs mathematical methods to prove the correctness of model transformations, which is particularly significant in safety-critical and high-assurance domains such as aerospace, healthcare, and automotive, where errors can have catastrophic consequences.

The formal verification of model properties use broad spectrum of methodologies, incorporating advanced model-checking techniques that employ logical inferences, deductive verification, and algorithm checking to identify defects in models, each playing a vital role in ensuring model correctness (Klare and Gleitze, 2019; Cheng and Tisi, 2017; Troya et al., 2018; Aït Ameur and Aït-Sadoune, 2012; Abbas et al., 2021; Braga et al., 2014; Calegari and Szasz, 2013; Prehofer, 2016; Moffett et al., 2013; Liebrenz et al., 2018; Kang et al., 2013; Hadad et al., 2020). Other common methods include propositional logic (Asztalos et al., 2010), theorem proving (Aït Ameur and Aït-Sadoune, 2012), and systematic reification (Almeida et al., 2019). These techniques offer high levels of confidence in transformation accuracy by validating properties like consistency, semantic preservation, and adherence to constraints.

Weidmann (2018), Barkowsky and Giese (2024), Fritsche et al. (2021) and Weidmann and Anjorin (2021) highlight the use of bidirectional transformation tools, like eMoflon, which use TGGs to formally link source and target models, thus effectively managing their consistency.

Furthermore, spectrum-based fault localization is employed to predict the presence of bugs in models based on test case data and the corresponding code coverage criteria. Troya et al. (2018) demonstrate that this technique provides superior fault localization compared to some existing practices by analyzing the effectiveness of the generated test cases in identifying defects.

Embedded systems, particularly hybrid control systems that exhibit both continuous and discrete behavior, pose significant challenges due to their complex and responsive nature (Krzywicki et al., 2021). These systems often rely on informal model semantics, complicating the maintenance of consistency, especially in tools like Simulink. Liebrenz et al. (2018) support the application of formal logic in the development of reliable embedded systems. They propose the use of differential dynamic logic to convert informal assertions into rigorously formalized language semantics, which can then be verified using theorem provers within Simulink. This approach not only supports the verification of both continuous and discrete properties, but also facilitates the automated generation of models in hybrid systems.

However, apart from several advantages, formal verification is resource-intensive and often requires specialized expertise, limiting its adoption to niche applications.

6.3.3. Unit testing: Localized validation

Unit testing, cited 10 times (5%), focuses on testing individual components of the transformation logic. It ensures that smaller, specific parts of the system function correctly, providing localized correctness.

Techniques like manual inspection (Magalhães et al., 2020b; Hutchesson and McDermid, 2011; Ameller et al., 2021; Gerpheide et al., 2016; Debnath et al., 2011) are used to verify model units, particularly in early development stages or for simpler systems. While unit testing is less comprehensive than other approaches, it plays a vital role in catching errors at the micro level before they propagate to higher abstraction levels. The main limitation of unit testing is its inability to address integration and system-wide issues.

6.3.4. Regression analysis: Preserving properties after changes

Regression analysis, with 7 mentions (3%), is employed to verify that modifications or updates to models or transformation rules do

Summary

A total of **100** studies (50%) focus on system-level testing of models, while **34** studies (17%) use a formal verification approach for verification, and **10** studies (5%) use model units for verification. In contrast, **7** studies (3%) focus on regression testing of the models. We see several verification methodologies, ranging from dynamic formal methods to static code analysis procedures. However, all these methods greatly depend on the model artifact being verified.

Take-away messages

1. System and module testing dominate as the preferred techniques in MDD, emphasizing their effectiveness in validating higher-level functionality and integration.
2. Formal verification methods are critical for high-assurance domains but face limited adoption due to complexity and resource demands, highlighting the need for more accessible and automated tools.
3. Regression analysis and unit testing are underutilized, presenting opportunities for research to enhance iterative model evolution and localized error detection.

not introduce new defects. This technique is particularly valuable in iterative development processes or when extending existing models.

Additionally, Magalhães et al. (2016) propose a Model Transformation Profile (MTP), and a transformation environment specifically designed for model transformation development. They address challenges related to the changeability and maintenance of models, particularly when a transformation chain undergoes modifications or is reused. Their validation process demonstrates that this approach leads to the generation of executed transformation code that is not only reusable but also portable.

Approaches like mutation techniques (Cuadrado et al., 2017; Vierhauser et al., 2010b; Cuadrado et al., 2018; de Lara et al., 2017), AADL error models (Qiuyan et al., 2011; Mkaouar et al., 2022; Yang et al., 2014), and spectrum fault localization (Troya et al., 2018) help ensure that changes in models preserve desired properties and ensure system correctness.

These verification approaches showcase differences in their focus: while some prioritize maintainability and reusability (e.g., MTP), others emphasize fault detection and validation (e.g., mutation techniques, AADL error models, spectrum fault localization). Despite their effectiveness, regression analysis remains underutilized, particularly for large-scale or evolving systems, highlighting the need for further research to enhance scalability, automation, and adoption of these verification techniques.

6.3.5. Comparison with related empirical work

In this section, we provide a comparative summary between the verification techniques identified in this study and those reported in prior empirical work (Höppner and Tichy, 2024), where the importance of model transformation language capabilities is assessed empirically. This comparison underlines the alignment, differences, and relative strengths of each class of verification approach, offering insight into their practical adoption, methodological intent, and perceived effectiveness. The summary is presented in Table 13.

Table 13
Comparison with prior empirical work.

Verification class	Comparison with empirical findings from (Höppner and Tichy, 2024)
System Testing	Widely adopted in both empirical studies and practice. Valued for its ability to validate end-to-end functionality. Frequently used in large-scale or integration-heavy systems. Recognized for intuitive setup, though resource-intensive.
Module Testing	Seen as a balance between precision and scalability. Commonly used where modular architectures exist. Confirms usefulness in catching localized defects. Higher test coverage than unit testing in complex systems.
Unit Testing	Helpful in early-stage development. Limited in catching inter-component issues. Considered lightweight and easy to implement. Typically used together with broader testing techniques.
Regression Analysis	Often underutilized in empirical studies. Useful in iterative and incremental model transformation. Effective to find newly introduced bugs post-modification. Needs improved automation and scalability tools.
Formal Verification	High accuracy but low adoption in practical settings. Steep learning curve and high tool complexity. Mainly used in safety-critical and high-assurance domains. Offers unmatched confidence in transformation correctness.

6.4. RQ 4. What are the different automated tools used to ensure that model properties are preserved in the transformation?

Model verification is a significant aspect of ensuring the accuracy and reliability of MDD processes. Various automated tools have been developed to facilitate this verification process, each designed to address specific aspects of model transformation and enhance the overall integrity of the process. These tools span various categories—modeling, transformation, verification and validation, and domain—specific tools—highlighting the diverse needs of MDD processes.

Interestingly, 74 of the studies (37%) opt not to use specialized tool assistance, relying instead on manual methodologies for verification.

Tool Utilization: Tools like anATLyzer (Cuadrado et al., 2017), Vitruvius (Klare et al., 2021), and EMF plugins (Rose et al., 2009) are commonly employed for verification purposes. Other tools include ProB (Leuschel et al., 2011), Vapor (Andraus and Sakallah, 2004), Groove (Ege and Tichy, 2019), SPIN (Narayanan and Karsai, 2008), SysML (Bilic et al., 2020; Baouya et al., 2015; Lugou et al., 2016), Motter (Jilani et al., 2014), eMoflon (Westfechtel and Greiner, 2018; Weidmann, 2018), DSLtrans (Hölldobler et al., 2015), DEMO (Gray et al., 2020), Eclipse (Rose et al., 2009), MTB (Troya et al., 2018), Mujava (Selim et al., 2012), Totem (de Lara et al., 2017), UPPAAL (Daw et al., 2014), Simulink (Liebrenz et al., 2018), VIATRA (Csertán et al., 2002), and IBM Rational Rose (Egyed, 2010). These suit various model types and primarily target ATL, Java, C++, or B transformation languages, offering IDEs for early bug detection, visualization, and resolution (Cuadrado et al., 2018, 2022; Almasri et al., 2022).

Specific Tool Functions: ProB offers automatic property verification for formal B models, enhancing accuracy and efficiency (Ameller et al., 2021; Zhu and Liu, 2009; Leuschel et al., 2011; Yang et al., 2014; Zhang et al., 2017), with the potential for scalability to support large-scale enterprise systems. UPPAAL is preferred for quantitative analysis and model simulation, while Simulink and its dynamic logic support excels in simulating hybrid systems (Basile et al., 2019).

Summary

A total of **100** studies (50%) focus on automated tool support in transforming and testing models, while **74** studies (37%) do not specify any tool usage.
Transformation tools, like eMoflon and Henshin, are widely used to automate model-to-model transformations, reducing manual effort and improving efficiency.

Take-away messages

1. Tools like ProB, UPPAAL, Simulink, and eMoflon offer specialized functionalities for formal verification, simulation, and consistency management, reflecting the diverse demands of model verification tasks in MDD.
2. Despite significant progress in tool development, the gap between available tools and the requirements of certain complex verification tasks suggests a need for ongoing refinement and expansion of tool capabilities.

Application in Component-Based Development: UML state machine models are instrumental in testing communication protocols and capturing behavior of component ports that align with specified protocols. Tools like IBM Rational Rose effectively perform protocol conformance checking in embedded systems (Moffett et al., 2013; Gadelha Queiroz and Vaccare Braga, 2014; Chen et al., 2013). Additionally, graphical models facilitate error and defect identification at an abstract level through visualizations (Ege and Tichy, 2019; Schneider et al., 2024).

Challenges and Innovations: Despite the variety of languages available for modeling hardware systems, manual abstraction remains a challenge, often leading to potential errors and omissions. The Vapor tool addresses this by employing automated inferences to significantly reduce defects in functional properties (Andraus and Sakallah, 2004; Nguyen and Dang, 2018).

Automated tools are paramount for the verification of properties in complex embedded systems, playing a vital role in maintaining the reliability and accuracy of model transformations. The continuous development and refinement of such tools are necessary to overcome the challenges posed by manual interventions and enhance the effectiveness of model verification processes. Common requirements for transformation tools include scalability, accuracy, and usability. Tools must handle large-scale and complex models effectively, ensuring correctness and reliability while maintaining performance. Support for various modeling and transformation languages (e.g., ATL, B, Java, and UML) is essential to address diverse MDD requirements. Additionally, tools should provide IDEs with visualization, early bug detection, and resolution capabilities to facilitate ease of use for developers.

The need for a composite approach to resolve inconsistencies stems from the limitations of individual tools in addressing all aspects of model verification and property preservation. Inconsistencies can arise from various sources, such as ambiguity in model artifacts, language incompatibilities, or manual interventions. A composite approach integrates multiple techniques such as formal verification, simulation, and automated defect detection to address these challenges exhaustively.

However, despite the advances in tool development, the reliance on manual methodologies by several studies highlights a significant gap in tool coverage or the complexity of certain verification tasks that automated tools have yet to fully address. Innovations such as the Vapor tool, Henshin, and eMoflon, which employ automated inferences to minimize defects, indicate ongoing efforts to bridge these gaps.

6.5. RQ 5. What are the challenges of model transformation and property preservation?

MDD faces significant challenges in preserving model properties during transformations. These challenges stem from system inconsistency, limitations in modeling languages, error detection, and non-functional quality attributes. These challenges are categorized into subcategories in Table B.14. We defined six major subcategories, which we discuss in the next sections.

6.5.1. System inconsistency

Inconsistency remains a challenge in model transformation, particularly evident when models are translated into higher-level transformations using declarative transformation rules (Hidaka and Tisi, 2024; Faunes et al., 2012). This inconsistency often results in non-persistent quality attributes and a weakened model structure, ultimately leading to an unmaintainable system with non-reusable components.

The primary issues associated with model inconsistency include ambiguity, redundancy, non-scalability, and poor model translation, as detailed in Table B.14. Ambiguity in transformations can introduce significant confusions in model intent and function, as noted by Antignac et al. (2018) and Hilken et al. (2018). Redundancy, highlighted by Le et al. (2013), Egyed (2010), Lano et al. (2018), Rose et al. (2009) and Li et al. (2020), not only bloats the model but also undermines structural integrity and consistency. Moreover, the challenges of non-scalability, highlighted by Cuadrado et al. (2017) and Troya et al. (2018), result in major difficulties in effectively managing complex models and web services.

Reder and Egyed (2012) and Wijs and Engelen (2013) emphasize the need for improved consistency management during model translation to mitigate errors that can arise from such transformation processes. The persistence of certain errors, even after multiple iterations, highlight the challenges in detecting and correcting inconsistencies, as observed by Bilic et al. (2020).

Klare and Gleitze (2019) and Dávid et al. (2016) propose transformation methodologies that establish clearer relations between instances, aiming to ensure consistent verification in the generated target model. Furthermore, duplicated information within models can contribute to structural inadequacies, complicating the transformation process and requiring thorough management to preserve model integrity.

Addressing model inconsistency requires a composite approach that includes enhancing the clarity of model definitions, improving transformation tools and methods, and refining verification processes to ensure the integrity and maintainability of the transformed models. Hybrid verification methods can be utilized to offer a balanced solution to address the challenges of consistency and coherence. These methods combine the rigor of formal approaches, such as model checking and theorem proving, with the practicality and accessibility of semi-formal techniques like UML diagrams and DSLs.

6.5.2. Modeling language semantics

Model transformations fundamentally rely on robust language support to accurately translate both syntax and semantics. This is particularly challenging in model-driven approaches that extend beyond the two-level modeling paradigm to include multi-level models (Almeida et al., 2019). These transformations often require the preservation of complex multi-level semantics to ensure the integrity and functionality of the resulting models.

Numerous researchers have addressed the challenge of preserving model semantics during transformations. Aït Ameur and Aït-Sadoune (Aït Ameur and Aït-Sadoune, 2012) investigate the re-engineering of existing models to integrate new functionalities within model patterns, conceptualizing service composition as a transition system among various services. However, the modeling languages employed to define these compositions often exhibit ambiguity and lack formal semantics, resulting in the generation of unstructured models. This ambiguity complicates the transformation process and heightens the risk of semantic errors.

Flake and Mueller (2003), Berramla et al. (2015), Han et al. (2016) and Rahim et al. (2015) emphasize the importance of formal semantics in overcoming the limitations posed by inadequate semantic definitions. They advocate for more rigorous formalization of semantics to provide clearer and consistent modeling frameworks. Similarly, Liu et al. (2010) and Hölldobler et al. (2015) highlight the issues arising from semantically deficient domain languages, which may suffer inconsistencies and errors during transformation processes.

Transformation languages, such as the ATL, currently focus on structural correctness, but integrating advanced semantic checking mechanisms would allow them to verify that transformations preserve the behavioral and semantic properties of models. This enhancement could involve embedding formal semantic rules into transformation languages, enabling automatic validation of properties like model consistency, completeness, and adherence to domain-specific semantics. Enhancing the clarity and structure of modeling languages through formal semantics and specific language constraints is elementary for successful model transformations. This approach not only mitigates risks associated with semantic inconsistencies but also ensures that the transformed models fully represent the intended functionalities and interactions.

6.5.3. Issues in property preservation

Preserving non-functional properties during transformations introduces trade-offs that can impact system quality. Property preservation is decisive in model transformation to maintain the integrity and quality of meta-models across dimensions such as reliability, conformance, scalability, traceability, maintainability, performance, and accuracy. Researchers such as He et al. (2016), Jadoon (2024), Batot and Sahraoui (2016), Abate et al. (2019), Zhu and Liu (2009), Dávid et al. (2016), Hülsbusch et al. (2010), Moffett et al. (2013) and Muliawan (2008) highlight the significant challenges in ensuring property preservation during model transformations.

Demuth et al. (2016), Prehofer (2016) and Leblebici et al. (2017) observe that both functional and non-functional properties are affected during transformation and refinement of one model into another. Particularly in large models with extensive underlying functionalities, preserving non-functional properties becomes a complex task due to the interdependencies among various properties.

Stevens (2010) and Jadoon et al. (2020) discuss the issue of model traceability and its impact on verification and maintenance processes, noting that untraced artifacts can significantly hinder these activities.

Abate et al. (2019) address the difficulties in embedding non-functional properties within complex models, proposing automated abstractions to facilitate the generation of source models. However, they caution that transforming these source models into target meta-models often results in the non-persistence of these attributes, especially in low-level languages that lack comprehensive abstract compilation capabilities.

Furthermore, Gerpheide et al. (2016) and Panach et al. (2015) emphasize the challenges of preserving non-functional quality attributes and highlight the necessity of assessing these attributes during model transformations. Fatwanto and Boughton (2008) point out the difficulties in verifying quality attributes, which often involve trade-offs among competing properties.

Addressing the challenges of property preservation requires a basic approach that includes enhancing the methods for tracing model transformations, developing standard techniques for maintaining non-functional properties, and ensuring comprehensive verification mechanisms. To overcome scalability limitations, the development of scalable transformation tools that leverage distributed or parallel processing is a promising direction.

6.5.4. Model verification

Evaluating model transformations is integral to ensuring that transformed models retain the desired properties, yet the generation of test data for this purpose is loaded with challenges that can lead to defects. Ameller et al. (2015), Iqbal and Al-Azzoni (2022) and Abade et al. (2015) highlight the difficulties associated with exhaustive testing in models, especially in scenarios involving regression analysis, where generating exhaustive test data is often impractical. Despite these challenges, verifying that a system model adheres to its specifications remains a fundamental requirement.

Several methods are employed to test model transformations at various stages of MDD, each with its own challenges. Hutchesson and McDermid (2011) and Fleurey et al. (2004) discuss issues like establishing appropriate verification criteria, generating adequate test data, and addressing concerns related to the input space. Both functional and structural testing methods are important, as they help to assess the conformity of model transformations to expected standards.

Calegari and Szasz (2013) note that a thorough verification of model transformations can be resource-intensive and that the quality assurance of the transformation process heavily depends on the chosen verification approach.

Model verification poses significant challenges, but is required to maintain the integrity of transformations in MDD. Effective verification strategies must balance the depth of testing with practical constraints like time and resource availability. Developing advanced tools and techniques that can automate parts of the verification process and enhance the accuracy of the generation of test data is key to improving the reliability of model transformations.

6.5.5. Model complexity

Model complexity is a significant challenge, often aggravated due to poor modularization and unstructured components. Hölldobler et al. (2015), Dávid et al. (2016), de Lara et al. (2017) and Kolahdouz-Rahimi and Lano (2011) indicate that as the number of modules increases, models tend to become unstructured and ambiguous, significantly complicating maintenance and scalability.

Asztalos et al. (2010) discuss the limitations of formal verification, which is often only applicable to specific transformations due to the inherent complexity of models. Verification of software models, requires significant computational resources, which may not always be feasible (Fradet et al., 2023). Hinkel et al. (2019) address the added complexity introduced by platform dependencies, particularly in state-oriented models, which further complicates the transformation process.

Esbai and Erramdani (2015) highlight the challenges of managing complex hardware platforms, addressing that these complexities require the use of automated abstraction methods, which are prone to errors and omissions. Almeida et al. (2019) observed that enhancing or updating existing functionalities often leads to increased model complexity, making these models more difficult to transform effectively.

Le Calvar et al. (2019) emphasize the need for multiple simple views to effectively represent the complex data structures of models, suggesting that simplifying views helps manage complexity. Ege and Tichy (2019) and Magalhães et al. (2016) argue that the complexity of a model negatively impacts its dependability, maintenance, and coherence.

Addressing model complexity requires a distinct approach that includes better modularization techniques, the use of appropriate traceability methods, and the simplification of complex models into manageable components. To overcome complexity limitations, the development of scalable transformation tools that leverage distributed or parallel processing is a promising direction.

Summary

The major challenges addressed are related to model inconsistency, modeling language issues, property preservation, model verification, model complexity, and issues in tool support.

Take-away messages

1. Ensuring system consistency across diverse models requires advanced frameworks and tools to address ambiguities and manage changes effectively.
2. Improved error detection techniques, such as automated and AI-driven testing, are essential to identify residual issues in complex model transformations.
3. Addressing trade-offs in non-functional quality attributes and formalizing modeling languages are significant for enhancing transformation precision and system reliability.

6.5.6. Issues in tool support

The limitations concerning automated property verification tools are well recognized in the field, particularly due to their lack of generality and dependence on the specific type of system model under development. Researchers such as van Amstel et al. (2008), Andraus and Sakallah (2004), Le Calvar et al. (2019), Castellanos et al. (2015), Hinkel et al. (2019), and Rose et al. (2009) have highlighted various aspects of these limitations.

van Amstel et al. (2008) explore model quality metrics and note that ordinary testing tools might not always be suitable for specific model verification needs. Le Calvar et al. (2019) discuss the challenges of handling complex models that present inconsistent instances, which are often poorly managed by existing automated tools. These tools often encounter scalability issues, leading to models that are difficult to maintain and have poor structural integrity, as further elaborated by Andraus and Sakallah (2004).

Castellanos et al. (2015) point to the absence of tool automation to compose reusable models, significantly hindering model reusability. Hinkel et al. (2019) emphasize the lack of specialized tool support to manage complex models, complicating maintenance and replacement. Rose et al. (2009) address the deficiency in tool support for managing model consistency, emphasizing the inadequacy of current tools in tracking and validating model inconsistencies.

The current limitations in tool support for model verification and maintenance highlight a vital need for the development of more robust, flexible, and specialized tools. These tools must not only support a wider range of model types, but also provide better scalability, usability, and precision in handling complex models and maintaining their consistency and reusability.

6.6. RQ 6. What are the benefits of model transformation and property preservation?

Model transformation and property preservation in MDD offer significant advantages that enhance system quality, maintainability, and efficiency. Through thematic analysis, several key benefits were identified, which are classified in subcategories in Table B.15. In the following, we discuss the different categories and refer to the papers highlighting the specific benefits.

6.6.1. Transformation development

The development of model transformation involves a range of best practices that enhance the effectiveness and efficiency of this process. These practices include services-oriented support (Hilken et al., 2018), concrete modeling (Antignac et al., 2018; Magalhães et al., 2020a), modeling abstractions (Westfechtel and Greiner, 2018), constraints (Cuadrado et al., 2017), modeling intents (Lúcio et al., 2016), and transformation chains (Hölldobler et al., 2015).

Hilken et al. (2018) support the reuse of existing service models to compose new ones, aligning with the principles of minimal resource utilization within web service architectures. This approach not only saves resources, but also leverages proven designs to ensure reliability and efficiency.

Antignac et al. (2018) focus on concrete modeling, emphasizing the importance of development-centric approaches, particularly for model-to-code transformations. Such approaches ensure that the transformation process is directly aligned with the development needs, enhancing the practicality and applicability of the models.

Westfechtel and Greiner (2018) discuss the importance of modeling abstractions and high-level system representations to simplify complexity. This is vital for managing large-scale systems and reducing the cognitive load on developers.

Cuadrado et al. (2017) highlight the benefits of employing modeling constraints and rules to ensure the validity and optimal selection of model units for reuse. This structured approach helps maintain consistency and integrity throughout the transformation process.

Lúcio et al. (2016) address the advantages of modeling intents, which serve as main artifacts in transformation processes such as format conversion or code-generation from bi-directional models. These intentions guide the transformation, ensuring that all activities are goal-oriented and aligned with the overall goals of the system.

Lastly, transformation chains, as discussed by Hölldobler et al. (2015), involve a sequence of model transformations that iteratively refine the system until platform-specific application code is generated. This method is supported by the iterative and incremental development models commonly seen in modern software engineering, as addressed by Liu et al. (2005).

6.6.2. Formal approaches

The integration of formal approaches in model transformation is a necessary aspect emphasized in the literature by Combemale et al. (2007), Ameller et al. (2021), Cuadrado et al. (2017), Diskin et al. (2018), Westfechtel and Greiner (2018), Liebrenz et al. (2018), Lukács and Bartha (2022) and Moffett et al. (2013).

Ameller et al. (2021) and Westfechtel and Greiner (2018) endorse the use of mathematical analysis techniques to handle model artifacts. They propose a formal proof of a theorem with predicates and logic as basic tools for verifying the correctness of transformation models. This method ensures that transformations are not only effective, but also mathematically sound.

Cuadrado et al. (2017) discuss the benefits of formal refinement procedures that facilitate the transformation of high-level informal models into detailed formal models. This process ensures the preservation of correctness throughout the transformation, thereby enhancing the integrity of the final model.

Diskin et al. (2018) and Liebrenz et al. (2018) focus on the application of propositional logic in Verilog models, which contributes to their accuracy and reliability. The precision offered by propositional logic is useful in domains where even minor errors can lead to significant consequences.

Combemale et al. (2007) and Moffett et al. (2013) highlight the advantages of using model verification and mathematical logic for the formal verification of process models. This approach provides a robust framework to ensure that the models adhere to the expected behaviors and specifications.

Formal approaches in model transformation provide a foundation for developing transformations that are not only implementable but also verifiable through rigorous mathematical methods. These techniques ensure that the models are robust, reliable, and capable of meeting the high standards required in the application domains. The ongoing development and refinement of these formal methods are essential for the advancement of the MDD field.

6.6.3. Domain-specific languages and tools

DSLs and tools are essential for efficiently handling complex model transformations. DSLs are often built on top of model transformation languages, providing domain-specific abstractions that simplify the definition and management of transformations. While model transformation languages like ATL or QVT define how models are transformed, DSLs guide this process within specific domains, improving clarity, reuse, and productivity in modeling (Fowler, 2010; Wąsowski and Berger, 2023). Their significance is supported in work by Egyed (2010), Esbai and Erramdani (2015), Flake and Mueller (2003), Hölldobler et al. (2015), Hudson and Sprinkle (2011) and Liu et al. (2010).

Hudson and Sprinkle (2011) support the use of DSLs to preserve semantic and syntactic equivalence in models, using conformance rules in these languages to ensure consistency. This approach enhances the interpretability of model transformations by maintaining alignment with domain-specific requirements.

Egyed (2010) emphasizes the role of DSLs in managing model inconsistencies and facilitating reuse, thus improving the robustness and efficiency of MDD processes. Liu et al. (2010) focus on the importance of enhancing semantic coherence in models through DSLs, which are particularly beneficial for complex systems due to their well-defined semantics.

Hölldobler et al. (2015) highlight the advantages of CDTrans, a DSL extension of UML, which provides well-defined domain semantics for accurate model representation. This tool ensures that the transformations accurately reflect the intended meanings and functionalities within a given domain.

Bernal et al. (2019), Flake and Mueller (2003) and Meghzili et al. (2017) stress the importance of formal semantics in DSLs to demonstrate state-oriented models, ensuring that the models adhere to their intended states and behavior. Esbai and Erramdani (2015) analyze the application of semantically rich modeling patterns for precise model transformations, like the model view presenter, which help structure the transformation process to produce more predictable and manageable outcomes.

Dedicated DSLs and tools are essential in model transformation, providing the linguistic and tool-based support needed to effectively handle the complexities of diverse application domains. By embedding precise semantics and syntax rules modified to specific needs, these tools not only streamline the transformation process, but also enhance the overall quality and reusability of the models.

6.6.4. Property preservation

The focus on property preservation is evident in various studies that discuss the importance of maintaining model quality attributes such as conformance, accuracy, reliability, performance, consistency, traceability, maintenance, and reuse. These attributes and the approaches to preserve them are detailed in Table B.15 and are supported by significant research contributions from Almeida et al. (2019), Fleurey et al. (2004), Gadducci et al. (2023), Jadoon et al. (2019, 2020), Klare and Gleitze (2019), Liu et al. (2010), Lopez-Herrejon et al. (2010), Narayanan and Karsai (2008), Paige et al. (2007), Reder and Egyed (2012), Stevens (2020), Wada et al. (2008), Westfechtel and Greiner (2018) and Zhu and Liu (2009).

Gadducci et al. (2023), Lopez-Herrejon et al. (2010) and Paige et al. (2007) emphasize the importance of employing conformance models to accurately measure compliance between source and target models, ensuring bi-directional transformations maintain the integrity of the original specifications. Klare and Gleitze (2019) advocate for the use of a commonalities approach to effectively manage consistency across model transformations.

Almeida et al. (2019) highlight the benefits of model reification, a process that preserves the semantic properties of models by systematically modeling class types and instances. This method ensures that the essence of the original model is retained even through complex transformations.

Jadoon et al. (2019, 2020), Rutle et al. (2018) and Westfechtel and Greiner (2018) discuss the significant role of trace management in transformations, particularly in tracking changes as they propagate from the source to the target models. This traceability management is beneficial for maintaining the lineage and rationale behind each transformation step.

Reder and Egyed (2012) explore performance management strategies that enhance the efficiency and speed of model transformations, ensuring that they can be executed within acceptable time frames and resource allocations.

Meanwhile, Fleurey et al. (2004), Liu et al. (2005), Narayanan and Karsai (2008), Wada et al. (2008) and Zhu and Liu (2009) investigate the benefits of consistency management, which has been validated through various complex model transformations, producing satisfactory results that emphasize the effectiveness of these strategies.

6.6.5. Post-development support

Post-development support is required for the sustainable use of models in software engineering, which includes activities such as model maintenance and reuse (Pietron et al., 2024). This phase is facilitated by various techniques that aid in the generation of modularized model units, promoting efficiency and adaptability. Key contributions in this area include the works by Castellanos et al. (2015), de Lara et al. (2017), Greiner et al. (2017), Guo et al. (2012), Hinkel et al. (2019) and Macías et al. (2019).

Macías et al. (2019), Greiner et al. (2017) and Castellanos et al. (2015) highlight the benefits of reuse-based modeling, like the prioritization of existing models, libraries, and patterns to reduce development time and increase efficiency. Using pre-existing resources, organizations can minimize the effort required in model creation and focus on enhancing and adapting models to new requirements.

de Lara et al. (2017) discuss the utilization of typing requirement models that record generic specifications. These models are highly valued in the post-development phases as they embody best practices for reuse, characterized by their independence and ease of integration. This approach eases the adoption of models across different projects and contexts, enhancing their utility, and reducing redundancy.

Guo et al. (2012) explore the use of feature models to effectively evolve existing models. These models allow for the addition of necessary features and support the ongoing evolution of the model's capabilities, ensuring that they remain relevant and functional over time.

Hinkel et al. (2019) focus on modularization techniques that enable easier remodeling, maintenance, and management of model transformations. Modularization not only simplifies the complexity inherent in large models, but also supports better management of changes and updates.

Post-development support, particularly in the form of model maintenance and reuse, is essential for the long-term sustainability of MDD practices. Techniques that promote modularity and reuse not only conserve resources, but also enhance the adaptability and longevity of software models.

Summary
The key benefits highlighted in the studies include advancements in transformation development, the adoption of formal verification methodologies, the use of DSLs, effective property preservation practices, robust post-development support, and improved variability management.

6.6.6. Variability management in SPL models

Variability management in SPLs is a key aspect of MDD, enabling efficient customization and reuse of features across different products within a product line. Researchers such as Bilic, Feichtinger, Hutchesson, Lopez-Herrejon, Santos, and Vierhauser have extensively discussed advanced approaches to managing inconsistencies within SPL (Bilic et al., 2020; Feichtinger and Rabiser, 2020; Hutchesson and McDermid, 2011; Lopez-Herrejon et al., 2010; Lopez-Herrejon and Egyed, 2012; Santos et al., 2015; Vierhauser et al., 2010b,a).

Variability modeling techniques, such as feature modeling, allow developers to represent the commonalities and differences within an SPL, facilitating systematic reuse (Lopez-Herrejon and Egyed, 2012; Lopez-Herrejon et al., 2010; ter Beek et al., 2019; Di Ruscio et al., 2017). Feature modeling supports variability by using meta-models to transform features into decision models, facilitating configuration and customization processes. Bilic et al. (2020), Feichtinger and Rabiser (2020), Hutchesson and McDermid (2011), Santos et al. (2015), Vierhauser et al. (2010b) and Vierhauser et al. (2010a) highlight the benefits of using variability models to enhance the flexibility and adaptability of SPLs. These models allow organizations to tailor software products to meet specific customer needs without extensive redevelopment, thereby reducing time to market and improving product quality.

TRAVART is a transformation approach that enables converting and experimenting with various variability models, including feature models, decision models, and OVMs, facilitating model comparison, interoperability, and unification in variability modeling (Feichtinger et al., 2021).

The management of variability within SPLs is significant in maximizing the reuse of software artifacts and enhancing the adaptability of the products. Employing advanced modeling techniques such as variability and feature modeling not only helps in managing inconsistencies, but also supports the systematic customization and extension of product lines (ter Beek et al., 2019; Benavides et al., 2025).

6.6.7. Model verification and validation

Researchers explored various strategies to support the model verification and validation process (Oakes et al., 2015; Combemale et al., 2007; Beyer and Lemberger, 2017, 2024; Hilken et al., 2018; Asztalos et al., 2010; Diskin et al., 2018; Daw et al., 2014; Andraus and Sakallah, 2004; Calegari and Szasz, 2013; Leuschel et al., 2011). Hilken et al. (2018) endorse the use of classification terms for model transformation verification, providing concrete extensions to understand model components. Diskin et al. (2018) propose hierarchical refinements for testing models. Calegari and Szasz (2013) focus on semantic automation to detect language inconsistencies, using automated reasoning and analysis techniques. Beyer and Lemberger (2017, 2024) and Daw et al. (2014) emphasize the benefits of model verification for formal verification, ensuring reliable and accurate model transformations. Andraus and Sakallah (2004) highlight automatic abstractions to verify Verilog models. Combemale et al. (2007) emphasize the benefits of using propositional logic for transformation verification. Oakes et al. (2015) and Leuschel et al. (2011) highlight the advantages of reliable and scalable verification tools like ProB and DSLTrans. Asztalos et al. (2010) work on the conformance verification of models, emphasizing the significant role of predicate logic in verifying complex model transformations.

7. Threats to validity

In this section, we identify the threats to the validity of this study. We tried to mitigate them based on the approaches provided by Apostolos et al. for secondary studies (Ampatzoglou et al., 2019).

Study Selection Validity: The primary threats to validity in the selection of the study include:

1. The formulation of the search string and its possibility of failure in providing all the relevant papers.
2. The lack of maturity of the search engines of libraries.
3. Applying inclusion/exclusion criteria and quality criteria that might be subjective.

To mitigate these threats, the following approaches are used:

1. We performed snowballing to include those papers not covered by the search string.
2. We performed the search multiple times in four different rounds, considering various engines.
3. We defined objective criteria based on the earlier work and piloted them; we discussed them among the authors when issues arose.

Data Validity: The major threats to the validity of data are:

1. Bias in data extraction due to subjectivity.
2. Bias of classification schemes that provide the extraction of specific data.

To mitigate these threats, the following approaches are used:

1. The expert data extractors crosschecked the results during the pilot study.
2. Data classification schemes from previous studies were revised and new schemes were piloted to cover the study content.

Research Validity: To certify research validity, a defined research protocol is followed. The search and data extraction process is reported at each step. All natural results and analyses are shared with the researchers to ensure replication and independent investigation. The validity of the research is supplemented by iterating the search process four times, provided that the research protocol could be replicated.

8. Conclusion

We presented an SLR encompassing 202 studies from the last two decades, focusing on model transformation and property preservation in MDD. The selected studies are categorized by transformation types, with an in-depth discussion of various techniques for verifying and preserving model properties.

The analysis highlights the prevalent use of formal methods and case-driven evaluations, emphasizing the importance of theoretical rigor and practical validation. However, experimental studies, particularly those involving human subjects, are underrepresented, limiting insights into the practical usability and adoption of tools in real-world scenarios. No less than 74 studies (37%) rely on manual verification approaches, revealing a significant gap in automated tools. Key properties, such as model consistency, non-functional behavior, and maintainability, are unevenly addressed, indicating the need for a more comprehensive and balanced research focus.

Take-away messages

1. Formal verification methodologies and DSLs are essential for achieving precise and reliable model transformations, ensuring both semantic coherence and syntactic accuracy.
2. Property preservation practices and post-development support, such as modularization and reuse, enhance software model adaptability and reduce redesign efforts.
3. Effective variability management in SPLs enables systematic reuse and customization, improving responsiveness to diverse market and customer needs.

Model transformations demonstrate substantial benefits through formal verification, robust property preservation strategies, and post-development practices like reuse and modularization. These contribute to enhancing the robustness, adaptability, and sustainability of the software models. Additionally, variability modeling facilitates systematic artifact reuse, enabling efficient responses to diverse market needs.

Despite these advancements, significant challenges remain, including semantic inconsistencies, difficulties in preserving non-functional properties, and scalability limitations. The lack of adaptable and comprehensive tools to handle complex and large-scale systems further exacerbates such challenges.

Future research should focus on developing scalable and flexible tools that integrate automated verification processes across diverse model types. Enhancing formal semantics, refining variability management techniques, and employing advanced property preservation methodologies are essential to addressing the current gaps. By exploiting advanced AI methodologies, it is possible to analyze and manage the complexities of model transformations more effectively, enabling innovative solutions to existing challenges. Incorporating enforcement techniques within machine learning algorithms could refine verification processes, enabling more accurate and reliable property preservation for given models. Bridging the gap between theoretical research and industrial applications through empirical studies and real-world validation will strengthen the reliability and adaptability of model transformations, aligning them with the demands of modern software systems.

CRediT authorship contribution statement

Gullelala Jadoon: Writing – review & editing, Writing – original draft, Visualization, Validation, Investigation, Formal analysis, Data curation. **Maurice H. ter Beek:** Writing – review & editing, Supervision. **Alessio Ferrari:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Challenges of model transformation and property preservation

Table B.14 presents the challenges associated with the preservation of non-functional quality attributes in MDD, drawing on classified and analyzed data using thematic analysis (Braun and Clarke, 2021; Bano and Zowghi, 2015), as discussed in Section 5.9.

Appendix B. Benefits of model transformation and property preservation

Table B.15 presents the benefits associated with the preservation of non-functional quality attributes in MDD, drawing on classified and analyzed data using thematic analysis (Braun and Clarke, 2021; Bano and Zowghi, 2015), as discussed in Section 5.10.

Table B.14
Challenges of model transformation and property preservation.

Challenges	Challenges category	Description	Extracted from Studies	Frequency N=202
System Inconsistency	Ambiguity	An unstructured modeling artifact results in ambiguity and a lack of formal semantics.	1, 4, 5, 12, 45, 46, 68, 72, 78, 87	10
	Lack of deep analysis	The concrete metamodels are not analyzed deeply, so the modeling semantics are weak.	1, 4, 5, 14, 41, 42, 53, 73, 84	9
	Unstructured systems	Different modeling languages may lead to unstructured or inconsistent systems due to varying standards and approaches.	4, 5, 14, 46, 67, 68, 70, 72, 74, 77, 78, 81, 86, 87, 132	15
	Redundancy of properties	Redundancy of properties as a result of transformation affects the structure and consistency of the meta-models.	51, 53, 71, 78, 84	5
	Output vs. consistency	Sometimes complete consistency is not applicable in practical systems because the focus is on the output behavior.	1, 4, 5, 46, 69, 70, 71, 72, 73, 74, 75, 77, 78, 80, 81, 82	16
	Changeability issues	Inconsistencies in models arise whenever any alterations are made to the models.	1, 5, 12, 42, 46, 71, 83, 84, 86, 131	10
	Inconsistency in variability models	Annotated variability models are often prone to errors because defining every possible variant of a system model in an annotated way may result in conflicting models.	1, 4, 12, 44, 46, 69, 70, 71, 72, 73, 74, 75, 80, 82	13

(*continued on next page*)

Table B.14 (*continued*).

	Poor model analysis	This typically involves insufficient analysis of key aspects such as meta-model specifications, verification processes, and constraint management.	14, 75, 86, 87	4
	Translation inconsistency	In MDD, the system properties get inconsistent when a higher-level abstract transformation is translated into a lower-level transformation.	45, 46, 67, 70, 73, 77, 81	7
Error Detection	Poor error detection	Errors in annotated models often remain undetected after several model iterations.	33, 44, 75, 83, 84, 86, 87, 201	8
	Error-prone	For testing model transformations, generating test data is difficult and error sensitive.	8, 47, 83, 84	4
Modeling Languages	Lack of Multi-level semantics	Most of the model-driven approaches support conventional two-level modeling. One must preserve the multi-level semantics when transforming a multi-level model into a two-level one.	13, 28, 36, 85	4
	Lack of language support	Modeling languages do not mostly support the transformation of UML models to other Platform Specific Models.	15, 17, 52, 56, 182	5
	Lack of semantic support	Sometimes model semantics are embedded in models with complex domain-specific objects that are difficult to simplify.	13, 16, 23, 28, 56, 63, 82, 85, 181	9
	Lack of specification languages	The lack of standardized specification languages can result in ambiguous models and inconsistent system definitions.	23, 28, 36, 45, 48, 86, 87	7
	Lack of semantic verification	The lack of semantic verification can lead to misunderstandings and errors in how system models are interpreted and implemented.	16, 56, 63, 71, 72, 73, 74, 75, 79, 82, 85	11
Property Preservation	Non-persistent properties	Transforming source models to target meta-models, properties become non-persistent in low-level languages lacking full abstract compilation.	25, 65, 82	3
	Model conformance preservation	The lack of model conformance preservation can result in discrepancies between the model and its intended properties, leading to potential errors.	37, 39, 40, 49, 58, 76, 77	7
	Preservation of quality attributes	The non-preservation of quality attributes can compromise reliability, performance, and overall effectiveness of the system.	37, 39, 40, 65, 66, 67, 68, 76, 77, 130	10
	Lack of property assurance	Assurance of the critical properties of a system, using the hierarchical model transformation, is considered an important research challenge in MDD.	40, 49, 58, 64, 66, 76, 83, 84	8
	Lack of property identification	Identification of basic model properties is a challenging task while transforming models.	37, 49, 58, 65, 66, 183	6
Property Verification	Lack of regression testing	In regression testing of model transformations, exhaustive test generation is impossible, so testing remains incomplete.	26, 58, 54, 59, 60, 61, 188	7
	Lack of property verification	A set of properties needs verification to comply with the expected system behavior which is challenging.	26, 60, 61, 71, 76, 86, 87	7
	Manual property verification	Manual property verification can lead to overlooked errors and reduce the overall accuracy of the model transformations.	26, 59, 76, 86	4

(*continued on next page*)

Table B.14 (*continued*).

	Cost of property verification	Property verification cost can be significant, impacting both the time and resources required for ensuring system accuracy.	58, 59, 60, 61, 76	5
	Lack of model analysis	Without thorough analysis, correctness of models or meta-models cannot be guaranteed.	54, 59, 61, 62, 71, 76	6
	Lack of verification criteria	Model verification process followed by some distinct issues like verification criteria, test cases, and test data generation.	61, 71, 76, 80, 109	5
Informal Requirements	Mismanaged requirements	There are limitations in meta-model coincidence because of inconsistent requirements.	17, 18, 53, 66	4
	Manual requirements	The requirements specifications are mostly documented manually, which creates complexity when converted to model-driven design and architecture.	18, 86, 87	3
System Complexity	Platform dependency	In some models, state machine checking is always limited to finite platform-specific models.	7, 86, 178	3
	Lack of automated abstractions	Complex hardware needs automated abstraction methods, as manual model abstraction often results in errors and omissions.	8, 10, 20, 22	4
	Transformation complexity	Complexity of the model transformation increase with the number of modules.	10, 23, 83, 84, 86, 87, 174	7
	Reengineering complexity	Adding and updating functionality results in more complex models that are transformed using UML, Model View Presenter Pattern (MVP), QVT, and many other languages.	20, 22, 23	3
	Model complexity	For larger systems, generated models are complex and require multiple simple views to represent the source model data.	22, 78, 83, 84	4
	Lack of modularization	Modularity of the transformed models is affected whenever any manual operations are performed on the complex models.	31, 83, 84, 86	4
	Lack of maintainability	Major limitations when transforming complex models: reuse, portability and maintainability.	20, 22, 50, 83, 84	5
	Variability management	Variability models have complexities due to variations in features of product line models.	44, 55, 69, 70, 71, 72, 73, 74, 75	9
System Quality	Limitations of manual verification	Manual verification is always tedious and less effective.	6, 10, 11, 24, 75, 184	6
	Performance issue	Performance issues arise regarding computational power for composing complex models.	10, 66	2
	Lack of conformance	Sometimes transformed models do not conform to pre- or post-conditions of the other models in the transformation chain, generating different errors.	6, 11, 24, 27, 200	5
	Challenges of manual traceability	Manual approaches to maintain trace and manipulate the transformation are cumbersome.	6, 24, 63, 83, 84	5
	Lack of maintainability	Issues related to productivity and maintainability of models arise when models get complex.	27, 33, 48, 83, 84	5
	Poor resource usage	Software model checking can be used for the exhaustive model checking, but it requires high computational power.	45, 86, 87, 173	4

(*continued on next page*)

Table B.14 (*continued*).

Tool Support	Lack of specialized tool support	No tool support is available to verify the complex services provided by models.	1, 2, 3, 9, 27, 28, 75, 86, 87	9
	Lack of tool scalability	Ordinary testing tools are sometimes not applicable to complex model verification.	2, 3, 6, 9, 27, 52, 75	7
	Lack of tool efficiency	The lack of tool efficiency hampers the effectiveness of processes, leading to delays and potential inaccuracies.	3, 75, 187, 192	4
	Lack of tool optimization	Tool optimization is needed to deal with hierarchical and convoluted models.	9, 52	2
	Lack of automated tool support	The lack of automated tools to manage complex requirements, results in problem solving by manually updating the generated code of models, leading to other ambiguities.	9, 17, 27, 31, 75, 86, 186	7
	Lack of tools for multi-level models	Multi-level modeling lacks the definition of different concepts involved, resulting in the absence of automated tools for managing multi-level models.	3, 9, 17, 28, 30, 103	6
	Insufficient consistency tools	Consistency management approaches lack proper tool support, tracking, evaluation, and validation mechanisms.	3, 9, 12, 57, 69, 75, 185	7
	Large input space	Some tools do not cope with the problem of large input space and exhaustive testing.	9, 86, 87	3

Table B.15
Benefits of model transformation and property preservation.

Benefits	Benefits category	Description	Extracted from studies	Frequency N=202
Transformation Development	Service-Oriented Architecture	Service-Oriented Architecture: existing services of models are reused to compose new ones.	1, 4, 7, 8, 10, 19, 36	7
	Concrete modeling	Concrete modeling allows to create a detailed representation of a system or software product using a specific modeling language or tool focusing on model-to-code transformation.	4, 8, 12, 22, 50, 87, 88, 148, 149	9
	UML modeling	UML modeling provides a standardized visual language for specifying, constructing, and documenting the components of software systems.	7, 30, 37, 87, 140	6
	Modeling abstractions	Modeling abstractions are high-level representations of a system that simplify the complexity and focus on certain significant aspects of the system.	8, 12, 25, 129, 154	5
	Model constraints	Model constraints define rules and limitations ensuring model consistency and correctness within the specified domain.	10, 15, 36, 87, 88, 130	6
	Conceptual modeling	A hierarchical concept meta-model is generated, defining all common concepts of a transformation chain instead of modeling them repeatedly in every model transformation.	12, 87, 88, 117, 142, 150, 158, 172	8
	Transformation rules	Transformation rules indicate how one model is converted into another, ensuring consistency and alignment with the intended design.	10, 15, 20, 36, 87, 88, 126, 198	8

(*continued on next page*)

Table B.15 (*continued*).

	Platform-independent models	Platform-independent models are not tied to a specific technology or implementation. They can be used to represent a system in a way that is independent of the underlying platform or technology.	17, 137, 199	3
	Intensional views	Such views present important instances about source models; they may contain automated data suitable for model reuse.	22, 40	2
	Transformation chain	In MDD, a system is transformed using a chain of models until application code is generated.	23, 50, 87	4
	Transformation intents	Transformation intents are the goals or objectives a model transformation aims to achieve, such as converting a model from one format to another or generating code from a model.	49, 87	3
	Agile transformation	Such transformation uses Agile methodologies and practices in an organization to improve the efficiency and flexibility of the software development process.	52, 87, 88, 116, 167, 175	6
Formal Approaches	Formalization	Formal theorem proving verifies the transformation models using predicates and logic.	1, 5, 8, 9, 11, 26, 40, 48, 180	9
	Formal refinement	Formal refinement is transforming a high-level, informal model into a more detailed and formal model by adding more constraints while preserving the correctness of the model.	5, 48, 125, 151, 159, 164, 165, 171, 179	9
	Propositional logic	Propositional logic of Verilog assists in property verification and correctness.	7	1
	Accuracy	Formal reasoning is an effective way to check system model accuracy using propositional logic before code generation.	9, 143	2
	Defect detection	Defect detection refers to identifying and locating errors or inconsistencies in models, by comparing the model to a set of predefined rules or by applying formal verification techniques.	11, 26, 143, 191	4
	Property assurance	A system possesses an assured property whenever a particular assurance view of the system satisfies acceptance criteria posed as assurance constraints.	16, 40, 48, 65, 66, 67, 68, 69	8
	Defined semantics	Defined semantics refers to the precise and well-defined meaning of the concepts and relationships within the model, enabling automated reasoning and analysis techniques to ensure the model's consistency and correctness.	5, 48, 64, 70, 80, 181, 182	7
Model Checking	Model Validation	Model checking is an important means of verifying the correctness of models using a formal methodology.	2, 9, 30, 43, 139	5
	Efficiency	Model checkers are efficient and accurate ways of verification, used to verify models by using the least resources.	2, 9, 43, 193	4
	Model verification	Model verification is mathematically proving that a software model meets a set of formal specifications, ensuring that the model is consistent, correct, and free of errors.	9, 30, 48, 145, 146, 147, 156, 160, 161, 162, 166, 197	12

(*continued on next page*)

Table B.15 (*continued*).

	De-facto standard	A de-facto standard is a widely accepted and dominant modeling language or method that has become a standard through widespread use and acceptance in the software development industry rather than by formal standardization or legislation.	2, 9, 30, 43, 48	5
	Testing adequacy	By defining test adequacy criteria, test cases are generated, following a black-box, gray-box, or white-box testing approach.	9, 30, 54, 118, 119, 134	6
Domain Benefits	Language definition	DSLs are best to model complex systems due to the well-defined semantics of a modeling language.	3, 21, 28	3
	Domain semantics	For common models, DSLs, such as CDTrans, are very important. These DSLs contain well-defined domain semantics.	3, 21, 23, 28	4
	Domain-specific tools	The use of DSLs is promising for developing tools tailored to specific problem spaces, effectively diminishing the complexity of hand-made software.	21, 28, 85	3
Automation	Tool automation	Automated tools play an important role in verifying models for specific languages, which is impossible manually.	6, 21, 28, 47, 176	5
	Search automation	Automated search-based test data generation approaches are suitable for model transformations assisting to verify models.	47	1
	Semantic automation	This refers to automated reasoning and analysis techniques to check formal model correctness, consistency and completeness based on the semantics.	13, 16, 48, 80	4
Tool Support	Tool scalability	Some tools scale for transformed models and help modeling and verifying large systems.	6, 29, 34, 47, 201	5
	Enterprise tool support	Enterprise tool support provides comprehensive software solutions to manage, automate, and optimize complex models of business processes across an organization.	6, 29, 55	3
	Modeling tool support	Advanced tools can detect a wide range of non-trivial problems in model transformations using constraint solving to improve the analysis precision.	34, 47, 133, 135, 192	5
	Verification tool support	A wide range of verification tools are available, ranging from simple verifiers to advanced formal tools for assisting model transformation.	47, 58, 60, 61, 133	5
Non-Functional Property Preservation	Conformance model	A conformance model is a model that describes the requirements and constraints a system must meet to be considered compliant with a specific standard, regulation, or specification. It is used for testing, validation, and certification.	13, 16, 24, 33, 36, 41, 42, 51, 53, 55, 56, 62, 73, 77, 90, 194	16
	Consistency preservation	The commonalities approach for preserving consistency resulted in increased understandability and reduced errors.	13, 22, 24, 36, 41, 42, 56, 61, 62, 76, 77, 78, 79, 81, 82, 83, 141, 144, 152	19
	Semantic preservation	Model semantics are preserved by systematically modeling the class type and its instance using reification (converting any abstract idea into a model).	13, 16, 19, 33, 36, 48, 56, 60, 62, 80, 83, 86	12

(*continued on next page*)

Table B.15 (*continued*).

	Traceability preservation	Change in the model is propagated from the source to the target model in several transformations using model traceability management.	24, 41, 83, 84, 85, 128	6
	Performance management	Performance management approaches are proposed for models to increase the performance of validating design rules.	46, 136, 153, 195	4
	Consistency mapping	Consistency mapping is a process to determine a model's consistency by comparing it to a set of predefined rules or constraints and identifying any inconsistencies or errors that may exist in the model.	51, 53, 55, 73, 74, 75, 76, 77	8
Post Development Benefits	Modulariza-tion	Modularization breaks a software system into smaller independent modules. Modularization of models improves maintainability and reuse.	5, 10, 18, 21, 28, 29, 31, 41, 44, 50	9
	Reuse-based modeling	Reuse-based modeling is a methodology for creating software models that emphasizes the reuse of existing models, libraries, and patterns to reduce development time and increase efficiency.	1, 10, 15, 36, 44, 50	6
	Improved understand-ability	One can use some defined high-level formalized language that can provide full understandability of the specifications to make them more usable.	18	1
	Reduced complexity	To make models as precise, simple, and reusable as possible, one augments DSLs with concepts from multilevel modeling, where the number of abstraction levels is not limited.	28, 21	2
	Changeability	Changeability of a software model refers to the ease and flexibility with which a model can be modified, adapted, or extended to accommodate changes in the requirements or environment of the system it represents.	29, 70	2
	Maintainabil-ity support	MDD enables domain experts to specify, validate, and maintain software systems without requiring advanced programming skills.	18, 21, 28, 41, 87	5
SPLs	AADL variability support	AADL is an extensively used transformation language because of its scalability and variability characteristic support in critical systems.	14, 24, 35, 38, 76, 89, 91, 96, 97, 155, 157, 163, 169	13
	Multi-variant variability management	Multi-variant model transformation in SPL engineering is an important approach to integrate variability in the specified product model.	14, 35, 44, 56, 76, 92, 99, 100	8
	Variability modeling	Variability modeling explicitly represent variability in dedicated models that describe the common and variable characteristics of a set of (software) systems.	35, 44, 57, 60, 71, 72, 73, 75, 93, 98, 202	11
	Reuse variability support	SPL models have become one of the most prominent ways to promote the systematic reuse of software artifacts.	57, 60, 73, 74, 94, 95, 170	7
Maintenance	Graphical defect localization	There is another method that focuses on graphical models to locate errors and defects in transformed models, using visualizations at an abstract level.	11, 32, 124	3
	System refinement	Model transformations use the step-by-step transform method for constant system refinement and upgradation.	32, 44, 57, 62	4

(*continued on next page*)

Table B.15 (*continued*).

	Fault localization	Fault localization identifies a system's specific location or component causing an error or malfunction.	32, 121	2
Property Preservation	Symbolic property preservation	Verify pre/postcondition contracts on the resulting DSLTrans specification with symbolic-execution property provers.	14, 36, 42, 43, 58, 60, 82	7
	Iterative property preservation	We presented a new technique to verify the correctness of complex models that result from iterative refinement through model transformation.	26, 27, 36, 39, 45, 47, 59, 79, 81, 86, 114, 115, 196	13
	Logical property preservation	This description is based on first-order logic; therefore, if deduction rules are provided, a reasoning system can automatically use an assertion set to derive additional assertions verifying model transformations.	36, 42, 45, 59, 62, 77, 83, 84, 85, 112, 113, 202	12
Verification	Test case prioritization	Test case prioritization techniques are needed to rank test cases and help the tester during regression testing to be more efficient.	47, 58, 60, 104, 105, 106, 120, 127, 138	9
	Static analysis	Static analysis is the analysis of software or a system by examining its source code or model without executing it to identify potential bugs or vulnerabilities.	58, 60, 102, 122	4
	Functional testing	Functional testing validates the software system against the functional requirements or specifications. Functional tests try each software application function by providing appropriate input and verifying the output against the functional requirements.	47, 60, 61, 107, 108, 110, 123	6

Data availability

All the data produced is available in our permanent repository (Jadoon, 2024–2025): https://zenodo.org/records/15227409.

References

Abade, A., Ferrari, F., Lucrédio, D., 2015. Testing M2T transformations: A systematic literature review. In: Proceedings of the 17th International Conference on Enterprise Information Systems. ICEIS'15, 2, SciTePress, pp. 177–187. http://dx.doi.org/10.5220/0005378501770187.

Abate, C., Blanco, R., Garg, D., Hritcu, C., Patrignani, M., Thibault, J., 2019. Journey beyond full abstraction: Exploring robust property preservation for secure compilation. In: Proceedings of the 32nd Computer Security Foundations Symposium. CSF'19, IEEE, pp. 256–271. http://dx.doi.org/10.1109/CSF.2019.00025.

Abbas, M., Rioboo, R., Ben-Yelles, C.-B., Snook, C.F., 2021. Formal modeling and verification of UML activity diagrams (UAD) with FoCaLiZe. J. Syst. Archit. 114, 101911. http://dx.doi.org/10.1016/j.sysarc.2020.101911.

Aït Ameur, Y., Aït-Sadoune, I., 2012. Stepwise development of formal models for web services compositions: Modelling and property verification. In: Liddle, S.W., Schewe, K., Tjoa, A.M., Zhou, X. (Eds.), Proceedings of the 23rd International Conference on Database and Expert Systems Applications. DEXA'12, In: LNCS, vol. 7446, Springer, p. 9. http://dx.doi.org/10.1007/978-3-642-32600-4_2.

Almasri, N., Korel, B., Tahat, L., 2022. Verification approach for refactoring transformation rules of state-based models. IEEE Trans. Softw. Eng. 48 (10), 3833–3861. http://dx.doi.org/10.1109/tse.2021.3106589.

Almeida, J.P.A., Musso, F.A., Carvalho, V.A., Fonseca, C.M., Guizzardi, G., 2019. Preserving multi-level semantics in conventional two-level modeling techniques. In: Companion Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems Companion. MoDELS'19, IEEE, pp. 142–151. http://dx.doi.org/10.1109/MODELS-C.2019.00025.

Alshalalfah, A.-L., Ait Mohamed, O., Ouchani, S., 2023. A framework for modeling and analyzing cyber-physical systems using statistical model checking. Internet Things 22, 100732. http://dx.doi.org/10.1016/j.iot.2023.100732.

Ameller, D., Burgués, X., Collell, O., Costal, D., Franch, X., Papazoglou, M.P., 2015. Development of service-oriented architectures using model-driven development: A mapping study. Inf. Softw. Technol. 62, 42–66. http://dx.doi.org/10.1016/J.INFSOF.2015.02.006.

Ameller, D., Franch, X., Gómez, C., Martínez-Fernández, S., Araújo, J., Biffl, S., Cabot, J., Cortellessa, V., Fernández, D.M., Moreira, A., Muccini, H., Vallecillo, A., Wimmer, M., Amaral, V., Böhm, W., Bruneliere, H., Burgueño, L., Goulão, M., Teufl, S., Berardinelli, L., 2021. Dealing with non-functional requirements in model-driven development: A survey. IEEE Trans. Softw. Eng. 47 (4), 818–835. http://dx.doi.org/10.1109/TSE.2019.2904476.

Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., Chatzigeorgiou, A., 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Inf. Softw. Technol. 106, 201–230. http://dx.doi.org/10.1016/J.INFSOF.2018.10.006.

van Amstel, M.F., Lange, C.F.J., van den Brand, M.G., 2008. Metrics for analyzing the quality of model transformations. In: Proceedings of the 12th ECOOP Workshop on Quantitative Approaches on Object-Oriented Software Engineering. QAOOSE08'08, pp. 41–51.

Andraus, Z.S., Sakallah, K.A., 2004. Automatic abstraction and verification of verilog models. In: Proceedings of the 41st Design Automation Conference. DAC'04, ACM, pp. 218–223. http://dx.doi.org/10.1145/996566.996629.

Antignac, T., Scandariato, R., Schneider, G., 2018. Privacy compliance via model transformations. In: Workshops Proceedings of the 3rd European Symposium on Security and Privacy. EuroS&P'18, IEEE, pp. 120–126. http://dx.doi.org/10.1109/EUROSPW.2018.00024.

Apel, S., Batory, D., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, http://dx.doi.org/10.1007/978-3-642-37521-7.

Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G., 2010. Henshin: Advanced concepts and tools for in-place EMF model transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (Eds.), Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems. MoDELS'10, In: LNCS, vol. 6394, Springer, pp. 121–135. http://dx.doi.org/10.1007/978-3-642-16145-2_9.

Arrassen, I., Esbai, R., Meziane, A., Erramdani, M., 2012. QVT transformation by modeling: From UML model to MD model. In: Proceedings of the 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications. SETIT'12, IEEE, pp. 86–91. http://dx.doi.org/10.1109/SETIT.2012.6481895.

Asztalos, M., Lengyel, L., Levendovszky, T., 2010. Towards automated, formal verification of model transformations. In: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation. ICST'10, IEEE, pp. 15–24. http://dx.doi.org/10.1109/ICST.2010.42.

Ayala, I., Amor, M., Fuentes, L., 2023. Analysis and optimization of SPL products using goal models. In: Proceedings of the 31st International Requirements Engineering Conference. RE'23, IEEE, pp. 89–99. http://dx.doi.org/10.1109/RE57278.2023.00018.

Babaei, M., Dingel, J., 2023. Efficient regression testing of distributed real-time reactive systems in the context of model-driven development. Softw. Syst. Model. 22 (5), 1565–1587. http://dx.doi.org/10.1007/s10270-023-01086-5.

Bachmann, F., Clements, P.C., 2005. Variability in Software Product Lines. Tech. Rep.CMU/SEI-2005-TR-012, Carnegie Mellon University, URL https://insights.sei.cmu.edu/library/variability-in-software-product-lines/.

Bano, M., Zowghi, D., 2015. A systematic review on the relationship between user involvement and system success. Inf. Softw. Technol. 58, 148–169. http://dx.doi.org/10.1016/J.INFSOF.2014.06.011.

Baouya, A., Bennouar, D., Mohamed, O.A., Ouchani, S., 2015. A quantitative verification framework of SysML activity diagrams under time constraints. Expert Syst. Appl. 42 (21), 7493–7510. http://dx.doi.org/10.1016/j.eswa.2015.05.049.

Barkowsky, M., Giese, H., 2024. Incremental model transformations with triple graph grammars for multi-version models and multi-version pattern matching. Softw. Syst. Model. http://dx.doi.org/10.1007/s10270-024-01238-1.

Basile, D., ter Beek, M.H., Ferrari, A., Legay, A., 2019. Modelling and analysing ERTMS L3 moving block railway signalling with Simulink and Uppaal SMC. In: Larsen, K.G., Willemse, T.A.C. (Eds.), Proceedings of the 24th International Conference on Formal Methods for Industrial Critical Systems. FMICS'19, In: LNCS, vol. 11687, Springer, pp. 1–21. http://dx.doi.org/10.1007/978-3-030-27008-7_1.

Batot, E., Sahraoui, H., 2016. A generic framework for model-set selection for the unification of testing and learning MDE tasks. In: Proceedings of the 19th International Conference on Model Driven Engineering Languages and Systems. MoDELS'16, ACM, pp. 374–384. http://dx.doi.org/10.1145/2976767.2976785.

Batot, E., Sahraoui, H., Syriani, E., Molins, P., Sboui, W., 2016. Systematic mapping study of model transformations for concrete problems. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. MODELSWARD'16, SciTePress, pp. 176–183. http://dx.doi.org/10.5220/0005657301760183.

Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y., Mottu, J.-M., 2010. Barriers to systematic model transformation testing. Commun. ACM 53 (6), 139–143. http://dx.doi.org/10.1145/1743546.1743583.

ter Beek, M.H., Schmid, K., Eichelberger, H., 2019. Textual variability modeling languages: An overview and considerations. In: Proceedings of the 23rd International Systems and Software Product Line Conference. SPLC'19, 2, ACM, pp. 82:1–82:7. http://dx.doi.org/10.1145/3307630.3342398.

Benavides, D., Sundermann, C., Feichtinger, K., Galindo, J.A., Rabiser, R., Thüm, T., 2025. UVL: Feature modelling with the universal variability language. J. Syst. Softw. 225, 112326. http://dx.doi.org/10.1016/J.JSS.2024.112326.

Bernal, A., Cambronero, M.E., Núñez, A., Cañizares, P.C., Valero, V., 2019. Improving cloud architectures using UML profiles and M2T transformation techniques. J. Supercomput. 75 (12), 8012–8058. http://dx.doi.org/10.1007/s11227-019-02980-w.

Berramla, K., Deba, E.A., Senouci, M., 2015. Formal validation of model transformation with coq proof assistant. In: Proceedings of the 1st International Conference on New Technologies of Information and Communication. NTIC'15, IEEE, pp. 1–6. http://dx.doi.org/10.1109/NTIC.2015.7368755.

Bessifi, M., Younes, A.B., Ayed, L.B., 2021. BPMN2EVENTB supporting transformation from BPMN2.0 to Event B using Kermeta. In: Zhang, Y.-D., Senjyu, T., So-In, C., Joshi, A. (Eds.), Proceedings of the 5th International Conference on Smart Trends in Computing and Communications. SmartCom'21, In: LNNS, vol. 286, Springer, pp. 247–255. http://dx.doi.org/10.1007/978-981-16-4016-2_24.

Beyer, D., Lemberger, T., 2017. Software verification: Testing vs. Model checking – a comparative evaluation of the state of the art. In: Strichman, O., Tzoref-Brill, R. (Eds.), Proceedings of the 13th International Haifa Verification Conference. HVC'17, In: LNCS, vol. 10629, Springer, pp. 99–114. http://dx.doi.org/10.1007/978-3-319-70389-3_7.

Beyer, D., Lemberger, T., 2024. Six years later: testing vs. model checking. Int. J. Softw. Tools Technol. Transf. 26 (6), 633–646. http://dx.doi.org/10.1007/s10009-024-00769-8.

Bilic, D., Carlson, J., Sundmark, D., Afzal, W., Wallin, P., 2020. Detecting inconsistencies in annotated product line models. In: Proceedings of the 24th Systems and Software Product Line Conference. SPLC'20, ACM, pp. 20:1–20:11. http://dx.doi.org/10.1145/3382025.3414969.

Boronat, A., 2017. Structural model subtyping with OCL constraints. In: Proceedings of the 10th International Conference on Software Language Engineering. SLE'17, ACM, pp. 194–205. http://dx.doi.org/10.1145/3136014.3136026.

Boronat, A., 2022. Safe reuse in modeling language engineering using model subtyping with OCL constraints. Softw. Syst. Model. 22 (3), 797–818. http://dx.doi.org/10.1007/s10270-022-01028-7.

Braga, C., Santos, C., da Silva, V.T., 2014. Consistency of model transformation contracts. Sci. Comput. Program. 92, 86–104. http://dx.doi.org/10.1016/J.SCICO.2013.08.013.

Brambilla, M., Cabot, J., Wimmer, M., 2017. Model-driven software engineering in practice, second ed. In: Synthesis Lectures on Software Engineering, Morgan & Claypool, http://dx.doi.org/10.1007/978-3-031-02549-5.

Braun, V., Clarke, V., 2021. Thematic Analysis: A Practical Guide. Sage.

Cabot, J., Clarisó, R., Guerra, E., de Lara, J., 2010. Verification and validation of declarative model-to-model transformations through invariants. J. Syst. Softw. 83 (2), 283–302. http://dx.doi.org/10.1016/j.jss.2009.08.012.

Calegari, D., Szasz, N., 2013. Verification of model transformations: A survey of the state-of-the-art. Electron. Notes Theor. Comput. Sci. 292, 5–25. http://dx.doi.org/10.1016/j.entcs.2013.02.002, Proceedings of the XXXVIII Latin American Conference in Informatics. CLEI'12.

Castellanos, C., Borde, E., Pautet, L., Sébastien, G., Vergnaud, T., 2015. Improving reusability of model transformations by automating their composition. In: Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications. SEAA'15, IEEE, pp. 267–274. http://dx.doi.org/10.1109/SEAA.2015.76.

Chen, L., Babar, M.A., 2011. A systematic review of evaluation of variability management approaches in software product lines. Inf. Softw. Technol. 53 (4), 344–362. http://dx.doi.org/10.1016/j.infsof.2010.12.006.

Chen, L., Huang, L., Li, C., Zan, T., 2013. A bidirectional framework for model synchronization in component-based software development. In: Proceedings of the 13th International Conference on Quality Software. QSIC'13, IEEE, pp. 313–319. http://dx.doi.org/10.1109/QSIC.2013.27.

Chen, X., Liu, Q., Mallet, F., Li, Q., Cai, S., Jin, Z., 2022. Formally verifying consistency of sequence diagrams for safety critical systems. Sci. Comput. Program. 216, 102777. http://dx.doi.org/10.1016/j.scico.2022.102777.

Cheng, Z., Tisi, M., 2017. Incremental deductive verification for relational model transformations. In: Proceedings of the 10th International Conference on Software Testing, Verification and Validation. ICST'17, IEEE, pp. 379–389. http://dx.doi.org/10.1109/icst.2017.41.

Ciancone, A., Filieri, A., Mirandola, R., 2010. Mantra: Towards model transformation testing. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology. QUATIC'10, IEEE, pp. 97–105. http://dx.doi.org/10.1109/QUATIC.2010.15.

Combemale, B., Crégut, X., Garoche, P.-L., Thirioux, X., Vernadat, F., 2007. A property-driven approach to formal verification of process models. In: Filipe, J., Cordeiro, J., Cardoso, J. (Eds.), Proceedings of the 9th International Conference on Enterprise Information Systems. ICEIS'07, In: LNBIP, vol. 12, Springer, pp. 286–300. http://dx.doi.org/10.1007/978-3-540-88710-2_23.

Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D., 2002. VIATRA – visual automated transformations for formal verification and validation of UML models. In: Proceedings of the 17th International Conference on Automated Software Engineering. ASE'02, IEEE, pp. 267–270. http://dx.doi.org/10.1109/ase.2002.1115027.

Cuadrado, J.S., Burgueño, L., Wimmer, M., Vallecillo, A., 2022. Efficient execution of ATL model transformations using static analysis and parallelism. IEEE Trans. Softw. Eng. 48 (4), 1097–1114. http://dx.doi.org/10.1109/TSE.2020.3011388.

Cuadrado, J.S., Guerra, E., de Lara, J., 2018. AnATLyzer: an advanced IDE for ATL model transformations. In: Companion Proceedings of the 40th International Conference on Software Engineering. ICSE'18, ACM, pp. 85–88. http://dx.doi.org/10.1145/3183440.3183479.

Cuadrado, J.S., Guerra, E., de Lara, J., Clarisó, R., Cabot, J., 2017. Translating target to source constraints in model-to-model transformations. In: Proceedings of the 20th International Conference on Model Driven Engineering Languages and Systems. MoDELS'17, IEEE, pp. 12–22. http://dx.doi.org/10.1109/MODELS.2017.12.

Czarnecki, K., Helsen, S., 2006. Feature-based survey of model transformation approaches. IBM Syst. J. 45 (3), 621–645. http://dx.doi.org/10.1147/sj.453.0621.

Daniel, M., Lawrence, N., Michael, K., 2021. Embedding quality into software product line variability artifacts. Int. J. Softw. Eng. Appl. 12 (2/3), http://dx.doi.org/10.5121/ijsea.2021.12302.

Dávid, I., Syriani, E., Verbrugge, C., Buchs, D., Blouin, D., Cicchetti, A., Vanherpen, K., 2016. Towards inconsistency tolerance by quantification of semantic inconsistencies. In: Muccini, H., Malavolta, I., Gerard, S., Kolovos, D.S. (Eds.), Proceedings of the 1st International Workshop on Collaborative Modelling in MDE. COMMitMDE'16, In: CEUR Workshop Proceedings, vol. 1717, CEUR-WS.org, pp. 35–44, URL https://ceur-ws.org/Vol-1717/paper8.pdf.

Daw, Z., Cleaveland, R., Vetter, M., 2014. Integrating model checking and UML-based model-driven development for embedded systems. Electron. Commun. EASST 66, http://dx.doi.org/10.14279/TUJ.ECEASST.66.888, Proceedings of the 13th Workshop on Automated Verification of Critical Systems. AVoCS'13.

de Lara, J., Di Rocco, J., Di Ruscio, D., Guerra, E., Iovino, L., Pierantonio, A., Cuadrado, J.S., 2017. Reusing model transformations through typing requirements models. In: Huisman, M., Rubin, J. (Eds.), Proceedings of the 20th International Conference on Fundamental Approaches To Software Engineering. FASE'17, In: LNCS, vol. 10202, Springer, pp. 264–282. http://dx.doi.org/10.1007/978-3-662-54494-5_15.

Debnath, N., Leonardi, M.C., Ridao, M., Mauco, M.V., Felice, L., Montejano, G., Riesco, D., 2011. An ATL transformation from natural language requirements models to business models of a MDA project. In: Proceedings of the 11th International Conference on ITS Telecommunications. ITST'11, IEEE, pp. 633–639. http://dx.doi.org/10.1109/ITST.2011.6060132.

Dechsupa, C., Vatanawood, W., Thongtak, A., 2019. Hierarchical verification for the BPMN design model using state space analysis. IEEE Access 7, 16795–16815. http://dx.doi.org/10.1109/ACCESS.2019.2892958.

Demuth, A., Riedl-Ehrenleitner, M., Lopez-Herrejon, R.E., Egyed, A., 2016. Co-evolution of metamodels and models through consistent change propagation. J. Syst. Softw. 111, 281–297. http://dx.doi.org/10.1016/j.jss.2015.03.003.

Di Ruscio, D., Etzlstorfer, J., Iovino, L., Pierantonio, A., Schwinger, W., 2017. A feature-based approach for variability exploration and resolution in model transformation migration. In: Anjorin, A., Espinoza, H. (Eds.), Proceedings of the 13th European Conference on Modelling Foundations and Applications. ECMFA'17, In: LNCS, vol. 10376, Springer, pp. 71–89. http://dx.doi.org/10.1007/978-3-319-61482-3_5.

Diskin, Z., Maibaum, T., Wassyng, A., Wynn-Williams, S., Lawford, M., 2018. Assurance via model transformations and their hierarchical refinement. In: Proceedings of the 21st International Conference on Model Driven Engineering Languages and Systems. MoDELS'18, ACM, pp. 426–436. http://dx.doi.org/10.1145/3239372.3239413.

Drave, I., Michael, J., Müller, E., Rumpe, B., Varga, S., 2022. Model-driven engineering of process-aware information systems. SN Comput. Sci. 3 (6), 479. http://dx.doi.org/10.1007/S42979-022-01334-3.

Drawel, N., Laarej, A., Bentahar, J., El Menshawy, M., 2022. Transformation-based model checking temporal trust in multi-agent systems. J. Syst. Softw. 192, 111383. http://dx.doi.org/10.1016/j.jss.2022.111383.

Dyck, J., Giese, H., Lambers, L., 2018. Automatic verification of behavior preservation at the transformation level for relational model transformation. Softw. Syst. Model. 18 (5), 2937–2972. http://dx.doi.org/10.1007/s10270-018-00706-9.

Ege, F., Tichy, M., 2019. A proposal of features to support analysis and debugging of declarative model transformations with graphical syntax by embedded visualizations. In: Companion Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems. MoDELS'19, IEEE, pp. 326–330. http://dx.doi.org/10.1109/MODELS-C.2019.00051.

Egyed, A., 2010. Automatically detecting and tracking inconsistencies in software design models. IEEE Trans. Softw. Eng. 37 (2), 188–204. http://dx.doi.org/10.1109/TSE.2010.38.

El-Sharkawy, S., Yamagishi-Eichler, N., Schmid, K., 2019. Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. Inf. Softw. Technol. 106, 1–30. http://dx.doi.org/10.1016/j.infsof.2018.08.015.

Erazo-Garzón, L., Suquisupa, S., Bermeo, A., Cedillo, P., 2022. Model-driven engineering applied to user interfaces. a systematic literature review. In: Botto-Tobar, M., Vizuete, M.Z., León, S.M., Torres-Carrión, P., Durakovic, B. (Eds.), Revised Selected Papers of the 4th International Conference on Applied Technologies. ICAT'22, In: CCIS, vol. 1755, Springer, pp. 575–591. http://dx.doi.org/10.1007/978-3-031-24985-3_42.

Esbai, R., Erramdani, M., 2015. Model-to-model transformation in approach by modeling: From UML model to model-view-presenter and dependency injection patterns. In: Proceedings of the 5th World Congress on Information and Communication Technologies. WICT'15, IEEE, pp. 1–6. http://dx.doi.org/10.1109/WICT.2015.7489648.

Estdale, J., Georgiadou, E., 2018. Applying the ISO/IEC 25010 quality models to software product. In: Larrucea, X., Santamaría, I., O'Connor, R.V., Messnarz, R. (Eds.), Proceedings of the 25th European Conference on Systems, Software and Services Process Improvement. EuroSPI'18, In: CCIS, vol. 896, Springer, pp. 492–503. http://dx.doi.org/10.1007/978-3-319-97925-0_42.

Fatwanto, A., Boughton, C., 2008. Analysis, specification and modeling of non-functional requirements for translative model-driven development. In: Proceedings of the 4th International Conference on Computational Intelligence and Security. CIS'08, 2, IEEE, pp. 405–410. http://dx.doi.org/10.1109/CIS.2008.215.

Faunes, M., Sahraoui, H., Boukadoum, M., 2012. Generating model transformation rules from examples using an evolutionary algorithm. In: Proceedings of the 27th International Conference on Automated Software Engineering. ASE'12, ACM, pp. 250–253. http://dx.doi.org/10.1145/2351676.2351714.

Feichtinger, K., Rabiser, R., 2020. Variability model transformations: Towards unifying variability modeling. In: Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA'20, IEEE, pp. 179–182. http://dx.doi.org/10.1109/SEAA51224.2020.00037.

Feichtinger, K., Stöbich, J., Romano, D., Rabiser, R., 2021. TRAVART: An approach for transforming variability models. In: Proceedings of the 15th International Working Conference on Variability Modelling of Software-Intensive Systems. VaMoS'21, ACM, pp. 1–10. http://dx.doi.org/10.1145/3442391.3442400.

Fernandez Adiego, B., Darvas, D., Vinuela, E.B., Tournier, J.-C., Bliudze, S., Blech, J.O., Gonzalez Suarez, V.M., 2015. Applying model checking to industrial-sized PLC programs. IEEE Trans. Ind. Inform. (6), 1400–1410. http://dx.doi.org/10.1109/tii.2015.2489184.

Flake, S., Mueller, W., 2003. Formal semantics of static and temporal state-oriented OCL constraints. Softw. Syst. Model. 2 (3), 164–186. http://dx.doi.org/10.1007/S10270-003-0026-X.

Fleurey, F., Steel, J., Baudry, B., 2004. Validation in model-driven engineering: Testing model transformations. In: Proceedings of the 1st International Workshop on Model, Design and Validation. MoDeVa'04, IEEE, pp. 29–40. http://dx.doi.org/10.1109/MODEVA.2004.1425846.

Fowler, M., 2010. Domain-Specific Languages. Addison-Wesley.

Fradet, P., Girault, A., Krishnaswamy, R., Nicollin, X., Shafiei, A., 2023. RDF: A reconfigurable dataflow model of computation. ACM Trans. Embed. Comput. Syst. 22 (1), 12:1–12:30. http://dx.doi.org/10.1145/3544972.

Fritsche, L., Kosiol, J., Schürr, A., Taentzer, G., 2021. Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars. Int. J. Softw. Tools Technol. Transf. 23 (3), 335–368. http://dx.doi.org/10.1007/S10009-020-00588-7.

Gadducci, F., Laretto, A., Trotta, D., 2023. Specification and verification of a linear-time temporal logic for graph transformation. In: Fernández, M., Poskitt, C.M. (Eds.), Proceedings of the 16th International Conference on Graph Transformation. ICGT'23, In: LNCS, vol. 13961, Springer, pp. 22–42. http://dx.doi.org/10.1007/978-3-031-36709-0_2.

Gadelha Queiroz, P.G., Vaccare Braga, R.T., 2014. Development of critical embedded systems using model-driven and product lines techniques: A systematic review. In: Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse. SBCARS'14, IEEE, pp. 74–83. http://dx.doi.org/10.1109/sbcars.2014.19.

Geismann, J., Bodden, E., 2020. A systematic literature review of model-driven security engineering for cyber-physical systems. J. Syst. Softw. 169, 110697. http://dx.doi.org/10.1016/J.JSS.2020.110697.

Gerpheide, C.M., Schiffelers, R.R., Serebrenik, A., 2016. Assessing and improving quality of QVTo model transformations. Softw. Qual. J. 24 (3), 797–834. http://dx.doi.org/10.1007/S11219-015-9280-8.

Ghaedi Heidari, S., Ajoudanian, S., 2023. Automatic pattern-based consistency checking in model refactoring: introducing a formal behavioral preserving method. Innov. Syst. Softw. Eng. 20 (1), 65–84. http://dx.doi.org/10.1007/s11334-022-00525-8.

Giese, H., Wagner, R., 2008. From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. 8 (1), 21–43. http://dx.doi.org/10.1007/s10270-008-0089-9.

Giraldo Velásquez, F.D., 2017. A framework for evaluating the quality of modeling languages in MDE environments (Ph.D. thesis). Universitat Politècnica de València, http://dx.doi.org/10.4995/thesis/10251/90628.

Gogolla, M., Vallecillo, A., 2011. *Tract*able model transformation testing. In: France, R.B., Küster, J.M., Bordbar, B., Paige, R.F. (Eds.), Proceedings of the 7th European Conference on Modelling Foundations and Applications. ECMFA'11, In: LNCS, 6698, Springer, pp. 221–235. http://dx.doi.org/10.1007/978-3-642-21470-7_16.

Gray, T., Bork, D., de Vries, M., 2020. A new DEMO modelling tool that facilitates model transformations. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J. (Eds.), Proceedings of the 21st International Conference on Enterprise, Business-Process and Information Systems Modeling and the 25th International Working Conference on Exploring Modeling Methods for Systems Analysis and Development. BPMDS/EMMSAD'20, In: LNBIP, vol. 387, Springer, pp. 359–374. http://dx.doi.org/10.1007/978-3-030-49418-6_25.

Greiner, S., Schwägerl, F., Westfechtel, B., 2017. Realizing multi-variant model transformations on top of reused ATL specifications. In: Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development. MODELSWARD'17, SciTePress, pp. 362–373. http://dx.doi.org/10.5220/0006137803620373.

Guerra, E., Soeken, M., 2013. Specification-driven model transformation testing. Softw. Syst. Model. 14 (2), 623–644. http://dx.doi.org/10.1007/s10270-013-0369-x.

Guo, J., Wang, Y., Trinidad, P., Benavides, D., 2012. Consistency maintenance for evolving feature models. Expert Syst. Appl. 39 (5), 4987–4998. http://dx.doi.org/10.1016/J.ESWA.2011.10.014.

Gwasem, I., Du, W., McAllister, A., 2023. Acquiring testable NFRs utilizing goal models enhancing application requirements analysis in goal-driven software product lines. In: Proceedings of the 8th International Conference on Engineering Technologies and Applied Sciences. ICETAS'23, IEEE, pp. 1–8. http://dx.doi.org/10.1109/ICETAS59148.2023.10346316.

Hadad, A.S.A., Ma, C., Ahmed, A.A.O., 2020. Formal verification of AADL models by Event-B. IEEE Access 8, 72814–72834. http://dx.doi.org/10.1109/access.2020.2987972.

Han, D., Xing, J., Yang, Q., Wang, H., Zhang, X., 2016. Formal sequence: Extending UML sequence diagram for behavior description and formal verification. In: Workshops Proceedings of the 40th Computer Software and Applications Conference. COMPSAC'16, 2, IEEE, pp. 474–481. http://dx.doi.org/10.1109/COMPSAC.2016.118.

He, X., Zhang, T., Hu, C.-J., Ma, Z., Shao, W., 2016. An MDE performance testing framework based on random model generation. J. Syst. Softw. 121, 247–264. http://dx.doi.org/10.1016/j.jss.2016.04.044.

Hidaka, S., Tisi, M., 2024. Partial bidirectionalization of model transformation languages. In: Proceedings of the 27th International Conference on Model Driven Engineering Languages and Systems. MoDELS'24, ACM, pp. 1–12. http://dx.doi.org/10.1145/3640310.367408.

Hilken, F., Gogolla, M., Burgueño, L., Vallecillo, A., 2018. Testing models and model transformations using classifying terms. Softw. Syst. Model. 17 (3), 885–912. http://dx.doi.org/10.1007/S10270-016-0568-3.

Hinkel, G., Goldschmidt, T., Burger, E., Reussner, R., 2019. Using internal domain-specific languages to inherit tool support and modularity for model transformations. Softw. Syst. Model. 18 (1), 129–155. http://dx.doi.org/10.1007/S10270-017-0578-9.

Hölldobler, K., Rumpe, B., Weisemöller, I., 2015. Systematically deriving domain-specific transformation languages. In: Proceedings of the 18th International Conference on Model Driven Engineering Languages and Systems. MoDELS'15, IEEE, pp. 136–145. http://dx.doi.org/10.1109/MODELS.2015.7338244.

Höppner, S., Tichy, M., 2024. Traceability and reuse mechanisms, the most important properties of model transformation languages. Empir. Softw. Eng. 29 (2), http://dx.doi.org/10.1007/s10664-023-10428-2.

Hu, K., Zhang, T., Yang, Z., Tsai, W.-T., 2015. Exploring AADL verification tool through model transformation. J. Syst. Archit. 61 (3–4), 141–156. http://dx.doi.org/10.1016/j.sysarc.2015.02.003.

Hudson, M., Sprinkle, J., 2011. Simplification of semantically-rich model transformations through generated transformation blocks. In: Proceedings of the 18th International Conference and Workshops on Engineering of Computer-Based Systems. ECBS'11, IEEE, pp. 260–268. http://dx.doi.org/10.1109/ECBS.2011.28.

Hülsbusch, M., König, B., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H., 2010. Showing full semantics preservation in model transformation: A comparison of techniques. In: Méry, D., Merz, S. (Eds.), Proceedings of the 8th International Conference on Integrated Formal Methods. IFM'10, In: LNCS, vol. 6396, Springer, pp. 183–198. http://dx.doi.org/10.1007/978-3-642-16265-7_14.

Hutchesson, S., McDermid, J., 2011. Towards cost-effective high-assurance software product lines: The need for property-preserving transformations. In: Proceedings of the 15th International Software Product Line Conference. SPLC'11, IEEE, pp. 55–64. http://dx.doi.org/10.1109/SPLC.2011.32.

Iqbal, S., Al-Azzoni, I., 2022. Test case prioritization for model transformations. J. King Saud Univ. - Comput. Inf. Sci. 34 (8), 6324–6338. http://dx.doi.org/10.1016/J.JKSUCI.2021.08.011.

Jadhav, P.P., Pande, S.D., Joshi, R., Gonge, S., Shelke, M.P., Futane, P., Ahammad, S.K.H., 2024. CSADF: ingesting cuckoo search optimization algorithm enabled with fitness function for effective model transformation pertaining to ADF. Multimedia Tools Appl. 83, 79219–79234. http://dx.doi.org/10.1007/s11042-023-17887-1.

Jadoon, G., 2024. Preserving non-functional requirements in goal models using metamodels of the software product lines. In: Proceedings of the 28th International Systems and Software Product Line Conference. SPLC'24, 2, pp. 1–5. http://dx.doi.org/10.1145/3646548.3676541.

Jadoon, G., 2024–2025. Dataset for paper: Model transformation and property preservation in rigorous software development: a systematic literature review. http://dx.doi.org/10.5281/zenodo.15227409, Zenodo.

Jadoon, G., Ahmed, A., Din, I.U., Almogren, A., Zareei, M., Roshan-Biswal, R., Altameem, A., 2024. Dynamic property preservation in AIoT: A machine learning approach for data-efficient model transformation. IEEE Access 12, 130707–130722. http://dx.doi.org/10.1109/ACCESS.2024.3454717.

Jadoon, G., Shafi, M., Jan, S., 2019. A model-oriented requirements traceability framework for small and medium software industries. In: Proceedings of the 20th International Arab Conference on Information Technology. ACIT'19, IEEE, pp. 91–96. http://dx.doi.org/10.1109/ACIT47987.2019.8991116.

Jadoon, G., Shafi, M., Jan, S., 2020. Model transformations carried by the traceability framework for enterprises in software industry. Int. Arab. J. Inf. Technol. 17 (4A), 579–587. http://dx.doi.org/10.34028/IAJIT/17/4A/1.

Jilani, A.A., Iqbal, M.Z., Khan, M.U., 2014. A search based test data generation approach for model transformations. In: Di Ruscio, D., Varró, D. (Eds.), Proceedings of the 7th International Conference on Theory and Practice of Model Transformations. ICMT'14, In: LNCS, vol. 8568, Springer, pp. 17–24. http://dx.doi.org/10.1007/978-3-319-08789-4_2.

Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D., 2019. Survey and classification of model transformation tools. Softw. Syst. Model. 18 (4), 2361–2397. http://dx.doi.org/10.1007/S10270-018-0665-6.

Kang, E.-Y., Enoiu, E.P., Marinescu, R., Seceleanu, C., Schobbens, P.-Y., Pettersson, P., 2013. A methodology for formal analysis and verification of EAST-ADL models. Reliab. Eng. Syst. Saf. 120, 127–138. http://dx.doi.org/10.1016/j.ress.2013.06.007.

Karimi, M., Kolahdouz Rahimi, S., Troya, J., 2024. Ant-colony optimization for automating test models generation in model transformation testing. J. Syst. Softw. 208, 111882. http://dx.doi.org/10.1016/J.JSS.2023.111882.

Kessentini, M., Sahraoui, H., Boukadoum, M., 2011. Example-based model-transformation testing. Autom. Softw. Eng. 18 (2), 199–224. http://dx.doi.org/10.1007/s10515-010-0079-3.

Kitchenham, B.A., Charters, S.M., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE-2007-01, Keele University and University of Durham.

Klare, H., Gleitze, J., 2019. Commonalities for preserving consistency of multiple models. In: Companion Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems Companion. MoDELS'19, IEEE, pp. 371–378. http://dx.doi.org/10.1109/MODELS-C.2019.00058.

Klare, H., Kramer, M.E., Langhammer, M., Werle, D., Burger, E., Reussner, R., 2021. Enabling consistency in view-based system development – The VITRUVIUS approach. J. Syst. Softw. 171, 110815. http://dx.doi.org/10.1016/J.JSS.2020.110815.

Ko, J.-W., Chung, K.-Y., Han, J.-S., 2013. Model transformation verification using similarity and graph comparison algorithm. Multimedia Tools Appl. 74 (20), 8907–8920. http://dx.doi.org/10.1007/s11042-013-1581-y.

Kolahdouz-Rahimi, S., Lano, K., 2011. A model-based development approach for model transformations. In: Arbab, F., Sirjani, M. (Eds.), Revised Selected Papers of the 4th IPM International Conference on Fundamentals of Software Engineering. FSEN'11, In: LNCS, vol. 7141, Springer, pp. 48–63. http://dx.doi.org/10.1007/978-3-642-29320-7_4.

Krystosik, A., Turlej, D., 2006. EMLAN: A language for model checking of embedded systems software. IFAC Proc. Vol. 39 (21), 126–131. http://dx.doi.org/10.1016/s1474-6670(17)30171-4, Proceedings of the 6th IFAC Workshop on Programmable Devices and Embedded Systems. PDeS'06.

Krzywicki, K., Andrzejewski, G., Zajac, W., Krolikowski, T., Barkalov, A., Titarenko, L., 2021. IPNES – interpreted Petri net for embedded systems. Procedia Comput. Sci. 192, 2012–2021. http://dx.doi.org/10.1016/j.procs.2021.08.207, Proceedings of the 25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems. KES'21.

Lano, K., Kolahdouz-Rahimi, S., Fang, S., 2021. Model transformation development using automated requirements analysis, metamodel matching, and transformation by example. ACM Trans. Softw. Eng. Methodol. 31 (2), 18:1–18:71. http://dx.doi.org/10.1145/3471907.

Lano, K., Kolahdouz-Rahimi, S., Yassipour-Tehrani, S., 2019. Declarative specification of bidirectional transformations using design patterns. IEEE Access 7, 5222–5249. http://dx.doi.org/10.1109/access.2018.2889399.

Lano, K., Kolahdouz-Rahimi, S., Yassipour-Tehrani, S., Sharbaf, M., 2018. A survey of model transformation design patterns in practice. J. Syst. Softw. 140, 48–73. http://dx.doi.org/10.1016/j.jss.2018.03.001.

Le, D.M., Lee, H., Kang, K.C., Keun, L., 2013. Validating consistency between a feature model and its implementation. In: Favaro, J.M., Morisio, M. (Eds.), Proceedings of the 13th International Conference on Software Reuse. ICSR'13, In: LNCS, vol. 7925, Springer, pp. 1–16. http://dx.doi.org/10.1007/978-3-642-38977-1_1.

Le Calvar, T., Jouault, F., Chhel, F., Saubion, F., Clavreul, M., 2019. Intensional view definition with constrained incremental transformation rules. In: Companion Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems. MoDELS'19, IEEE, pp. 395–402. http://dx.doi.org/10.1109/MODELS-C.2019.00061.

Leblebici, E., Anjorin, A., Schürr, A., 2017. Inter-model consistency checking using triple graph grammars and linear optimization techniques. In: Huisman, M., Rubin, J. (Eds.), Proceedings of the 20th International Conference on Fundamental Approaches To Software Engineering. FASE'17, In: LNCS, vol. 10202, Springer, pp. 191–207. http://dx.doi.org/10.1007/978-3-662-54494-5_11.

Leuschel, M., Falampin, J., Fritz, F., Plagge, D., 2011. Automated property verification for large scale B models with ProB. Form. Asp. Comput. 23 (6), 683–709. http://dx.doi.org/10.1007/S00165-010-0172-1.

Li, P., Jiang, M., Ding, Z., 2020. Fault localization with weighted test model in model transformations. IEEE Access 8, 14054–14064. http://dx.doi.org/10.1109/access.2020.2966540.

Liebrenz, T., Herber, P., Glesner, S., 2018. Deductive verification of hybrid control systems modeled in Simulink with KeYmaera X. In: Sun, J., Sun, M. (Eds.), Proceedings of the 20th International Conference on Formal Engineering Methods. ICFEM'18, In: LNCS, vol. 11232, Springer, pp. 89–105. http://dx.doi.org/10.1007/978-3-030-02450-5_6.

Liu, J., He, K., Li, B., He, C., Liang, P., 2005. A transformation definition metamodel for model transformation. In: Proceedings of the International Conference on Information Technology: Coding and Computing. ITCC'05, 2, IEEE, pp. 373–378. http://dx.doi.org/10.1109/ITCC.2005.49.

Liu, Z., Jiang, F., Qian, S., 2010. Semantic consistency checking for model transformations. In: Proceedings of the 2nd International Conference on Computer Engineering and Technology. ICCET'10, 6, IEEE, pp. V6:364–V6:368. http://dx.doi.org/10.1109/ICCET.2010.5486198.

Lopez-Herrejon, R.E., Egyed, A., 2012. Towards fixing inconsistencies in models with variability. In: Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems. VaMoS'12, ACM, pp. 93–100. http://dx.doi.org/10.1145/2110147.2110158.

Lopez-Herrejon, R.E., Egyed, A., Trujillo, S., De Sosa, J., Azanza, M., 2010. Using incremental consistency management for conformance checking in feature-oriented model-driven engineering. In: Benavides, D., Batory, D.S., Grünbacher, P. (Eds.), Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS'10, In: ICB-Research Report, vol. 37, Universität Duisburg-Essen, pp. 93–100.

Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G.M., Syriani, E., Wimmer, M., 2016. Model transformation intents and their properties. Softw. Syst. Model. 15 (3), 647–684. http://dx.doi.org/10.1007/S10270-014-0429-X.

Lugou, F., Li, L.W., Apvrille, L., Ameur-Boulifa, R., 2016. SysML models and model transformation for security. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. MODELSWARD'16, SciTePress, pp. 331–338. http://dx.doi.org/10.5220/0005748703310338.

Lukács, G., Bartha, T., 2022. Formal modeling and verification of the functionality of electronic urban railway control systems through a case study. Urban Rail Transit 8 (3–4), 217–245. http://dx.doi.org/10.1007/s40864-022-00177-8.

Macías, F., Wolter, U., Rutle, A., Durán, F., Rodriguez-Echeverria, R., 2019. Multilevel coupled model transformations for precise and reusable definition of model behaviour. J. Log. Algebr. Methods Program. 106, 167–195. http://dx.doi.org/10.1016/J.JLAMP.2018.12.005.

Madlener, F., Weingart, J., Huss, S.A., 2010. Verification of dynamically reconfigurable embedded systems by model transformation rules. In: Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis. CODES/ISSS'10, ACM, pp. 33–40. http://dx.doi.org/10.1145/1878961.1878969.

Magalhães, A.P.F., Andrade, A.M.S., Maciel, R.S.P., 2016. A model driven transformation development process for model to model transformation. In: Proceedings of the 30th Brazilian Symposium on Software Engineering. SBES'16, ACM, pp. 3–12. http://dx.doi.org/10.1145/2973839.2973841.

Magalhaes, A.P.F., Andrade, A.M.S., Maciel, R.S.P., 2019. Model driven transformation development (MDTD): An approach for developing model to model transformation. Inf. Softw. Technol. 114, 55–76. http://dx.doi.org/10.1016/J.INFSOF.2019.06.004.

Magalhães, A.P., Maciel, R.S.P., Andrade, A.M.S., 2020a. An investigation of currently used aspects in model transformation development. In: Filipe, J., Smialek, M., Brodsky, A., Hammoudi, S. (Eds.), Revised Selected Papers of the 22nd International Conference on Enterprise Information Systems. ICEIS'20, In: LNBIP, vol. 417, Springer, pp. 412–436. http://dx.doi.org/10.1007/978-3-030-75418-1_19.

Magalhães, A.P., Maciel, R.S.P., Andrade, A., 2020b. Developing model transformations: A systematic literature review. In: Proceedings of the 22nd International Conference on Enterprise Information Systems. ICEIS'20, 2, SciTePress, pp. 80–89. http://dx.doi.org/10.5220/0009380100800089.

Meghzili, S., Chaoui, A., Strecker, M., Kerkouche, E., 2016. Transformation and validation of BPMN models to Petri nets models using GROOVE. In: Proceedings of the 2nd International Conference on Advanced Aspects of Software Engineering. ICAASE'16, IEEE, pp. 22–29. http://dx.doi.org/10.1109/icaase.2016.7843859.

Meghzili, S., Chaoui, A., Strecker, M., Kerkouche, E., 2017. On the verification of UML state machine diagrams to colored Petri nets transformation using isabelle/HOL. In: Proceedings of the 17th IEEE International Conference on Information Reuse and Integration. IRI'17, IEEE, pp. 419–426. http://dx.doi.org/10.1109/iri.2017.63.

Meng, Y., Huang, Z., Shen, G., Ke, C., 2021. A security policy model transformation and verification approach for software defined networking. Comput. Secur. 100, 102089. http://dx.doi.org/10.1016/j.cose.2020.102089.

Mens, T., Van Gorp, P., 2006. A taxonomy of model transformation. Electron. Notes Theor. Comput. Sci. 152, 125–142. http://dx.doi.org/10.1016/j.entcs.2005.10.021, Proceedings of the International Workshop on Graph and Model Transformation. GraMoT'05.

Meyma, M.M., Laaz, N., Mbarki, S., 2023. A new model-based approach for migrating Health 2.0 to Health 3.0 applications. In: Kacprzyk, J., Ezziyyani, M., Balas, V.E. (Eds.), Proceedings of the International Conference on Advanced Intelligent Systems for Sustainable Development. AI2SD'23, In: LNNS, Springer, pp. 673–682. http://dx.doi.org/10.1007/978-3-031-35248-5_59.

Mkaouar, H., Blouin, D., Borde, E., 2022. A benchmark of incremental model transformation tools based on an industrial case study with AADL. Softw. Syst. Model. 22 (1), 175–201. http://dx.doi.org/10.1007/s10270-022-00989-z.

Moffett, Y., Dingel, J., Beaulieu, A., 2013. Verifying protocol conformance using software model checking for the model-driven development of embedded systems. IEEE Trans. Softw. Eng. 39 (9), 1307–13256. http://dx.doi.org/10.1109/TSE.2013.14.

Mohagheghi, P., Dehlen, V., Neple, T., 2009. Definitions and approaches to model quality in model-based software development – A review of literature. Inf. Softw. Technol. 51 (12), 1646–1669. http://dx.doi.org/10.1016/j.infsof.2009.04.004.

Moon, M., Yeom, K., Chae, H.S., 2005. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. IEEE Trans. Softw. Eng. 31 (7), 551–569. http://dx.doi.org/10.1109/TSE.2005.76.

Mottu, J.-M., Sen, S., Tisi, M., Cabot, J., 2012. Static analysis of model transformations for effective test generation. In: Proceedings of the 23rd International Symposium on Software Reliability Engineering. ISSRE'12, IEEE, pp. 291–300. http://dx.doi.org/10.1109/issre.2012.7.

Mouakher, I., Dhaou, F., Attiogbé, J.C., 2022. Event-based semantics of UML 2.X concurrent sequence diagrams for formal verification. J. Comput. Sci. Tech. 37 (1), 4–28. http://dx.doi.org/10.1007/s11390-021-1673-5.

Muliawan, O., 2008. Extending a model transformation language using higher order transformations. In: Proceedings of the 15th Working Conference on Reverse Engineering. WCRE'08, IEEE, pp. 315–318. http://dx.doi.org/10.1109/WCRE.2008.19.

Munoz, D.-J., Gurov, D., Pinto, M., Fuentes, L., 2021. Category theory framework for variability models with non-functional requirements. In: Rosa, M.L., Sadiq, S.W., Teniente, E. (Eds.), Proceedings of the 33rd International Conference on Advanced Information Systems Engineering. CAiSE'21, In: LNCS, vol. 12751, Springer, pp. 397–413. http://dx.doi.org/10.1007/978-3-030-79382-1_24.

Narayanan, A., Karsai, G., 2008. Towards verifying model transformations. Electron. Notes Theor. Comput. Sci. 211, 191–200. http://dx.doi.org/10.1016/j.entcs.2008.04.041, Proceedings of the 5th International Workshop on Graph Transformation and Visual Modeling Techniques. GT-VMT'06.

Neis, P., Wehrmeister, M.A., Mendes, M.F., 2019. Model driven software engineering of power systems applications: Literature review and trends. IEEE Access 7, 177761–177773. http://dx.doi.org/10.1109/ACCESS.2019.2958275.

Nguyen, T.-H., Dang, D.-H., 2018. An approach for testing model transformations. In: Proceedings of the 10th International Conference on Knowledge and Systems Engineering. KSE'18, IEEE, pp. 264–269. http://dx.doi.org/10.1109/kse.2018.8573367.

Nguyen, T.-H., Dang, D.-H., 2021. A graph analysis based approach for specification-driven testing of model transformations. In: Proceedings of the 8th NAFOSTED Conference on Information and Computer Science. NICS'21, IEEE, pp. 224–229. http://dx.doi.org/10.1109/nics54270.2021.9701514.

Nguyen, P.H., Kramer, M.E., Klein, J., Le Traon, Y., 2015. An extensive systematic review on the model-driven development of secure systems. Inf. Softw. Technol. 68, 62–81. http://dx.doi.org/10.1016/J.INFSOF.2015.08.006.

Oakes, B.J., Troya, J., Lúcio, L., Wimmer, M., 2015. Fully verifying transformation contracts for declarative ATL. In: Proceedings of the 18th International Conference on Model Driven Engineering Languages and Systems. MoDELS'15, IEEE, pp. 256–265. http://dx.doi.org/10.1109/MODELS.2015.7338256.

Olajubu, O., Ajit, S., Johnson, M., Thomson, S., Edwards, M., Turner, S., 2017. Automated test case generation from high-level logic requirements using model transformation techniques. In: Proceedings of the 9th Computer Science and Electronic Engineering Conference. CEEC'17, IEEE, pp. 178–182. http://dx.doi.org/10.1109/ceec.2017.8101621.

Page, M.J., McKenzie, J.E., Bossuyt, P.M., Boutron, I., Hoffmann, T.C., Mulrow, C.D., Shamseer, L., Tetzlaff, J.M., Akl, E.A., Brennan, S.E., et al., 2021. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ 372, n71:1–n71:9. http://dx.doi.org/10.1136/bmj.n71.

Paige, R.F., Brooke, P.J., Ostroff, J.S., 2007. Metamodel-based model conformance and multiview consistency checking. ACM Trans. Softw. Eng. Methodol. 16 (3), 11:1–11:49. http://dx.doi.org/10.1145/1243987.1243989.

Panach, J.I., España, S., Dieste, Ó., Pastor, Ó., Juristo, N., 2015. In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. Inf. Softw. Technol. 62, 164–186. http://dx.doi.org/10.1016/j.infsof.2015.02.012.

Pietron, J., Jutz, B., Raschke, A., Tichy, M., 2024. Editql: A textual query language for evolving models. In: Proceedings of the 27th International Conference on Model Driven Engineering Languages and Systems. MoDELS'24, ACM, pp. 37–48. http://dx.doi.org/10.1145/3640310.3674101.

Prehofer, C., 2016. Property preservation for extension patterns of state transition diagrams. In: Ábrahám, E., Huisman, M. (Eds.), Proceedings of the 12th International Conference on Integrated Formal Methods. IFM'16, In: LNCS, vol. 9681, Springer, pp. 260–274. http://dx.doi.org/10.1007/978-3-319-33693-0_17.

Qiuyan, L., Jie, T., Qiuhong, P., Ji, W., Chao, L., 2011. Automatic transformation technology from AADL model to UML model. In: Proceedings of the 3rd International Conference on Communication Software and Networks. ICCSN'11, IEEE, pp. 255–258. http://dx.doi.org/10.1109/ICCSN.2011.6014264.

Rahim, M., Hammad, A., Boukala-Ioualalen, M., 2015. Towards the formal verification of SysML specifications: Translation of activity diagrams into modular Petri nets. In: Proceedings of the 3rd International Conference on Applied Computing and Information Technology and the 2nd International Conference on Computational Science and Intelligence. ACIT/CSI'15, IEEE, pp. 509–516. http://dx.doi.org/10.1109/ACIT-CSI.2015.97.

Rashid, M., Anwar, M.W., Khan, A.M., 2015. Toward the tools selection in model based system engineering for embedded systems—A systematic literature review. J. Syst. Softw. 106, 150–163. http://dx.doi.org/10.1016/J.JSS.2015.04.089.

Reder, A., Egyed, A., 2012. Incremental consistency checking for complex design rules and larger model changes. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (Eds.), Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems. MoDELS'12, In: LNCS, vol. 7590, Springer, pp. 202–218. http://dx.doi.org/10.1007/978-3-642-33666-9_14.

Rodriguez-Echeverria, R., Macías, F., Rutle, A., Conejero, J.M., 2021. Suggesting model transformation repairs for rule-based languages using a contract-based testing approach. Softw. Syst. Model. 21 (1), 81–112. http://dx.doi.org/10.1007/s10270-021-00891-0.

Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A., 2009. Enhanced automation for managing model and metamodel inconsistency. In: Proceedings of the 24th International Conference on Automated Software Engineering. ASE'09, IEEE, pp. 545–549. http://dx.doi.org/10.1109/ASE.2009.57.

Rutle, A., Iovino, L., König, H., Diskin, Z., 2018. Automatic transformation co-evolution using traceability models and graph transformation. In: Pierantonio, A., Trujillo, S. (Eds.), Proceedings of the 14th European Conference on Modelling Foundations and Applications. ECMFA'18, In: LNCS, vol. 10890, Springer, pp. 80–96. http://dx.doi.org/10.1007/978-3-319-92997-2_6.

Sahin, D., Kessentini, M., Wimmer, M., Deb, K., 2015. Model transformation testing: a bi-level search-based software engineering approach. J. Softw.: Evol. Process. 27 (11), 821–837. http://dx.doi.org/10.1002/smr.1735.

Samimi-Dehkordi, L., Zamani, B., Kolahdouz-Rahimi, S., 2018. EVL+Strace: a novel bidirectional model transformation approach. Inf. Softw. Technol. 100, 47–72. http://dx.doi.org/10.1016/j.infsof.2018.03.011.

Santos, A.R., de Oliveira, R.P., de Almeida, E.S., 2015. Strategies for consistency checking on software product lines: a mapping study. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. EASE'15, ACM, pp. 5:1–5:14. http://dx.doi.org/10.1145/2745802.2745806.

Schneider, S., Maximova, M., Giese, H., 2024. Bounded model checking for interval probabilistic timed graph transformation systems against properties of probabilistic metric temporal graph logic. J. Log. Algebr. Methods Program. 137, 100938. http://dx.doi.org/10.1016/j.jlamp.2023.100938.

Schonbock, J., Kappel, G., Wimmer, M., Kusel, A., Retschitzegger, W., Schwinger, W., 2013. TETRA$_{Box}$ - a generic white-box testing framework for model transformations. In: Proceedings of the 20th Asia-Pacific Software Engineering Conference. APSEC'13, IEEE, pp. 75–82. http://dx.doi.org/10.1109/apsec.2013.21.

Selim, G.M., Cordy, J.R., Dingel, J., 2012. Model transformation testing: The state of the art. In: Proceedings of the 1st Workshop on the Analysis of Model Transformations. AMT'12, ACM, pp. 21–26. http://dx.doi.org/10.1145/2432497.2432502.

Shamsujjoha, M., Grundy, J., Li, L., Khalajzadeh, H., Lu, Q., 2021. Developing mobile applications via model driven development: A systematic literature review. Inf. Softw. Technol. 140, 106693. http://dx.doi.org/10.1016/J.INFSOF.2021.106693.

Sick, B., Hathorn, T., Durr, O., 2021. Deep transformation models: Tackling complex regression problems with neural network based transformation models. In: Proceedings of the 25th International Conference on Pattern Recognition. ICPR'20, IEEE, pp. 2476–2481. http://dx.doi.org/10.1109/icpr48806.2021.9413177.

Sommerville, I., 2015. Software Engineering, tenth ed. Pearson.

Somogyi, N., Mezei, G., 2023. Verifying static constraints on models using general formal verification methods. In: Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering. MODELSWARD'23, SciTePress, pp. 85–93. http://dx.doi.org/10.5220/0011796500003402.

Spanoudakis, G., Zisman, A., 2001. Inconsistency management in software engineering: Survey and open research issues. In: Chang, S.-K. (Ed.), Handbook of Software Engineering & Knowledge Engineering: Vol. I Fundamentals. World Scientific, pp. 329–380. http://dx.doi.org/10.1142/9789812389718_0015.

Stevens, P., 2010. Bidirectional model transformations in QVT: Semantic issues and open questions. Softw. Syst. Model. 9 (1), 7–20. http://dx.doi.org/10.1007/S10270-008-0109-9.

Stevens, P., 2020. Connecting software build with maintaining consistency between models: towards sound, optimal, and flexible building from megamodels. Softw. Syst. Model. 19 (4), 935–958. http://dx.doi.org/10.1007/s10270-020-00788-4.

Stünkel, P., von Bargen, O., Rutle, A., Lamo, Y., 2020. GraphQL federation: A model-based approach. J. Object Technol. 19 (2), 18:1–18:21. http://dx.doi.org/10.5381/jot.2020.19.2.a18.

Stünkel, P., König, H., Lamo, Y., Rutle, A., 2021. Comprehensive systems: a formal foundation for multi-model consistency management. Form. Asp. Comput. 33 (6), 1067–1114. http://dx.doi.org/10.1007/s00165-021-00555-2.

Syriani, E., Gray, J., 2012. Challenges for addressing quality factors in model transformation. In: Proceedings of the 5th International Conference on Software Testing, Verification and Validation. ICST'12, IEEE, pp. 929–937. http://dx.doi.org/10.1109/icst.2012.198.

Szvetits, M., Zdun, U., 2016. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. Softw. Syst. Model. 15 (1), 31–69. http://dx.doi.org/10.1007/S10270-013-0394-9.

Tariq, S., Cheema, S.M., 2021. Approaches for non-functional requirement modeling: A literature survey. In: Proceedings of the 4th International Conference on Computing & Information Sciences. ICCIS'21, IEEE, pp. 1–6. http://dx.doi.org/10.1109/ICCIS54243.2021.9676398.

Terayawan, S., Vatanawood, W., 2019. Transforming control-flow patterns of YAWL to Petri nets. In: Proceedings of the 1st International Communication Engineering and Cloud Computing Conference. CECCC'19, ACM, pp. 23–28. http://dx.doi.org/10.1145/3380678.3380681.

Troya, J., Segura, S., Burgueño, L., Wimmer, M., 2022. Model transformation testing and debugging: A survey. ACM Comput. Surv. 55 (4), 72:1–72:39. http://dx.doi.org/10.1145/3523056.

Troya, J., Segura, S., Parejo, J.A., Ruiz-Cortés, A., 2018. Spectrum-based fault localization in model transformations. ACM Trans. Softw. Eng. Methodol. 27 (3), 13:1–13:50. http://dx.doi.org/10.1145/3241744.

Tsigkanos, C., Li, N., Jin, Z., Hu, Z., Ghezzi, C., 2020. Scalable multiple-view analysis of reactive systems via bidirectional model transformations. In: Proceedings of the 35th International Conference on Automated Software Engineering. ASE'20, ACM, pp. 993–1003. http://dx.doi.org/10.1145/3324884.3416579.

Uzun, B., Tekinerdogan, B., 2018. Model-driven architecture based testing: A systematic literature review. Inf. Softw. Technol. 102, 30–48. http://dx.doi.org/10.1016/j.infsof.2018.05.004.

Vierhauser, M., Dhungana, D., Heider, W., Rabiser, R., Egyed, A., 2010a. Tool support for incremental consistency checking on variability models. In: Benavides, D., Batory, D.S., Grünbacher, P. (Eds.), Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS'10, In: ICB-Research Report, vol. 37, Universität Duisburg-Essen, pp. 171–174.

Vierhauser, M., Grünbacher, P., Egyed, A., Rabiser, R., Heider, W., 2010b. Flexible and scalable consistency checking on product line variability models. In: Proceedings of the 25th International Conference on Automated Software Engineering. ASE'10, ACM, pp. 63–72. http://dx.doi.org/10.1145/1858996.1859009.

Wada, H., Suzuki, J., Oba, K., 2008. A model-driven development framework for non-functional aspects in service oriented architecture. Int. J. Web Serv. Res. 5 (4), 1–31. http://dx.doi.org/10.4018/JWSR.2008100101.

Wang, J., Kim, S.-K., Carrington, D., 2008. Automatic generation of test models for model transformations. In: Proceedings of the 19th Australian Conference on Software Engineering. ASWEC'08, IEEE, pp. 432–440. http://dx.doi.org/10.1109/aswec.2008.4483232.

Wąsowski, A., Berger, T., 2023. Domain-Specific Languages: Effective Modeling, Automation, and Reuse. Springer, http://dx.doi.org/10.1007/978-3-031-23669-3.

Wei, L., Shuyu, L., 2013. Research on verification method of AADL behavior model based on BIP. In: Proceedings of the 5th International Conference on Computational and Information Sciences. ICCIS'13, IEEE, pp. 1987–1990. http://dx.doi.org/10.1109/iccis.2013.519.

Weidmann, N., 2018. Tolerant consistency management in model-driven engineering. In: Companion Proceedings of the 21st International Conference on Model Driven Engineering Languages and Systems. MoDELS'18, ACM, pp. 192–197. http://dx.doi.org/10.1145/3270112.3275339.

Weidmann, N., Anjorin, A., 2021. Schema compliant consistency management via triple graph grammars and integer linear programming. Form. Asp. Comput. 33 (6), 1115–1145. http://dx.doi.org/10.1007/S00165-021-00557-0.

Weidmann, N., Anjorin, A., Leblebici, E., Schürr, A., 2019. Consistency management via a combination of triple graph grammars and linear programming. In: Proceedings of the 12th International Conference on Software Language Engineering. SLE'19, ACM, pp. 29–41. http://dx.doi.org/10.1145/3357766.3359544.

Westfechtel, B., Greiner, S., 2018. From single- to multi-variant model transformations: Trace-based propagation of variability annotations. In: Proceedings of the 21st International Conference on Model Driven Engineering Languages and Systems. MoDELS'18, ACM, pp. 46–56. http://dx.doi.org/10.1145/3239372.3239414.

Wijs, A., Engelen, L., 2012. Incremental formal verification for model refining. In: Proceedings of the 9th Workshop on Model-Driven Engineering, Verification and Validation. MoDeVVa'12, ACM, pp. 29–34. http://dx.doi.org/10.1145/2427376.242738.

Wijs, A., Engelen, L., 2013. Efficient property preservation checking of model refinements. In: Piterman, N., Smolka, S.A. (Eds.), Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'13, In: LNCS, vol. 7795, Springer, pp. 565–579. http://dx.doi.org/10.1007/978-3-642-36742-7_41.

Yang, Z., Hu, K., Ma, D., Bodeveix, J.-P., Pi, L., Talpin, J.-P., 2014. From AADL to timed abstract state machines: A verified model transformation. J. Syst. Softw. 93, 42–68. http://dx.doi.org/10.1016/j.jss.2014.02.058.

Yang, S., Xiao, H., Li, Z., Xie, X., 2024. Deriving and verifying B specifications from problem frames models via model transformation. In: Workshops Proceedings of the 32nd International Requirements Engineering Conference. RE'24, IEEE, pp. 206–213. http://dx.doi.org/10.1109/rew61692.2024.00033.

Zhang, F., Zhao, Y., Ma, D., Niu, W., 2017. Formal verification of behavioral AADL models by stateful timed CSP. IEEE Access 5, 27421–27438. http://dx.doi.org/10.1109/access.2017.2770323.

Zhu, L., Liu, Y., 2009. Model driven development with non-functional aspects. In: Proceedings of the 15th Workshop on Aspect-Oriented Requirements Engineering and Architecture Design. EA'09, IEEE, pp. 49–54. http://dx.doi.org/10.1109/EA.2009.5071584.