Software and Systems Modeling manuscript No.
(will be inserted by the editor)

Recommender Systems in Model-Driven Engineering

A Systematic Mapping Review

Lissette Almonte · Esther Guerra · Iván Cantador · Juan de Lara

Received: date / Accepted: date

Abstract Recommender systems are information filtering systems used in many online applications like music and video broadcasting and e-commerce platforms. They are also increasingly being applied to facilitate software engineering activities. Following this trend, we are witnessing a growing research interest on recommendation approaches that assist with modelling tasks and model-based development processes.

In this paper, we report on a systematic mapping review (based on the analysis of 66 papers) that classifies the existing research work on recommender systems for model-driven engineering (MDE). This study aims to serve as a guide for tool builders and researchers in understanding the MDE tasks that might be subject to recommendations, the applicable recommendation techniques and evaluation methods, and the open challenges and opportunities in this field of research.

Keywords Model-Driven Engineering · Recommender Systems · Systematic Mapping Review

Lissette Almonte
Modelling & Software Engineering Research Group
Universidad Autónoma de Madrid (Spain)
E-mail: Lissette.Almonte@uam.es

Esther Guerra
Modelling & Software Engineering Research Group
Universidad Autónoma de Madrid (Spain)
E-mail: Esther.Guerra@uam.es

Iván Cantador
Information Retrieval Group
Universidad Autónoma de Madrid (Spain)
E-mail: Ivan.Cantador@uam.es

Juan de Lara
Modelling & Software Engineering Research Group
Universidad Autónoma de Madrid (Spain)
E-mail: Juan.deLara@uam.es

1 Introduction

Recommender systems (RSs) [3] are information filtering systems that aim to predict the preferences of users for a given set of items, with the purpose of offering a typically prioritised list of potentially interesting items. RSs are widely used by commercial applications such as music and video broadcasting platforms, e-commerce sites and social networks, and they are increasingly being used to help developers with software engineering activities [113]. For example, we can find RSs that help in choosing appropriate third-party programming libraries [95, 135], recommend API method invocations [94], suggest code refactorings [30], propose features for mobile apps [58], and assist on the evaluation of change impact analysis [18], to name a few.

Modelling is fundamental in software engineering and central to some software development approaches like model-driven engineering (MDE) [19,122]. In MDE, models are the primary assets of the development process, since they are used for analysis, validation, simulation and code generation of the applications to be built, among other activities. The rationale of MDE is to improve software quality and to reduce accidental complexity and development times [62]. Following the trend in software engineering, in recent years, there have been proposals of RSs to assist in modelling tasks [5, 26, 100] and other activities in the MDE process [111, 119].

In the state-of-the-art, we find surveys of RSs and their associated techniques [65, 112], which review fundamental techniques for constructing RSs. We also find literature reviews on the use of RSs for software engineering [41, 78], centered on RSs for code-based activities. However, perhaps because the use of RSs for MDE is an emerging research topic, to the best of our knowledge, there are no systematic studies yet analysing how

RSs can be designed and employed to assist in MDE tasks. To fill this gap, this paper presents a systematic mapping review that covers publications ranging from 2004 to 2020 from the main digital libraries. Our study aims to answer the following research questions (RQs):

RQ1*: In which ways can recommender systems assist in the different tasks within MDE processes?*
RQ2*: Which recommendation techniques are most commonly used to support MDE tasks, and how are recommenders for MDE evaluated?*
RQ3*: What are the main opportunities in recommender systems for MDE solutions?*

We have selected 66 relevant papers from an initial set of 1,456 papers, and classified them under four dimensions: domain, tooling, recommendation and evaluation. For RQ1, we have found that most approaches are directed to complete and repair artefacts, and work over models. Many RSs are language-independent, while the language-dependent ones frequently target UML or process modelling notations. For RQ2, we have found that most RSs are knowledge-based followed by content-based, and offline experiments are the most common evaluation approach. Finally, for RQ3, we have found that there are hardly any RSs for model transformations or code generators; and few RSs target creating, reusing or finding artefacts. Moreover, we have identified several research opportunities including the need for effective repositories of MDE artefacts that mitigate the current lack of data; techniques for adapting RSs to the user's needs; mechanisms to exploit the crowd knowledge via collaborative filtering; the user-based evaluation of RSs within MDE; and the investigation of mechanisms for the effective integration of RSs with MDE tools and low-code platforms.

Given the increasing importance that RSs are gaining in software engineering [113], we expect a similar trend in recommenders for modelling and MDE tasks, as the increase in the number of papers on this topic during the last years shows (cf. Figure 2). Hence, our study may be useful for tool builders and researchers to understand the tasks that can be subject to recommendations, the applicable recommendation techniques and their evaluation methods, and the open challenges in this field of research.

The remainder of this paper is organised as follows. First, Section 2 provides background on RSs, and Section 3 overviews the main concepts and tasks within MDE. Then, Section 4 describes the scope and methodology of our systematic mapping review. Next, Section 5 reports on existing works that describe RSs to assist in modelling tasks. Section 6 discusses the results of our review, answers the research questions, analyses threats to the validity of the study, and describes open challenges and interesting research directions. Finally, Section 7 concludes with a summary.

2 Recommender Systems

In this section, we first provide an overview of RSs (Section 2.1), then we describe the main types of recommendation techniques (Section 2.2), and lastly, we present the most frequent methodologies and metrics for evaluating RSs (Section 2.3).

2.1 Introduction to recommender systems

RSs are software tools and associated techniques that suggest items considered relevant for a particular user [3], usually in scenarios or applications where the space of items is very large and item search and selection are difficult or even overwhelming to the user. For this purpose, RSs explicitly or implicitly gather information about the user's preferences for a set of items (e.g., movies, songs, books or products), and subsequently use the collected information to make personalised predictions on items relevant to a target user, such as which movie to watch or which book to read next. Making use of information retrieval and machine learning techniques, RSs facilitate decision-making in domains where there are many options to choose from. Instead of requiring users to specify their interests by means of a query, these systems proactively suggest items of potential relevance to the users.

In general, RSs follow a process that encompasses three main steps:

1. Collecting relevant user information;
2. Learning from the collected information to build user profiles; and
3. Applying a heuristic function or a previously built model to select and rank the items that the user is most likely to prefer.

In order to present personalised item suggestions to a user, RSs build a *user profile* that captures past choices and preferences of the user. This information can be either *explicitly* provided by the user or *implicitly* inferred by the system. Explicit feedback refers to preference statements made by the users about items they know, and which are typically stored as numeric *ratings* or unary/binary values (e.g., *likes* and *thumbs up/down*). In contrast, implicit feedback is inferred by observing and mining user interactions within the system, such as previous search queries, product purchases

and mouse clicks, among others. Other features that can be used to model the preferences of users include demographic data, personality traits, emotional states, and trust relationships [112].

In a similar way, RSs characterise the items that may be recommended by means of *item profiles*. These profiles take different item attributes into account, such as metadata and text features extracted from item descriptions or textual contents [77].

2.2 Types of recommender systems

RSs are commonly classified into the following major categories, depending on how they generate personalised recommendations:

- *Content-based* systems, which recommend items that are similar to other items the target user liked in the past [77];
- *Collaborative filtering* systems, which base their suggestions on the items liked by "similar" people to the target user [96, 120];
- *Knowledge-based* systems, which exploit domain knowledge to describe and relate users and items for providing personalised recommendations [22];
- *Context-aware* systems, which consider the current user's context (e.g., location, time, weather) to enrich personalised recommendations [4];
- *Social-based* systems, which analyse and exploit the social network connections of the target user to generate recommendations [46];
- *Demographic-based* systems, which use demographic data to represent user and item profiles considered in the recommendation generation process [102]; and
- *Hybrid* systems, which combine two or more of the previous types of RSs [23].

Additionally, RSs can be categorised according to the algorithmic approach they use to compute the relevance of items [3]. In this regard, we can distinguish between two types of systems:

- *Memory-based* systems, where the relevance of items is estimated through heuristic formulae [120]; and
- *Model-based* systems, which predict item relevance by using a data-based model built via machine learning techniques, e.g., matrix factorisation [70] or neural networks [48].

The following subsections explain in more detail the above categories and types of RSs, which will be considered in our review.

2.2.1 Content-based recommenders

Content-based (CB) systems recommend similar items to those items the target user liked in the past [3]. They use item attributes or features to represent both user and item profiles and establish the corresponding user/item similarities. In general, they consider textual information (e.g., keywords, metadata and social tags) to build the user and item profiles [77].

A CB recommender is able to provide accurate personalised suggestions when it has enough information about the target user's preferences, since content similarities can be easily established. Moreover, it is capable of suggesting items for which no preferences have been expressed yet (i.e., cold items), since recommendations are generated via content-based item similarities.

However, this type of RSs has certain disadvantages. One of them is the *overspecialisation* problem, in which the user is exposed to items that are very similar to the ones the user already knows, limiting the discovery of diverse, relevant items. In this sense, CB recommenders are not suitable for domains and applications where, at a certain point, the user has to be suggested novel, fresh or even unexpected (serendipitous) recommendations, e.g., in the news articles domain. Another drawback of these systems is the *new user cold-start* problem, as a RS needs a considerable amount of users' preferences before it can provide well-suited recommendations.

2.2.2 Collaborative filtering recommenders

Collaborative filtering (CF) systems make suggestions to the target user based on items preferred by like-minded people [3]. They rely on the feedback (commonly ratings) that users give about the items. Hence, user and item similarities are established via explicit or implicit rating-based similarities and patterns [48, 120].

Differently to CB approaches, a CF system is able to provide novel and diverse recommendations for the target user. Even in situations of rating sparsity, CF has shown better performance than CB in many real-world applications, and represents the most widely used approach for providing personalised recommendations [70].

Nonetheless, similarly to CB approaches, CF recommenders suffer from the new user cold-start problem, i.e., they need to have enough ratings to provide accurate recommendations. CF also manifests the so-called *item cold-start* problem, since an item can only be recommended after being rated. Moreover, CF is affected by situations of high sparsity, where the number of collected ratings is very small with respect to the total number of possible ratings given by users to items.

2.2.3 Knowledge-based recommenders

Knowledge-based (KB) systems recommend items using domain-specific knowledge about how item attributes and features could meet the user's needs and interests [22]. Many recommendation approaches can be categorised as KB. Among them, two types of approaches have gained great interest in the literature: case-based and constraint-based [112]. Case-based systems address the recommendation task via case-based reasoning methods, which aim to solve a new problem (i.e., a new case) by remembering previous similar cases and reusing knowledge about them. Constraint-based systems, on the other hand, predominantly exploit knowledge, commonly expressed by means of explicit rules, on how user requirements are related to the item attributes and features.

The main advantage of KB systems is their capability of providing and explaining accurate recommendations that entail a deep understanding of the user's preferences, which cannot be achieved by CB and CF approaches. Although KB systems usually do not suffer from cold-start problems, they are affected by the so-called *knowledge acquisition bottleneck*. This consists on the need of learning models or domain experts to model and build the used knowledge bases. Additionally, KB recommenders usually are ad-hoc solutions to particular problems, and thus their generalisation to other problems or domains is difficult or not possible.

2.2.4 Hybrid recommenders

Hybrid systems make use of two or more recommendation methods, such as CB and CF, to take advantage of their benefits and avoid some of their limitations [23]. Because of this, many real-world RSs are hybrid. Without entering into details, we can identify three main ways to implement a hybrid system:

- Incorporating some feature of one recommendation method into another one, e.g., a CF strategy that uses CB similarities;
- Combining the recommendations generated separately by two methods, e.g., via ranking aggregation and diversification techniques; and
- Building a unifying recommendation model that incorporates characteristics of distinct methods, e.g., a matrix factorisation model with both collaborative and content-based features.

2.2.5 Other recommenders

There are other types of RSs that can be considered orthogonal to CB, CF and KB systems, since they exploit particular data following CB and CF strategies. Special attention can be drawn to the next recommenders:

- *Context-aware recommenders*, commonly abbreviated as CARS, take into consideration contextual information associated or influencing to user preferences when generating personalised recommendations [3]. Quoting Dey [32], "*context is any information that can be used to characterise the situation of an entity.*" In CARS, context commonly refers to circumstances in which recommendations are produced, such as the time, the weather and the user's current location.
  CARS are appropriate for applications where contextual variables determine or have a high impact on the relevance of the suggested items. For instance, in a RS for travelling, the vacation recommendations for the winter season can be very different from those generated for the summer [4]. Naturally, not all contextual information available might be relevant, and the fact that contextual factors and data sources differ from application to application makes CARS difficult to implement and evaluate. Moreover, CARS may require extra effort from the users, who may need to provide information about certain contextual conditions, such as their current mood and companion.
- *Social-based recommenders* generate personalised recommendations in social media [46]. A widely explored approach in these systems is the exploitation of explicit relationships between users in social networks. In this sense, many solutions are based on the so-called trust-based model, where the social influence and trust of users are established and propagated through the social network. In fact, research supports the theory that social trust can be used as a positive way to generate explanations of provided recommendations [133]. Social-based methods perform well when used together with other recommendation approaches, like CB or CF, since social network information can help in dealing with the user and item cold-start problems [112].
- *Demographic-based recommenders* make use of demographic data about the users, e.g., age, gender and address [102]. Taking this information into account, the recommendation algorithms identify users or items that are demographically compatible with the target user. These systems assume that users with similar demographic attributes may rate similarly, and have been applied to alleviate cold-start problems of traditional recommendation approaches.

2.3 Evaluation of recommender systems

RSs need to be evaluated at different phases of their life-cycle. At design time, it is necessary to assess the adequacy of the selected recommendation approach for the application at hand. This evaluation is done via *offline experiments* by running potential recommendation algorithms on the same dataset of user-item interactions (i.e., the rating matrix) and comparing their performance by means of several metrics [45]. This type of evaluation permits measuring the quality of the algorithms in accomplishing a recommendation task, but neglects user-centred aspects related to usage satisfaction, acceptance and experience with the system.

In offline experiments, as commonly done for evaluating machine learning methods, the dataset is usually split into two subsets: the *training* set, used to build the recommendation model, and the *test* set, used to evaluate the built model. Sometimes, a third subset, the *validation* set, is also used for parameter tuning of the trained model, before testing.

There are two main types of recommendation tasks that can be evaluated: the *item rating prediction* (which is in disuse in the RSs community) and the *item ranking generation* (or top-N recommendation). Each of them has specific metrics, sometimes adopted and adapted from the machine learning and information retrieval areas. In the rating prediction task, the objective of a RS is to accurately predict the numeric value of the rating a user would give to an item, and thus metrics such as the mean absolute error (MAE) and the root mean square error (RMSE) are considered. In the ranking generation task, the goal of a RS is to provide the user with a personalised ranked list of relevant items, with special interest in the items at the first (top) positions of the list. In this case, metrics oriented to measure the accuracy of the ranking are used, e.g., precision, recall, mean reciprocal rank (MRR), and normalised discounted cumulative gain (nDCG) [45]. These accuracy metrics can be complemented with metrics for other ranking characteristics, such as diversity, novelty and coverage [14].

RSs should also be evaluated after deployment. This can be done via *online experiments*, which usually are user-centric [67]. The built system is deployed in a real environment and tested by end-users in real-time, commonly online. In these experiments, a widely used evaluation methodology is *A/B testing*, where two versions A and B of the system are deployed, and one of them implements the recommendation algorithm or functionality that is being evaluated. After a period of time, the user feedback and behaviour recorded in both systems are analysed and compared according to certain metrics. There is also the possibility of performing a *user study* where a prototype of the system is deployed in a controlled setting and evaluated with a reduced set of users, maybe recruited by crowd-sourcing. User studies can also follow the A/B testing methodology, and incorporate online questionnaires to gather the usage satisfaction and opinions about the system and its functionalities.

3 Model-Driven Engineering

In this section, we provide a brief overview of the main concepts, artefacts and tasks within MDE solutions. We do not aim to be exhaustive, but to provide the necessary context to understand the kind of support needed from RSs in MDE. The interested reader can see [19] for a more detailed account of MDE.

3.1 MDE artefacts

Figure 1 shows a schema with the main elements of MDE solutions. In MDE, models are the main assets, from which other artefacts – like code or other models – may be derived in an automated way. Models conform to a meta-model, which defines the modelling language syntax and determines the set of models that are valid. Meta-models comprise a structural diagram plus additional constraints formulating restrictions that cannot be expressed diagrammatically. The structural diagram is defined as a class diagram, frequently using standards like the Meta-Object Facility (MOF) [85] and implementations like the Eclipse Modeling Framework (EMF) and Ecore [131]. Constrains are described using constraint languages like the Object Constraint Language (OCL) [97].

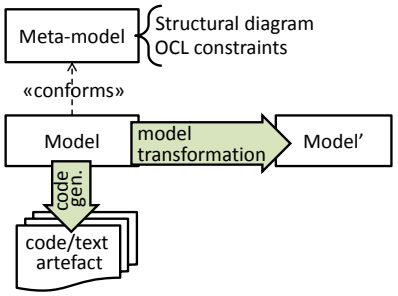

Fig. 1 Main elements of MDE solutions.

In addition to models, MDE solutions may include model transformations to modify existing models (e.g., performing refactorings or optimizations) or to create new models out of existing ones (e.g., creating a design

model from a requirements model). Model transformations are defined using either specialized transformation languages like ATL [59] or QVT [107], general-purpose languages like Java, or technologies like XSLT [79].

Finally, textual artefacts – like code, configuration files or documentation – can be produced from models using code generators. These are typically written in specialized template languages, such as Acceleo [2] or EGL [115].

3.2 MDE tasks

Model-based software solutions involve the creation of models using a modelling language. Modelling languages can be either general-purpose like the UML, or domain-specific languages created for a particular domain. Therefore, in MDE, engineers may need to *create* the following kinds of artefacts: models, meta-models (i.e., modelling languages), model transformations, and code generators.

As in any software engineering process, in MDE, it is desirable to be able to *reuse* existing artefacts to avoid their creation from scratch. This requires the ability to *find* similar artefacts, or fragments of them, in existing repositories.

The syntactic correctness of models is crucial to enable sound solutions and be able to apply model transformations and code generators on them. Therefore, it is important to *complete* partial models to become conformant to their meta-model. This completion process applies not only to models, but also to model transformations, code generators and meta-models, as these can be seen as models that have a specific semantics and conform to their own meta-models.

MDE artefacts may have errors, and so, they may need to be *repaired* either syntactically to conform to their meta-models, or semantically to conform to some specified requirements. Meta-model/model co-evolution [38] is a particular case of the latter, whereby a meta-model evolves to accommodate changing or new requirements, and the broken models need to be repaired to make them conform to the new meta-model version. Other tasks related to repairing artefacts include the creation of input test data (e.g., input models for testing model transformations) [13], oracles (e.g., transformation contracts) [44], and fixes [119].

4 Survey Methodology and Scope

Following accepted guidelines for systematic mappings [104, 105, 139], we have performed a systematic mapping review to analyse how pervasive is the use of RSs to support modelling and MDE, identifying the tasks that have been subject to recommendations and the recommendation techniques most frequently applied. The surveyed articles typically introduce RSs that facilitate some modelling or MDE task.

To collect articles on this topic, we sought into Scopus, the ACM digital library and the Web of Science using a formal query comprising 23 terms. The query retrieves articles whose title, abstract or keywords contain at least one term related to RSs, and at least one term related to modelling or MDE.

Table 1 shows the considered terms, so that the retrieved articles should contain in their title, abstract or keywords some term from each column of the table. We included terms like *model completion*, *model reuse* and *model repair* as related to RSs, because we detected that some approaches did not use standard terminology and vocabulary of the RSs area (cf. Section 2). However, they pursue the same goal of recommending a reduced set of modelling items or actions among a large set of possible ones. We executed the query in September 2020, and only considered peer-reviewed papers written in English and published in journals, conferences, workshops and book chapters.

Table 1 Terms used in the formal search query. Articles must contain in their title, abstract or keywords at least one term from each column.

Recommender Systems / Purpose	Modelling / MDE
recommender	model-driven
recommendation	domain-specific language
model completion	state machine
model reuse	model transformation
model repair	code generation
transformation completion	code generator
transformation reuse	unified modelling language
transformation repair	UML
generator completion	
generator reuse	
generator repair	
quick fix	
quick fixes	
assistant	
assistance	

Table 2 shows a summary of the search results. The query initially retrieved 1,456 documents: 979 from Scopus, 316 from the ACM digital library, and 161 from the Web of Science. After removing duplicates, 1,175 unique documents remained.

Next, the unique documents were filtered in two subsequent phases. In the first phase, four reviewers examined the abstracts of all documents to identify which ones proposed some kind of recommendation for

Table 2 Research papers retrieved per database.

Detail	Num. Papers
Databases queried	
Scopus	979
ACM	316
Web of Science	161
First revision phase	
Total retrieved	1,456
Unique	1,175
Discarded	1,024
First selection	151
Second revision phase	
Relevant	53
Not available	9
Not relevant	89
Snowballing	
Snowballing papers	13
Total relevant	66

modelling tasks. The reviewers were two professors specialised in MDE, one professor with expertise in RSs and information retrieval, and one doctoral student in both research areas. Overall, 151 documents were selected by at least one of the reviewers and were moved to the next phase, and 1,024 were discarded for being unrelated to our study. In both phases, we used inclusion criteria based on quality (peer-reviewed papers), language (papers written in English), and focus (we discarded papers unrelated to modelling, recommenders or modelling assistants, as well as papers focussing on using MDE techniques for creating RSs).

In the second phase, the 151 selected papers were carefully read. From these papers, 53 were considered relevant for the study, 9 were unavailable, and 89 were not relevant since they either proposed RSs for activities not related to models or modelling, or did not describe recommenders or modelling assistants.

Finally, we conducted a snowballing process [105], analysing related works in the bibliography of the selected papers. This resulted in the selection of 13 additional relevant papers. Overall, a final set of 66 relevant papers was considered, ranging from 2004 to September 2020, covering almost 16 years of research. These 66 documents account for 51 different approaches, as in some cases, there are several documents covering the same approach. Figure 2 shows the distribution of papers over the studied period of time. We observe an increasing trend that likely suggests a growing interest in the field. Please note that the query may not fully cover the year 2020 as it was executed on September 2020.

Figure 3 categorises the relevant papers according to the publication type. Most papers are from conferences and workshops, which denotes that the research

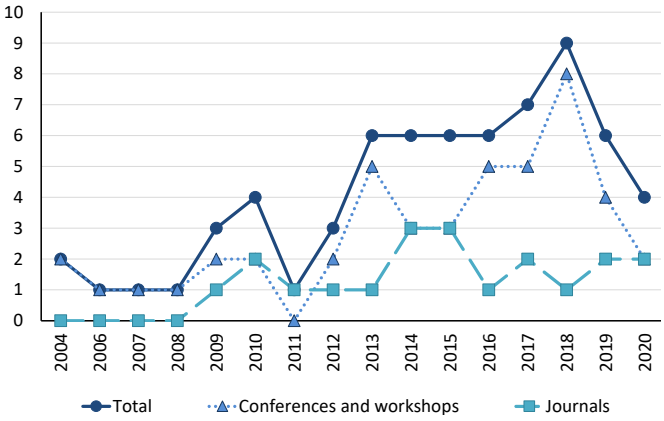

Fig. 2 Relevant papers per year.

area is still young. The most frequent conferences and journals of publication are the International Conference on Model-Driven Engineering Languages and Systems (MoDELS) (9 papers), the International Conference on Model-Driven Engineering and Software Development (MODELSWARD) (6 papers), the International Conference on Software Engineering (ICSE) (4 papers), and the Journal on Software and Systems Modeling (SoSyM) (3 papers). Hence, the primary publication venues are devoted to software engineering and modelling.

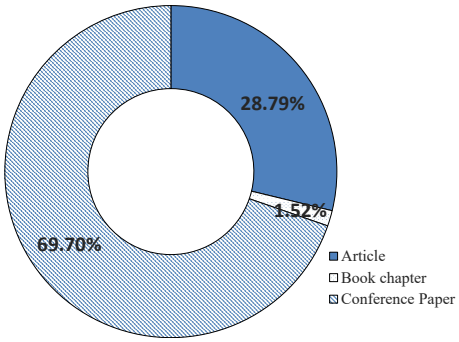

Fig. 3 Distribution of works depending on publication type.

In the next section, we propose a classification of the works along four dimensions, and analyse the papers with respect to these categories.

5 Recommender Systems in MDE

We organise our review according to the four dimensions of the feature model [61] depicted in Figure 4: *domain*, *tooling*, *recommendation* and *evaluation*.

The *domain* dimension encompasses analysis variables in the context of MDE applications, such as the type of artefact that is subject of the recommendation

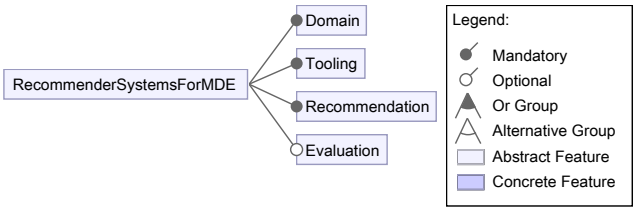

Fig. 4 Dimensions for analysing the use of RSs in MDE.

and the purpose of the recommendation. The *tooling* dimension includes aspects related to the recommendation tool, such as its maturity, its support for the integration with other MDE tools, and its proactiveness to request or apply recommendations. The *recommendation* dimension entails variables used in the RSs area to characterize how recommendations are generated [106], such as the recommendation method, the user preferences used to calculate the recommendation, the recommended items and the recommendation tasks. Lastly, the *evaluation* dimension refers to the methodologies and metrics used to evaluate the recommenders.

These dimensions represent assessment criteria from both the MDE and the RSs perspectives. The dimensions and variables are orthogonal, although some of them may have cross dependencies in some of the surveyed cases. For instance, the recommendation purpose may influence the metrics used to evaluate a given recommender. However, these dependencies have less impact than those between the variables belonging to a given dimension, which are addressed in the analysis presented herein.

The next four subsections analyse and classify the selected papers along the dimensions and inner variables.

5.1 Domain

A fundamental aspect of any RS is its application domain. This comprises the three orthogonal features that we present in the feature model of Figure 5.

First, we consider the type of artefact that is the subject of recommendation: *model*, *meta-model*, model *transformation* or *code generator*. These are the four main elements of most MDE solutions [19].

Second, we distinguish whether the RS is *language-independent*, or on the contrary, it is tied to a particular language for modelling (e.g., UML [136], Simulink [128]), meta-modelling (e.g., MOF [85], Ecore [131]), model transformation (e.g., ATL [59], QVT [107]), or code generation (e.g., Acceleo [2], EGL [115]).

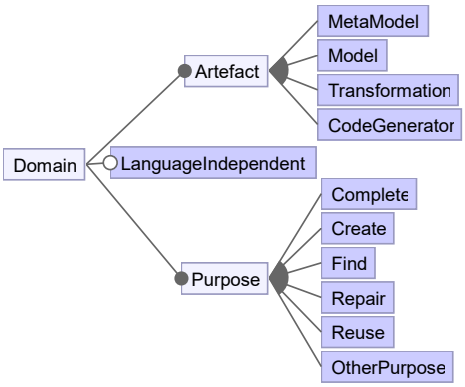

Fig. 5 Domain dimensions for RSs in MDE.

Finally, we look at the purpose of the RS, that is, the kind of task that the recommender facilitates[1]. As we will see later, the reviewed papers target one or more of the following six types of tasks, introduced in Section 3: *complete*, *create*, *find*, *repair*, *reuse* and *other purpose*. When the purpose is *complete*, the artefact already exists and the RS provides suggestions on how to extend it. When the purpose is *create*, the recommender helps in constructing the initial version of a new artefact from scratch. If the purpose is *find*, the RS facilitates the discovery of relevant elements or artefacts within a repository. Recommenders targeting *repair* tasks suggest solutions to fix errors in an existing artefact. These solutions may imply the creation, deletion or modification of different elements inside the artefact. When the purpose is *reuse*, the system helps in reusing an existing artefact (or part of it) within another artefact. This task goes beyond *find* as the recommender provides assistance in integrating the reused artefact in the new context. Finally, in *other purpose* we collect the tasks with a purpose different from the mentioned before.

Table 3 classifies the surveyed papers by purpose and artefact type, and marks the language-independent approaches with an asterisk (*). Since some systems can be used with various purposes, they can appear in several cells of the table, sometimes with different language-independent marks.

Overall, we can see that there are virtually no recommenders for code generators, and recommenders that help in creating new artefacts from scratch are also scarce. Most RSs are for models, especially for model completion and model repair, and the context in the latter case is sometimes model/meta-model co-evolution.

[1] We use purpose and task interchangeably, though the latter can be more detailed. For example, for repairing (a purpose) we can find fine-granular tasks, like creating input test data, oracles and selecting fixes (cf. Section 3).

Table 3 Purpose of recommendation vs. recommended artefacts (approaches marked with * are language-independent).

Purpose \ Artefact	Meta-Model	Model	Transformation	Code Generator
Complete	DoMoRe [5, 6] Hermes [34–36] Kögel et al. [68, 69] Refacola [130]	Baya [27] Deng et al. [31] DiaGen* [82] DIG MDE [92] DoMoRe* [5, 6] Elkamel et al. [37] Heinemann [49] Hermes* [34–36] IPSE [40] Kermeta* [86] Kögel et al.* [68, 69] Koschmider et al. [50, 51, 71] Li et al. [75] PME* [99] Rangiha et al. [110] RapMOD [72, 73] Refacola* [130] Savary-Leblanc [121] Sen et al.* [126, 127] SimVMA [132] SMART [47]	AXSM [52] CONVErT [10]	
Create	DSL-maps [103]	UCcheck [9]		
Find	Extremo [123–125]	Cerqueira et al. [26] Extremo* [123–125] Matikainen et al. [81] SBPR [63, 64]		
Repair	Batot et al. [12] Clarisó et al. [28] PARMOREL [11, 53] Refacola [130]	AMOR [21] Anguel et al.* [8] ASIMOV* [38] BAM [137] BPMoQualAssess [60] B-repair [24] DiaGen* [82] DPF* [108] IntellEdit* [93] Mani et al.* [79] MDSafeCer [87] Nassar et al.* [91] PARMOREL* [11, 53] Refacola* [130] ReVision* [98] SMART [47]	anATLyzer [117–119]	Mani et al. [79]
Reuse	Hermes [34–36]	Hermes* [34–36] Koschmider et al. [50, 51, 71] Paydar et al. [100, 101] REBUILDER [42] SimVMA [132]	Refactory [111]	
Other Purpose		Bobek et al. [17] MAGNET [16] ModBud [116]		

In the following, we analyse the application domain of the approaches grouped by their purpose.

Complete. Most approaches whose purpose is completing an artefact target model completion, and among them, four are also applicable to meta-models. In addition, two approaches deal with completing model transformations.

Approaches to model completion can be classified into two categories. The first one comprises techniques to recommend how to extend a partially specified model to make it correct (i.e., the recommended complete model satisfies every specified domain and meta-model well-formedness constraint). The proposed model completions are typically computed us-

ing *search-based* techniques, for example with solvers based on Alloy [54] (a constraint solver over models), Prolog (a logic-based programming language) or via rules. DiaGen [82], DIG MDE [92], IPSE [40], Kermeta [86], Refacola [130] and the work by Sen et al. [126, 127] belong to this category. DiaGen generates possible completions based on hyper-graph grammar rules, and the others use Prolog or Alloy for this task. This search may have a high computational cost. For this reason, when a partial model cannot be completed automatically because of its complexity, DIG MDE identifies the failing constraints and suggests how to manually change the model to enable its completion.

The second model completion category comprises works providing step-wise recommendations on how to evolve a given model. This model does not need to be partial, as in the first category. Suggestions usually come from repositories of existing models, fragments or patterns. For example, SimVMA [132] provides step-wise suggestions to evolve Simulink models based on model clone analysis; Heinemann [49] recommends elements defined in model libraries (e.g., blocks from Simulink libraries) based on data mining existing models; the approach by Kögel et al. [68, 69] recommends model changes applicable to the same context of the last model change; DoMoRe [5, 6] suggests domain concepts and names for new model elements; RapMOD [72, 73] offers auto-completion actions for (UML) graphical models, similarly to the vision paper [121]; Elkamel et al. [37] recommend UML classes that are similar to the ones in the UML class diagram being developed; Li et al. [75] and Deng et al. [31] recommend activity nodes for process models; Rangiha et al. [110] recommend tasks and actor roles in a social process-modelling tool; Koschmider et al. [50, 51, 71] recommend process fragments to complete a process model; Baya [27] recommends mashup model patterns based on the context, the user and different expert recommendations, and helps in weaving the selected pattern into the partial model under development; and Hermes [34–36] permits building Eclipse-based RSs that help in completing models with recommended elements from other models in a repository.

Instead of profiting from repositories of models, PME's recommendations are based on an analysis of the language meta-model [99]. PME enables proactive (graphical) modelling, meaning that plausible modifications according to the models' meta-model are automatically applied, and the user is prompted only when several optional modifications exist. SMART [47] supports the use of test-driven development to create UML diagrams (class, use cases, state machines and sequence diagrams). It uses an action language to specify behavioural tests, and when a test fails, it suggests ways to complete the model to make it pass the test.

Among the previous model completion approaches, four can also be applied to meta-models. DoMoRe works on domain models, like UML class diagrams and entity-relationship diagrams, and therefore can be used to add concepts of a domain of interest to meta-models. The approach of Kögel et al. [69] recommends complementary changes to a user editing action, and can be applied at the meta-model level, e.g., to recommend generalization relations to a core super-class. Refacola is a refactoring constraint language and framework, extended to (meta-)model assistance in [130]. It is meta-level independent, providing assistance for completing partial (meta-)models to become syntactically correct. Finally, the Hermes framework can be configured with recommendation strategies. It is applicable to models within the EMF ecosystem, and hence to meta-models as well.

Regarding the completion of model transformations, CONVErT [10] synthesises transformation code starting from examples of source and target models and their correspondences. Correspondences are specified manually, but there is also a recommender of likely correspondences based on similarity heuristics such as the name, structure and neighbourhood of model elements. AXSM [52] is a mapping recommender integrated in a tool to build data transformations via declarative mappings, from which translators written in XSLT, Java or ATL can be synthesised. AXSM recommends potential mappings based on heuristics grounded on the data schemas and on prior user selections.

Create. Only two of the analysed works target the creation of artefacts, one for meta-models and the other for models. The first one is DSL-maps [103]. Given the requirements of a DSL expressed as a mind-map, DSL-maps recommends meta-modelling patterns addressing them. The designer can select patterns among the ranked suggestions, and the tool combines the patterns to synthesise an initial meta-model, which the designer can then refine. The second approach is a modelling assistant for use case diagrams called UC-check [9]. This tool has a wizard to create new use case diagrams using an existing one as a reference, from which suitable actors and use cases are recommended.

Find. The analysed papers include approaches to query repositories and suggest relevant artefacts for models and meta-models, but not for transformations or code generators. Extremo [123–125] is an extensible

tool-independent assistant that helps finding relevant information for models and meta-models out of heterogeneous data sources (e.g., ontologies, XML schemas, RDF data, meta-models), and the results are ranked according to their suitability for the user. The rest of approaches are specific for some kind of model: the RS of Cerqueira et al. [26] finds and recommends sequence diagrams that match the user preferences; Matikainen et al. [81] tackle the problem of recommending the state machine from a library that implements the best policy to control a robot; and SBPR [63, 64] recommends process models from a repository according to the user business profile in LinkedIn (e.g., skills, interests and current position).

Repair. Repair approaches have been proposed for all kinds of artefacts: models, meta-models, transformations and code generators.

Regarding model repair, most works aim to recommend fixes for inconsistencies found in a given model (i.e., violations of the model's meta-model cardinality or well-formedness constraints). These approaches differ either in the applied technique to compute and rank the repairs, or in the application domain. In particular, IntellEdit [93] ranks quick fix solutions to model inconsistency problems according to the least-change principle; PARMOREL [11,53] determines the model repair actions based on the user preferences and on the experience gained from repairing under different personalisation settings; the Diagram Predicate Framework (DPF) [108] and the approach by Nassar et al. [91] implement repairs as transformation rules; DiaGen [82] represents models as hyper-graphs and uses hypergraph patches to produce recommendations for repairing models; Refacola [130] uses constraint-based rules; BPMoQualAssess [60] provides guidelines to improve the actual value of quality metrics for business process models; B-repair [24] is specific to the B formal specification language and ranks the suggested repairs based on their estimated quality; ReVision [98] tracks model inconsistencies to the editing action originating them in the model history, and fixes this action to obtain a consistent model; MDSafeCer [87] detects missing information for supporting key evidence in process-based argumentations, and recommends how to resolve such deviations; ASIMOV [38] assists in the co-evolution of models and meta-models by proposing model co-evolution actions that a meta-modeller must have defined previously; and Anguel et al. [8] also tackle the co-evolution problem, but they automatically fix resolvable changes and recommend co-evolution actions to deal with non-resolvable changes.

There are also some model repair approaches that do not tackle model conformance, but they target other kinds of model-related problems. In particular, Mani et al. [79] compute repairs for input test models that make a code generator produce an incorrect output; in addition to *complete*, the suggestions for fixing behavioural tests in SMART [47] can also be classified as repairs; the Business Application Modeller (BAM) [137] permits specifying temporal rules for process models and, for some types of rules, it recommends how to fix their violations; and AMOR [21] is a model repository for model versioning that includes a recommender of possible resolutions for model conflicts.

With respect to meta-model repair, two of the works target OCL integrity constraints [12, 28]. Batot et al. [12] tackle the co-evolution of OCL constraints upon meta-model changes. Their approach recommends a ranked list of OCL modifications that are correctly typed by the new meta-model version, and minimise the number of changes and information loss. Clarisó et al. [28] repair OCL constraints which are too restrictive or too lax. Their method suggests weaker or stronger candidate versions of the problematic constraint, and the user can select one of them. In addition, two of the model repair approaches can be used to repair meta-models as well. PARMOREL allows repairing meta-models having duplicate attributes in related classes, or properties modelled both as attributes and as references [11]. Refacola [130], on the other hand, can help repairing syntactically incorrect meta-models, e.g., with inconsistent opposite or containment references (typical problems at the model level that can also happen in meta-models).

We found only one work supporting model transformation repair. This is anATLyzer [117–119], a tool integrated with the ATL IDE that identifies errors and recommends a ranked list of quick fixes to repair the transformation syntactically. Fixes are ranked taking into account the number of problems they solve, remaining errors and newly introduced errors.

Finally, we classify the approach by Mani et al. [79] as applicable to code generators because even if it suggests model repairs, these are applied in the context of code generation with XSLT.

Reuse. Recommenders in MDE have been applied to the reuse of models and transformations. Regarding model reuse, SimVMA [132] recommends Simulink models similar to the one that is being developed, and which the designer can import or clone for their reuse; REBUILDER [42] finds UML diagrams similar to a given query, and supports their full or partial composition into the given design; Paydar et al. [100,101] propose a reuse technique whereby the designer provides an input UML use case diagram, the most similar use

cases are retrieved from a model repository, and then the activity diagrams associated to these use cases are semi-automatically adapted to (i.e., reused in) the new usage context; Koschmider et al. [50, 51, 71] propose both a recommender of process model fragments, and an explicit search facility to retrieve complete process models or fragments and insert them in the current modelling context, adapting them if needed; and Hermes [34–36] can incorporate model search strategies to find model elements suitable for reuse. Being generic, Hermes can also be applied to meta-models.

As for transformation reuse, it is supported by Refactory [111]. This tool permits defining generic refactorings over role models, so that developers can reuse the refactorings on new languages by binding the role model elements into elements of the language meta-model. Refactory includes a recommender that helps in identifying possible bindings, likely starting from some manually bound elements to avoid a high number of suggestions.

Other purpose. The remaining papers have very specialised purposes. Bobek et al. [17] propose a recommender for process modelling, which suggests the elements of a configurable diagram (i.e., a process with variability) that should be included in the current modelled process. MAGNET [16] guides users on the next tutorials to speed up the learning curve of a modelling tool. Finally, ModBud [116] is an envisioned framework to build assistants that educate novice modellers on abstraction. Such assistants may provide recommendations on a constructed model by comparison with a prescriptive model devised by the assistant.

In Table 3, the language-independent approaches have an asterisk (*). We can see that most works are tied to a particular language, but a substantial amount of those applicable to models are language-independent. For this purpose, they are frequently defined over a meta-modelling framework – like the Eclipse Modeling Framework (EMF) [131], Kermeta [57], GME [74], or DPF [109] – which enables their application to models of any language defined within the framework. This is the case of [11, 34–36, 53, 68, 69, 86, 91, 93, 98, 99, 108, 123–127, 130]. In the case of DiaGen [82], language independence is achieved by representing models as hypergraphs, and language definitions as hyper-graph grammars. Other approaches are meta-level independent, and since meta-models are also models, such approaches are suitable for both meta-models and models [5,6,11,34–36,53,68,69,123–125,130]; however, when applied to meta-models, the approaches are dependent on the meta-modelling language used, like EMF's Ecore. Finally, language independence can be a gradual term. For example, DoMoRE is language-independent as it is applicable to arbitrary domain models, but it cannot deal with other types of models such as behavioural models.

Table 4 summarises the languages that the language-dependent approaches handle. Most are widely used languages, like UML diagrams (11 approaches), business process models (9 approaches), Ecore (7 approaches), Simulink (2 approaches), OCL (2 approaches), XSLT (1 approach) and ATL (1 approach). The rationale is that building a RS generally involves a high effort and may require from training data, which may pay off for widespread languages, but the development may be too expensive for lesser used domain-specific languages (DSLs). There are some exceptions of RSs for DSLs though, typically embedded in tools built by the researchers [10, 52, 92, 111].

Table 4 Languages targeted by recommender systems.

Language	Approach	Purpose
Meta-Model		
Ecore	DoMoRe [5, 6] DSL-maps [103] Extremo [123–125] Hermes [34–36] Kögel et al. [68, 69] PARMOREL [11, 53] Refacola [130]	Complete Create Find Repair Reuse
OCL	Batot et al. [12] Clarisó et al. [28]	Repair
Model		
Business process models	BAM [137] Bobek et al. [17] BPMoQualAssess [60] Deng et al. [31] Koschmider et al. [50, 51, 71] Li et al. [75] MDSafeCer [87] Rangiha et al. [110] SBPR [63, 64]	Complete Find Repair Reuse Other
DSL for embedded systems	DIG MDE [92]	Complete
Mashup models	Baya [27]	Complete
Simulink	Heinemann [49] SimVMA [132]	Complete Reuse
State/abstract machines	B-repair [24] Matikainen et al. [81]	Find Repair
UML behavioural diagrams	SMART [47]	Complete Repair
UML class/ structural diagrams	AMOR [21] Elkamel et al. [37] IPSE [40] ModBud [116] RapMOD [72, 73] REBUILDER [42] Savary-Leblanc [121]	Complete Reuse Other
UML sequence diagrams	Cerqueira et al. [26]	Find
UML use case diagrams	Paydar et al. [100, 101] UCcheck [9]	Create Reuse
Transformation		
ATL	anATLyzer [117–119]	Repair
DSL for refactorings	Refactory [111]	Reuse
Marama Torua	AXSM [52]	Complete
CONVErT	CONVErT [10]	Complete
Code Generator		
XSLT	Mani et al. [79]	Repair

5.2 Tooling

Next, we analyse the tool support of the approaches using the criteria shown in the feature model of Figure 6.

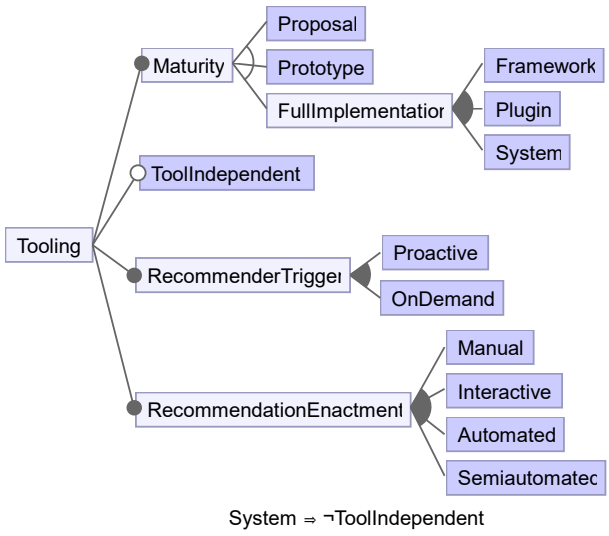

Fig. 6 Tooling dimensions for RSs in MDE.

First, we look at the maturity of the supporting tools. We distinguish between *proposals* with no implementation, *prototypes* built as proof-of-concepts of the proposed ideas, and mature tools that make a full implementation available either as a *framework*, a *plugin* or a *system*. Frameworks typically offer generic functionality that can be customised by manually written code (e.g., by subclassing). Plugins encapsulate functionality that complements other tools, such as the Eclipse IDE. Systems can be either complete new applications that incorporate recommendation facilities, or extensions of existing MDE tools with a RS.

Second, we classify the approaches as *tool-independent* if they can complement or be integrated into other MDE tools. The constraint in the feature model states that systems cannot be tool-independent, since the RSs are embedded in the tools themselves.

Third, RSs may trigger recommendations *on demand*, *proactively*, or both. In the first case, the user needs to explicitly start the recommendation process. In the latter case, the RS makes recommendations without user intervention, when certain conditions are met.

Finally, we analyse the support for enacting the recommendations. This can be *manual* if the RS provides a list of recommendations and it is up to the user to decide how to use them; *interactive* if the RS permits the user to select a recommendation, which then becomes applied to the given context; *automated* if the recommendation is automatically applied without user intervention; and *semiautomated* if the recommendation enactment is automated but the user may be prompted during the process, e.g., to input some value or decide between alternative options.

Table 5 classifies the revised papers according to these features. In the following, we discuss the different approaches attending to their maturity level, tool independence, recommender trigger, and recommendation enactment.

Maturity. The second column of Table 5 displays the maturity level of the approaches. When there are several incremental papers on the same approach, the column only shows the maturity achieved in the latest one (i.e., the highest maturity level). Among the 51 approaches, 4 (7.8%) are proposals with no implementation, 19 (37.2%) present prototypes as proof-of-concept, and the remaining 28 (55%) provide full implementations. Most full implementations are either plug-ins (13) or systems/extensions of systems (13), while frameworks (3) are less pervasive. We categorize the Hermes [34–36] framework for developing RSs as a plug-in as well, as it uses a plug-in architecture that exposes and profits from Eclipse extension points.

Tool independence. This feature applies to approaches that make a prototype or full implementation available, but not to proposals that have not been realized in practice (even though they may have the potential to become tool-independent). Most tool-supported RSs in MDE have been developed either as full software systems, or as extensions of the following existing systems: the ATL development environment [117–119], some data mashup tools [27], the Generic Eclipse Modeling System (GEMS) [92], the Ecore Diagram Editor [5, 6], DPF [108], DiaGen [82], Fujaba [40], AutoFOCUS3 [16], the AMASS platform [87], the AMOR model versioning system [21], Kermeta [86], the Generic Modeling Environment (GME) [99], Sparx Enterprise Architect [72,73], and the meta-modelling tool AToM^3 [126, 127]. All these approaches built as complete systems or system extensions are tool-dependent (84.31%). In some cases, the tools are implemented atop EMF to achieve generality. However, we only consider that an approach is tool-independent if, in addition, it provides explicit means to facilitate its integration with other tools. Under this perspective, only four (7.84%) approaches are truly independent from any modelling tool. We comment on these approaches next.

The framework developed by Batot et al. [12] recommends how to co-evolve OCL invariants upon Ecore meta-model changes (i.e., it is language-dependent);

Table 5 Recommender systems for MDE: Tooling. We use *n.a.* as abbreviation for *not applicable*.

Approaches	Maturity	Tool	Recommender Trigger		Recommendation		Enactment	
		Independent	OnDemand	Proactive	Manual	Interact.	Auto.	Semi-
AMOR [21]	Plug-in		✓			✓		
anATLyzer [117–119]	System		✓			✓		
Anguel et al. [8]	Prototype		✓					✓
ASIMOV [38]	System		✓					✓
AXSM [52]	Plug-in		✓			✓		
BAM [137]	Plug-in			✓	✓			
Batot et al. [12]	Framework	✓	✓		✓			
Baya [27]	Prototype			✓		✓		
Bobek et al. [17]	Prototype		✓				✓	
BPMoQualAssess [60]	Prototype		✓		✓			
B-repair [24]	System		✓			✓		
Cerqueira et al. [26]	Prototype		✓		✓			
Clarisó et al. [28]	Proposal	*n.a.*	✓		✓			
CONVErT [10]	Prototype			✓		✓		
Deng et al. [31]	Prototype		✓			✓		
DiaGen [82]	System		✓			✓		
DIG MDE [92]	System		✓		✓		✓	
DoMoRe [5, 6]	System		✓	✓		✓		
DPF [108]	Prototype		✓				✓	
DSL-maps [103]	Plug-in		✓			✓		
Elkamel et al. [37]	Prototype			✓		✓		
Extremo [123–125]	Plug-in	✓	✓			✓		
Heinemann [49]	Prototype		✓		✓			
Hermes [34–36]	Framework, Plug-in	✓	✓	✓	✓	✓	✓	✓
IntellEdit [93]	Framework		✓			✓		
IPSE [40]	Plug-in		✓			✓		
Kermeta [86]	Plug-in		✓				✓	
Kögel et al. [68, 69]	Prototype			✓	✓			
Koschmider et al. [50, 51, 71]	Prototype		✓			✓		
Li et al. [75]	Prototype		✓			✓		
MAGNET [16]	System		✓		✓			
Mani et al. [79]	Prototype		✓		✓			
Matikainen et al. [81]	Prototype			✓			✓	
MDSafeCer [87]	Plug-in		✓		✓			
ModBud [116]	Proposal	*n.a.*	✓	✓	✓			
Nassar et al. [91]	Plug-in		✓			✓	✓	
PARMOREL [11, 53]	Plug-in		✓				✓	
Paydar et al. [100, 101]	Prototype		✓			✓		
PME [99]	Plug-in			✓				✓
Rangiha et al. [110]	Prototype			✓		✓		
RapMOD [72, 73]	Plug-in			✓		✓		
REBUILDER [42]	System		✓		✓			
Refacola [130]	Prototype	✓	✓			✓		
Refactory [111]	Prototype		✓		✓			
ReVision [98]	System		✓			✓		
Savary-Leblanc [121]	Proposal	*n.a.*	*unknown*	*unknown*		✓		
SBPR [63, 64]	System		✓		✓			
Sen et al. [126, 127]	System		✓		✓			
SimVMA [132]	Proposal	*n.a.*	✓			✓		
SMART [47]	System		✓			✓		
UCcheck [9]	System		✓			✓		

however, the framework is not specific for particular editors, and is extensible with new heuristics to guide the search of recommendations. Refacola [130] achieves tool independence by being based on a constraint-based domain-specific language to specify model-assistance operations. Extremo [123–125] is a modelling assistant that defines extension points (the extensibility mechanism provided by Eclipse) to allow its integration with external modelling and meta-modelling tools within Eclipse. Finally, Hermes [34–36] is not a concrete RS but a framework with a plugin-based architecture to develop RSs within Eclipse. Its extension points allow defining new recommendation strategies and the integration with modelling editors and heterogeneous data repositories.
Other approaches can be used with several tools, but are still tool-dependent. This is the case of UCcheck [9], an assistant for use case diagrams coded in Python that supports use case diagrams specified with TTool – a free software from Telecom Paris – and the Cameo Systems Modeler.

Recommender trigger. As the fourth column of Table 5 shows, most RSs provide recommendations on user demand (41 approaches out of 51, an 80.39%). Fewer approaches provide recommendations proactively without user intervention (12 out of 51, a 23.53%), typically by monitoring the user editing actions to update the recommendations in return. Only a few tools (3 of them, a 5.88%) can trigger the recommendations both on demand and proactively: the recommender of domain model elements DoMoRe [5,6], the envisioned modelling learning environment ModBud [116], and the generic RS framework Hermes [34–36]. Finally, Savary-Leblanc [121] does not give enough details on how to access the recommendations, so we mark it as *unknown* in the table.

Recommendation enactment. The last four columns in Table 5 display how the works enact the recommendations. In most cases, recommendations can be applied either manually (31.37%) or interactively (49.02%). Automated enactments typically occur in model completion and model repair. As an example, DIG MDE [92] automatically completes a model, and if this is not possible, it recommends the user how to fix the model manually. In turn, the tool by Nassar et al. [91] permits repairing models either automatically or interactively. Three approaches (5.88%) provide semiautomated enactment of recommendations: two are co-evolution approaches [8, 38] that automatically infer and apply a migration strategy, but the user may need to select between alternative solution steps, e.g., in the case of non-resolvable changes; the other corresponds to the proactive modelling approach in PME [99], where models are automatically modified according to the models' meta-model, and the user is only prompted if several optional modifications exist. Finally, since Hermes [34–36] is a framework to build RSs, it provides mechanisms to support all types of recommendation enactment.

5.3 Recommendation

MDE researchers have applied diverse recommendation approaches for a variety of tasks and purposes. In this section, we characterise, categorise and analyse the works on MDE recommenders according to the features shown in the diagram of Figure 7.

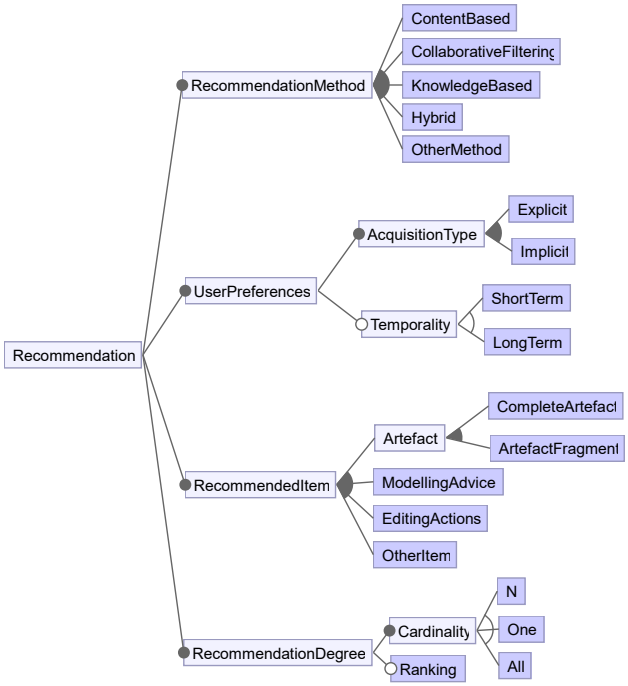

Fig. 7 Recommendation dimensions for RSs in MDE.

As a first feature of analysis, we consider the recommendation method used. The majority of the RSs apply one of the four main techniques explained in Section 2.2: *content-based*, *collaborative filtering*, *knowledge-based* and *hybrid*. In addition, some works use ad-hoc techniques that do not fall into the previous categories. They are represented by *OtherMethod* in Figure 7.

Second, RSs collect user information to provide personalised recommendations (feature *UserPreferences* in the diagram). In this respect, we investigate how this information is collected (feature *AcquisitionType*). In

some cases, the user's preferences are gathered *implicitly* by monitoring the user interactions with the system or analysing the current state of the modelling/MDE activity. In other cases, the user needs to *explicitly* provide his/her preferences to the system, for example via questionnaires. In addition, we examine the *temporality* of the collected preferences, which can reflect recent, likely temporal preferences for the task at hand (i.e., *ShortTerm*) or more general and enduring preferences (i.e., *LongTerm*).

Third, we analyse the types of items provided as recommendations (feature *RecommendedItem*). These can be *complete* artefacts (e.g., a model), *fragments* of an artefact (e.g., a class), *advices* that the user can profit from during a modelling activity, or *editing actions* (e.g., in the context of model repair). The diagram includes the *OtherItem* feature for items that do not fall in any of the previous categories.

Finally, the feature *RecommendationDegree* comprises the amount of recommendations presented to the user (*Cardinality*) and whether they are ranked (*Ranking*).

Table 6 categorises the surveyed approaches according to these features. Taking this categorisation into consideration, we start by analysing the approaches attending to the recommendation method they use, and then, we analyse them based on the other features.

Content-based. These approaches use different content encoding and similarity notions to represent and relate items for generating personalised recommendations.

First, we comment on the content-based approaches that recommend complete artefacts. Cerqueira et al. [26] compare two alternative encodings of sequence diagrams (*bag-of-words* and a vector encoding structural features) for the recommendation of sequence diagrams matching the user's preferences. The RS uses a content-based filtering algorithm to find the closest sequence diagrams. The RS proposed by Paydar et al. [100, 101] facilitates the reuse of models with functional requirements of web applications. For this purpose, the system recommends similar use cases to the one provided by the user, and then adapts the activity diagrams linked to the selected use case to the provided one. Item similarities are computed based on name similarity of the use case elements and on the diagram context. SimVMA [132] uses clone detectors to estimate similarities. It uses near-miss clones to recommend similar Simulink models, from which low-granularity recommendations can also be extracted.

Similarity has also been exploited to recommend artefact fragments. For example, Elkamel et al. [37] use similarity metrics to suggest similar classes to newly created classes. The developer may accept the suggested classes with all or some of their attributes and methods. DoMoRe [5, 6] addresses the same problem by means of semantic similarities. It relies on an extensive knowledge database – called *SemNet* – of several million domain-specific terms and their relationships to provide context-sensitive recommendations. In particular, DoMoRe suggests names for new elements, and possible related concepts to the selected one (e.g., upon selecting a class, the system suggests possible sub-/superclasses, container and aggregated classes, related and associated classes). Savary-Leblanc [121] envisions a RS that calculates the similarity using semantic distances obtained from lexical databases like WordNet [84]. Extremo [123–125] also employs semantic similarity based on WordNet to provide a ranked list of recommended model elements, upon an explicit query of the user.

Content-based similarity has been applied to transformation development as well. CONVErT [10] helps discovering and specifying transformation correspondences using concrete visualisations. A RS suggests mappings between source and target models based on different similarity heuristics, choosing mappings that resemble examples provided by the user. In a similar vein, AXSM [52] recommends mappings based on similarity criteria (source/target element tag names, element types, structural similarity, example data item equivalences) and previous user selections within the MaramaTorua tool. Refactory [111] supports the definition of generic refactorings over role models so that developers can reuse the refactorings on new languages by mapping the role model elements into elements of the language meta-model. The tool includes a RS to complete the mapping using structural similarity and other heuristics, like name similarity.

Collaborative filtering. These approaches exploit information about past behaviour or opinions from the user community [56]. In some cases, users correspond to developers for which personalised recommendations are generated, and in other cases, users (and items) are mapped to elements within the artefacts that are target of recommendations.

MAGNET [16], PARMOREL [11, 53] and ModBud [116] belong to the first case. MAGNET is a RS within the AutoFOCUS3 modelling tool to help beginners to learn using the tool. It monitors user actions and proposes short videos illustrating what to do next. The RS model is based on data collected during a tutorial with a previous set of users. PARMOREL uses reinforcement learning to find a sequence of actions that repairs the issues present in a model. The algorithm initially reuses the experience obtained from

Table 6 Recommender systems for MDE: Recommendation method.

Approaches	Acquisition Type	Temporality	Recommended Item	Recommendation Degree	
				Cardinality	Ranking
Content-based					
AXSM [52]	Implicit, Explicit	Short-term	Artefact fragment	All	✓
Cerqueira et al. [26]	Implicit, Explicit	Short-term	Complete artefact	N	✓
CONVErT [10]	Implicit, Explicit	Short-term	Artefact fragment	N	✓
DoMoRe [5, 6]	Implicit	Short-term	Artefact fragment	All	✓
Elkamel et al. [37]	Implicit	Short-term	Artefact fragment	All	
Extremo [123–125]	Explicit	Short-term	Artefact fragment	All	✓
Paydar et al. [100, 101]	Implicit	Short-term	Complete artefact	N	✓
Refactory [111]	Implicit	Short-term	Artefact fragment	N	
Savary-Leblanc [121]	Implicit	*unknown*	Artefact fragment	*unknown*	
SimVMA [132]	Implicit, Explicit	Short-term	Complete artefact, artefact fragment	All	✓
Collaborative filtering					
MAGNET [16]	Implicit	Short-term	Modelling advice	All	✓
Matikainen et al. [81]	Implicit	Short-term	Complete artefact	N	✓
ModBud [116]	Implicit, Explicit	*unknown*	Modelling advice	*unknown*	*unknown*
PARMOREL [11, 53]	Implicit, Explicit	Long-term	Editing actions	One	
Knowledge-based					
AMOR [21]	Implicit	Short-term	Editing actions	All	✓
Anguel et al. [8]	Implicit	Short-term	Editing actions	All	
ASIMOV [38]	Implicit	Short-term	Editing actions	N	
BAM [137]	Implicit	Short-term	Editing actions	All	✓
Baya [27]	Implicit	Short-term	Artefact fragment	N	✓
Bobek et al. [17]	Implicit	Short-term	Artefact fragment	N	✓
BPMoQualAssess [60]	Implicit	Short-term	Modelling advice	All	
Deng et al. [31]	Implicit	Short-term	Artefact fragment	N	✓
DiaGen [92]	Implicit	Short-term	Editing actions	All	
DIG MDE [92]	Implicit, Explicit	Short-term	Artefact fragment	All	
DPF [108]	Implicit	Short-term	Artefact fragment	All	
DSL-maps [103]	Implicit	Short-term	Artefact fragment	All	✓
IPSE [40]	Implicit	Short-term	Artefact fragment, modelling advice	One	
Kermeta [86]	Implicit	Short-term	Artefact fragment	All	
Li et al. [75]	Implicit	Short-term	Artefact fragment	N	✓
Mani et al. [79]	Implicit	Short-term	Editing actions	All	
MDSafeCer [87]	Implicit	Short-term	Modelling advice	All	
Nassar et al. [91]	Implicit	Short-term	Editing actions	All	
RapMOD [72, 73]	Implicit	Short-term	Artefact fragment	N	✓
REBUILDER [42]	Implicit	Short-term	Complete artefact, artefact fragment	All	✓
Refacola [130]	Implicit	Short-term	Editing actions	N	
ReVision [98]	Implicit	Short-term	Editing actions	All	✓
Sen et al. [126, 127]	Implicit, Explicit	Short-term	Artefact fragment	N	
UCcheck [9]	Implicit	Short-term	Modelling advice	All	
Hybrid: Content-based, collaborative filtering					
Heinemann [49]	Implicit	Short-term	Artefact fragment	N	✓
Kögel et al. [68, 69]	Implicit	Short-term	Editing actions	N	
Koschmider et al. [50, 51, 71]	Implicit, Explicit	Short-term	Complete artefact, artefact fragment	N	✓
Hybrid: Content-based, knowledge-based					
B-repair [24]	Implicit	Short-term	Editing actions	N	✓
Hybrid: Content-based, social-based					
Rangiha et al. [110]	Implicit, Explicit	Long-term	Artefact fragment	All	✓
SBPR [63, 64]	Implicit	Short-term	Complete artefact	All	✓
Other Method					
anATLyzer [117–119]	Implicit	Short-term	Editing actions	All	✓
Batot et al. [12]	Implicit	Short-term	Editing actions	N	✓
Clarisó et al. [28]	Implicit, Explicit	Short-term	Editing actions	N	✓
IntellEdit [93]	Implicit	Short-term	Editing actions	All	✓
PME [99]	Implicit	Short-term	Artefact fragment	All	✓
SMART [47]	Implicit	Short-term	Editing actions	All	
Any Method					
Hermes [34–36]	Implicit, Explicit	Long-term	Editing actions	N	✓

other users' repairs, and learns after each repair. ModBud is an envisioned framework to build modelling bots to assist novice users. The authors foresee using machine learning to predict good modelling decisions for given design requirements.
Matikainen et al. [81] address the second case. Their RS selects the best-performing state machine to control a robotic vacuum cleaner. Room layouts are interpreted as users, robot state machines as items, and item ratings are based on the performance of the robot state machines on the room layouts.

Knowledge-based. Most approaches belong to this category. They use techniques that can be generally classified as *constraint-based* or *case-based*. Constraint-based techniques determine the recommendations by looking for a set of items that fulfil established domain-dependent rules. Case-based techniques, in contrast, provide recommendations to a problem by examining past solutions for alike problems (cases) [56].
Some of the constraint-based recommenders found in the literature are built upon technologies such as Alloy and Prolog. Specifically, Sen et al. [126] use Prolog as a backend of the AToM^3 language workbench [29] to suggest completions of a partial model. The work was extended by using Alloy [127] to recommend the closest valid complete model within a given scope. Kermeta [86] also uses Alloy to provide completion suggestions. Refacola [130] provides a constraint-based language to express model-assistance operations in a declarative way. In the domain of education, IPSE [40] relies on Prolog to guide users in creating a class diagram. The guidelines are explicitly modelled by the teacher by means of constraints suggesting hints whenever matched. For the domain of embedded systems, DIG MDE [92] uses Prolog to guide the user in completing combinatorially challenging modelling problems on the basis of user-defined rules.
RSs for completion and repair are sometimes based on (graph transformation) rules. DPF [108] computes completion rules which ensure the satisfaction of well-formedness predicates. RapMOD [72] matches editing operations in UML structural diagrams to a catalogue of modelling activities, and ranks the candidate activities by relevance. Different from model completion, rule-based model repair may require deleting elements to obtain a valid model. Hence, Nassar et al. [91] derive graph transformation programs able to fix an invalid model by first deleting superfluous objects and links, and then adding necessary elements. DiaGen [82] uses hyper-graph grammar rules and hyper-graph patches (graph modifications) to propose both model completions and repairs. Similarly, ReVision [98] proposes model repairs based on consistency-preserving editing rules, with heuristics that avoid undoing former editing steps.
For quality assurance, BPMoQualAssess [60] recommends improvements for process models based on rules modelling expected quality criteria (e.g., regarding size, nesting levels, and element ratios), and UCcheck [9] provides advices for improving use case diagrams based on sets of rules and guidelines.
Recommenders for model/meta-model co-evolution can also be rule-based. This is the case of ASIMOV [38], where the language designer specifies the migration assistance rules, and modellers use them to obtain recommendations for model migration. In contrast, the approach by Anguel et al. [8] suggests a migration strategy based on meta-model matching and the use of logic programming.
Some rule-based RSs target behavioural models. MDSafeCer [87] recommends how to resolve flaws of safety argumentations attached to process models. For this purpose, it first identifies the problematic elements, and then uses rules to provide advices to resolve the deviations. Also for process modelling, BAM [137] uses model-checking to detect errors in process models, and suggests corrections for the errors in relation to user-defined validation rules and Dwyer's temporal specification patterns [33].
Some works use patterns following a case-based approach. In particular, Baya [27] relies on a knowledge base of curated patterns, several similarity metrics and ranking algorithms as a basis for the recommendation of the next steps when building mashup models. Moreover, it applies weaving to incorporate the recommended pattern into the mashup model. The process model recommenders of Li et al. [75] and Deng et al. [31] extract task relations and patterns from process models, which are then used to recommend activity nodes for the current model. The AMOR [21] model versioning system recommends resolution patterns for conflicts between two model versions. The patterns can be mined from repositories or specified manually. DSL-maps [103] uses a catalogue of patterns to transition from the requirements of a DSL (given as a mind-map) to its design (given as a meta-model). It performs a lexical analysis of the requirements to match them against an ontology-based description of the patterns, and suggests a ranked list of patterns to realize the requirements. Mani et al. [79] also use patterns to assist when repairing faults in input models of code generators. Their approach identifies correct output fragments that are similar to the incorrect one, and suggests repair actions based on run-time data.

Finally, probabilistic forms of knowledge representation are also possible. For example, REBUILDER [42] combines case-based reasoning with WordNet and Bayesian networks to enable reusing UML diagrams, or part of them. Bobek et al. [17] also use Bayesian networks to recommend following tasks when instantiating a configurable process model.

Hybrid. Some works combine several recommendation methods to benefit from their strengths and mitigate particular limitations. The surveyed papers have combined content-based techniques with collaborative filtering, social-based and knowledge-based methods.
Three approaches combine collaborative filtering with content-based recommendations. The first one, by Kögel et al. [68, 69], recommends model changes by looking at the previous model history (e.g., what other developers did on previous model versions) and co-occurring model changes. Heinemann [49] evaluates the use of association rules and collaborative filtering to recommend Simulink library elements for the current model. The collaborative filtering method considers models as users, and elements as items. Finally, the RS of Koschmider et al. [50,51,71] uses both similarity metrics and frequency of use by the community to recommend complete process models or fragments.
For behavioural modelling, B-repair [24] suggests automatic repairs of faulty models written in the B formal specification language. The approach uses two types of rules (hence being knowledge-based) to suggest fixes in state machine transitions. Then, it uses machine learning (features learnt from state machine transitions, a content-based approach) to estimate the quality of the repairs and rank the recommendations.
Finally, SBPR [63, 64] combines the traditional content-based approach with social-based recommendation to suggest business process models for reuse. For this purpose, it extracts information from the user profile in LinkedIn[2]. Similarly, the approach by Rangiha et al. [110] profits from social tagging to recommend suitable actors and roles in a social business process modelling tool. In addition, it recommends tasks based on similarity metrics.

Other method. A few works use non-traditional recommendation methods based on search and static analysis.
On the one hand, two approaches use model search as the underlying technique for recommendation, both targeting OCL. Clarisó et al. [28] generate potential fixes to OCL constraints by using mutation. Batot et al. [12] tackle the co-evolution of OCL constraints and meta-models using multi-objective optimization guided by criteria like correctness and minimization of changes and information loss.
On the other hand, several works provide recommendations out of the static analysis of models, metamodels or OCL expressions. PME [99], which extends the generic modelling environment (GME) to support proactive modelling, recommends further editing actions (e.g., connecting an object to another) upon user actions (e.g., selecting an object). The recommendations are created by the syntactic analysis of the meta-model and OCL constraints. IntellEdit [93] recommends quick fixes for repairing models based on the static analysis of failing OCL expressions. It ranks the recommended fixes by the amount of required changes (from lower to higher). AnATLyzer [117–119] extends the ATL IDE for developing model transformations with the detection of type errors and suggestions of quick fixes. Errors are detected by static analysis and model finders. The proposed quick fixes are ranked by the number of errors that they correct. The ranking can be calculated dynamically using speculative analysis [88] (i.e., the simulated execution of all possible repairs and the analysis of their consequences), or statically using rankings pre-computed on a set of transformations with injected faults. Finally, SMART [47] supports test-driven development of UML models. It statically analyses the test cases and their execution logs to report errors. Moreover, it suggests quick fixes for automatically solving structural errors (e.g., adding a missing model element), and provides guidance to solve behavioural errors triggered during the test case execution (e.g., displaying a sequence diagram with the test case execution, or a summary of the changes in attribute values or the model state).

Any method. The framework Hermes [34–36] for the creation of RSs can be extended with any recommendation strategy and recommendation method. It provides facilities to define the recommendation context, which can be obtained either implicitly or explicitly. Developers of RSs can persist user preferences (long-term temporality) and set filters and ranks for their strategies.

Once we have classified the works according to the recommendation method, we characterise how they collect the user's preferences (acquisition type), the temporality of those preferences, and the size and ordering of the recommendation sets (recommendation degree).

Acquisition type. All works but Extremo collect data implicitly. The most common type of implicit data is the user's previous interaction with the system (including the current selection of elements in the

[2] `https://www.linkedin.com/`

editor) and the in-progress model. In some cases, like SBPR, this includes user information from LinkedIn. In addition, 12 approaches [10, 11, 26, 28, 50–53, 71, 92, 110, 116, 123–127, 132] also collect data explicitly. In these cases, data is acquired through questionnaires, parameters, tags or requirement definitions, in combination with implicit data acquisition methods like analysing the user's in-progress model.

Temporality. Most approaches (90%) collect preferences for their use during a short period of time, typically the current modelling session or model state. PARMOREL [11, 53] uses long-term preferences by storing the experience gained from each repair, allowing the algorithm to improve its performance in consecutive executions. The process model recommender of Rangiha et al. [110] exploits persistent social tags to express, e.g., required skills for tasks and skill-sets of users. Hermes [34–36], being a framework, enables developers to persist user preferences as required by the recommendation strategy. Finally, two works [116,121] do not provide detailed temporality information.

Recommendation degree. When it comes to the recommended items, most approaches (53%) present *all* recommendations found by the method to the user. Since this might be overwhelming if there are many options, some approaches (4%) present just *one* recommendation, and others (39%) present the top N recommendations.

The filtering criteria vary depending on the recommendation method, and are frequently used to *rank* the suggestions. Examples of filtering criteria include the most similar models (as in [26]), the quick fixes repairing more errors (as in [119]), or the fixed model or constraint having the lowest number of modifications with respect to the original one (as in [28, 93]). Sometimes, the quality of the recommendation is calculated using pre-trained machine-learning models, as in [24]. In the case of anATLyzer [119], the user can choose either a fast but less accurate ranking of recommended quick fixes (based on a pre-calculated estimation of the repair power of quick fixes); or a slower but more accurate one (based on a speculative application of the quick fixes to the current transformation). As Table 6 shows, some early works do not provide information about cardinality or ranking.

5.4 Evaluation

In this section, we review how the RSs have been evaluated on the basis of the two orthogonal features shown in Figure 8: the followed evaluation *methods* and the used evaluation *metrics*.

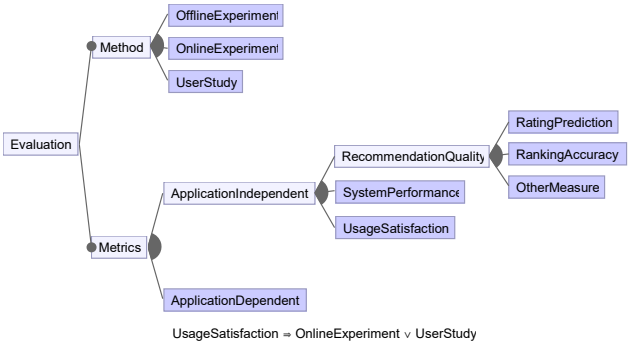

Fig. 8 Evaluation dimensions for RSs in MDE.

As the figure shows, we first distinguish three types of evaluation methods [45]: *offline experiments*, *online experiments*, and *user studies*. Offline experiments correspond to analytical studies on datasets, online experiments are user-centric studies that evaluate the system in a real setting, and user studies consist of experiments planned for small groups of participants.

In addition, we identify several metrics to assess different recommendation goals and quality measures. We classify the metrics as *application-independent* and *application-dependent*. In turn, application-independent metrics are divided into three groups. The first one comprises traditional metrics used to evaluate RSs regardless the application or task for which they have been developed. Here, we distinguish between *recommendation quality* (accuracy) metrics – i.e., *rating prediction* metrics (e.g., MAE and RMSE) and *ranking accuracy* metrics (e.g., precision, recall, nDCG and MRR) – and *other measures* that capture non-accuracy recommendation characteristics, such as diversity, coverage and novelty. A second group of domain-independent metrics is related to *system performance*, such as consumed time and required resources to perform a task. The third group of domain-independent metrics is formed by *usage satisfaction* metrics, such as user engagement, perceived usefulness and trust on the system, as well as system usability, responsiveness, security and privacy. As Figure 8 shows, measuring usage satisfaction requires performing on-line experiments or user studies, as offline experiments do not involve users.

Finally, application-dependent metrics are devised for particular MDE applications and tasks. They include metrics such as the average number of constraint violations in model repair recommendations, or the total number of valid matches in the recommendation of model transformation mappings.

Table 7 categorises the analysed RSs according to the method (*offline*, *online* and *user study*) and metrics used for their evaluation. An approach can appear mul-

Table 7 Recommender systems for MDE: Evaluation.

Approaches	Application Independent					Application Dependent
	Rating Prediction	Ranking Accuracy	Other Measure	System Performance	Usage Satisfaction	
Offline experiment						
AMOR [21]						✓
anATLyzer [119]						✓
Batot et al. [12]						✓
Baya [27]		✓		✓		
B-repair [24]		✓				
CONVErT [10]		✓				
Deng et al. [31]		✓		✓		
DiaGen [82]				✓		
DIG MDE [92]				✓		
Extremo [123–125]				✓		✓
Heinemann [49]		✓				
IntellEdit [93]		✓				✓
Kögel et al. [68, 69]		✓				
Li et al. [75]		✓		✓		
Mani et al. [79]						✓
Matikainen et al. [81]						✓
PARMOREL [11, 53]				✓		
PME [99]				✓		✓
Refacola [130]		✓		✓		✓
Refactory [111]						✓
SBPR [64]		✓				
Online experiment						
ASIMOV [38]				✓		✓
DoMoRe [5, 6]					✓	
User study						
anATLyzer [119]						✓
AXSM [52]					✓	
Baya [27]				✓	✓	
Cerqueira et al. [26]		✓			✓	
CONVErT [10]					✓	
DSL-maps [103]					✓	
Elkamel et al. [37]		✓				
IPSE [40]					✓	
Koschmider et al. [50, 51, 71]				✓	✓	✓
MAGNET [16]					✓	
Paydar et al. [100, 101]		✓				
RapMOD [72, 73]		✓		✓		✓
No evaluation						
[8,9,17,28,34–36,42,47,60,86,87,91,98,108,110,116,121, 126, 127, 132, 137]						

tiple times in the table if it was evaluated by means of several methods. Additionally, Table 8 presents a matrix crossing the methods and metrics used in the papers. Overall, we can observe that online experiments are the least used evaluation method, and that neither rating prediction nor non-accuracy metrics are used; the former are indeed in disuse in the RS research field. A total of 19 approaches (37%) have no evaluation.

Offline experiment. This is the most popular evaluation method, used in 21 of the revised approaches. Making use of data records with past user behaviour and feedback, among other information, offline experiments simulate past and present real conditions without requiring the participation of users during the evaluation process.

These experiments exploit available datasets to compute a variety of aspects about a RS, such as its scalability and performance, the precision and quality of its recommendations, and the reduction of modelling effort. However, since data repositories of models are not as common as, e.g., those for programming languages, a fundamental issue about offline experimentation for MDE recommenders is the availability of artefacts over which the evaluation can be performed.

Table 8 Recommender systems for MDE: Evaluation vs metrics.

Metrics \ Evaluation	Offline Experiment	Online Experiment	User Study
Rating Prediction			
Ranking Accuracy	Baya [27] B-repair [24] CONVErT [10] Deng et al. [31] Heinemann [49] IntellEdit [93] Kögel et al. [68,69] Li et al. [75] Refacola [130] SBPR [64]		Cerqueira et al. [26] Elkamel et al. [37] Paydar et al. [100,101] RapMOD [72,73]
Other Measure			
System Performance	Baya [27] Deng et al. [31] DiaGen [82] DIG MDE [92] Extremo [123–125] Li et al. [75] PARMOREL [11,53] PME [99] Refacola [130]	ASIMOV [38]	Baya [27] Koschmider et al. [50,51,71] RapMOD [72]
Usage Satisfaction		DoMoRe [5,6]	AXSM [52] Baya [27] Cerqueira et al. [26] CONVErT [10] DSL-maps [103] IPSE [40] Koschmider et al. [50,51,71] MAGNET [16]
Application-Dependent	AMOR [21] anATLyzer [119] Batot et al. [12] Extremo [123–125] IntellEdit [93] Mani et al. [79] Matikainen et al. [81] PME [99] Refacola [130] Refactory [111]	ASIMOV [38]	anATLyzer [119] Koschmider et al. [50,51,71] RapMOD [72,73]

To address this issue, we have observed four solutions in the literature. The first one is the generation of synthetic data. For the case of repair recommenders, the set of artefacts is typically generated by applying mutation operators over a set of seed artefacts to obtain faulty artefact variants. This approach was used by anATLyzer to evaluate the recommendation of quick fixes over transformations [117–119]; by B-repair to evaluate fixes over state machines [24]; by IntellEdit to evaluate if its content-assistant solves errors in models [93]; by Matikainen et al. to evaluate the recommendation of state machines for robotic cleaners [81]; and by Mani et al. to evaluate the effectiveness of its model repair recommender [79]. The seed artefacts may come from third parties (as in the case of anATLyzer and Mani et al.), be generated synthetically (as in IntellEdit and Matikainen et al.) or manually (as in B-repair). A second solution is to locate repository sources of the appropriate type. For example, CONVErT was evaluated through models from the Illinois Semantic Integration Archive [10], Extremo gathered heterogeneous information from several sources such as OMG meta-models or the AtlanMod meta-model Zoo [123], Refacola used the whole AtlanMod Ecore meta-model Zoo [130], Heinemann used models of a Simulink repository [49], Matikainen et al. used floor plans from the Google SketchUp database of 3D models [81], and the evaluation of SBPR involved process models from different sources [64]. A third solution consists of taking example artefacts from published papers (as in B-repair), or datasets used by other authors (as in the case of Kögel et al. in 2016 [68,69]).

Finally, another solution is to obtain real-world artefacts from companies, like Li et al. [75] and Deng et al. [31], who used a dataset of 221 business processes collected from a local government in China, in combination with a synthetic dataset. Table 9 shows the public (i.e., available) datasets and repositories used in the surveyed papers.

In addition to mutating artefacts to introduce faults, we have found other modifications in artefacts. In RSs for model completion, the models of the considered dataset are removed elements to enable triggering the recommendations, and their effects are compared with the original model. This is the strategy followed by Heinemann [49], Li et al. [75], Deng et al. [31] and Baya [27]. In the first case, half of the model elements were removed; in the second case, the recommendation starts from the second activity node; and in the two last cases, model portions of increasing size were systematically removed.

Some approaches require training the recommender. For this purpose, the dataset is partitioned into sets for training and validation, as done by Li et al. [75], Deng et al. [31] and Heinemann [49]. To estimate the generalizability of the method and avoid problems related to overfitting and selection bias, k-fold cross-validation is recommended for statistical analysis [25]. This way, Deng et al. [31] use 5-fold cross-validation: the dataset is partitioned into five subsets, one is taken for validation (testing), the rest for training, and the procedure is repeated 5 times with each subset. Similarly, Heinemann [49] used 10-fold cross-validation.

Regardless the use of datasets, some systems are empirically compared against baselines, which can be naive methods as done by Heinemann [49], who used a RS that suggests the most popular Simulink blocks in libraries. A few cases use existing recommenders built by other researchers, like Li et al. [75] and Deng et al. [31], who compare their approach against two other recommenders for process models. In other cases, the system is evaluated with and without its recommendation component enabled [99,111]. Finally, some approaches are evaluated analytically, like Extremo [124,125], whose extensibility is assessed via integration with several third-party tools and formats, or PME [99], where the authors built an analytical model to estimate the modelling effort.

Online experiment. Only two of the revised approaches were evaluated using online experiments, both in the context of external projects. ASIMOV [38] was evaluated using a real commercial scenario named Alps Furniture. Two groups of users were asked to co-evolve models either using ASIMOV or manually, and the results were analysed to assess the effort and time reduction achieved when using the tool. The domain modelling recommender DoMoRe [5, 6] was used in various industrial and research environments, and the user feedback and experience allowed identifying potential aspects for improvement.

User study. There are 12 approaches evaluated with user studies. These typically involve a small group of users that perform some tasks, making it possible to analyse the effectiveness of the users on completing the tasks with and without the recommender, as well as to gather information about user experience via questionnaires [112].

We have identified 3 types of user studies, in which: i) users perform tasks using the proposed recommender; ii) users utilise the recommender in an A/B testing setting (i.e., some users perform tasks with the recommender, and some others without it); and iii) the recommendations are compared to the decisions an expert user would make (i.e., the expert user plays the role of oracle function).

The first type of user studies was applied to AXSM [52] to evaluate usage satisfaction; to the RS proposed by Cerqueira et al. [26] to evaluate the usage satisfaction and the accuracy of its sequence diagram recommendations; to CONVErT [10] to get user feedback on the usefulness and usability of the tool to develop transformations aided by interactive recommendations; to DSL-maps [103] to assess the perceived usability and usefulness of its pattern assistant to build meta-models; to IPSE [40] to measure usage satisfaction about its support to help learning UML skills; to MAGNET [16] to get user feedback on the usefulness of the recommendations to learn using AutoFOCUS3; and to RapMOD [72, 73] to measure the quality of its graphical model auto-completion recommendations and the reduction of modelling effort.

The second study type was used by Baya [27] to evaluate (in a crowdsourced user study) whether recommending and weaving mashup model patterns reduces the development time, the number of user interactions, and the time between user interactions. In addition, the participants filled-in a questionnaire to evaluate their satisfaction with the tool. Also in this category, Elkamel et al. [37] evaluate the relevance and accuracy of the recommended elements for UML diagrams, and Koschmider et al. [50, 51, 71] asked two sets of students to create process models with and without recommenders. In the latter case, the authors measured the time spent, the quality of the results and the usage satisfaction.

Finally, two approaches compare their recommendations with the a-priori choices of expert users. The authors of anATLyzer [119] evaluated the usefulness

Table 9 Public datasets used in the evaluations.

Dataset	URL	Paper
Alps Furniture meta-model	https://backus1.uniandes.edu.co/~enar/dokuwiki/doku.php?id=asimovevaluation	[38]
anATLyzer quick fix website (ATL transform.)	http://sanchezcuadrado.es/projects/anatlyzer/quickfixes.html	[119]
ATL transformations Zoo	https://www.eclipse.org/atl/atlTransformations/	[118]
AtlanMod Ecore Meta-model Zoo	https://web.imt-atlantique.fr/x-info/atlanmod/index.php?title=Ecore	[130]
Extremo website (meta-models)	https://github.com/angel539/extremo/wiki/Performance-Evaluation	[123]
Extremo website (model instances)	https://github.com/angel539/extremo/wiki/Case-Studies	[123]
Google SketchUp (3D models)	https://www.sketchup.com/products/3d-warehouse	[81]
Illinois Semantic Integration Archive	http://pages.cs.wisc.edu/~anhai/wisc-si-archive/	[10]
Matlab Central File Exchange (Simulink files)	https://www.mathworks.com/matlabcentral/fileexchange/	[49]
Model versioning benchmarks	http://www.modelversioning.org/index3899.html?option=com_content&view=article&id=54&Itemid=68	[21]
PARMOREL github (models)	https://github.com/MagMar94/ParmorelRunnable	[53]
PARMOREL website (models)	https://ict.hvl.no/project-parmorel/	[11]
ProB Public Examples Repository	https://www3.hhu.de/stups/downloads/prob/source/	[24]
Refactory website (generic model refactorings)	http://www.modelrefactoring.org	[111]
State machine execution contract	http://ecariou.perso.univ-pau.fr/contracts/exec-contract.html	[12]
State machine model and OCL queries	https://github.com/atlanmod/LazyOcl_StateMachineExample	[12]
UML-based Web Engineering (UWE) website	https://uwe.pst.ifi.lmu.de/examples.html	[100, 101]
Yahoo! Pipes	http://www.pipes.digital/pipes	[27]

of its quick fixes and the utility of its ranking with respect to the free choices made by two independent developers. Paydar et al. [100,101] used the opinion of experts as the golden standard to evaluate the accuracy of their algorithms to detect behaviour/concepts in use cases, annotate activity diagrams with entities from class diagrams, and recommend use cases based on similarity metrics.

Application-independent metrics. The most used ranking accuracy metrics are precision, recall and F-measure [10,26,27,31,49,64,68,69,72,73,93,100,101]. Some papers consider additional metrics to evaluate the accuracy of the recommendations, such as mean reciprocal rank (MRR) [100, 101]; 11-point interpolated average precision [100,101]; the average number of recommended alternative solutions per successful recommendation [130]; the hit rate, which is the fraction of correct recommendations in the recommendation list [31]; or relevance and accuracy rates [37].
Several authors measure the performance of their approaches, being time metrics the most common, in particular, the time to compute recommendations [11, 27,53,63,73,82,99,123,130], and the time spent by the user to perform a task [27, 38, 72].
Finally, usage satisfaction metrics include mostly feedback from the users after using the system. The feedback is collected informally [5,6,52], by means of questionnaires [10, 16, 27, 40, 103] or asking the users to rank the provided recommendations using a Likert scale [26].

Application-dependent metrics. These are metrics specific to MDE activities, such as the number of model editing operations [38, 72, 79, 99], the edit distance between conflict pairs [21], the average number of properties changed per applied quick fix [130], the number of attempts to co-evolve a model [38], the lines of code needed to integrate a meta-modelling tool with the RS [124, 125], the number of meta-model constraints fixed in a co-evolution scenario [12], the amount of constraint violations [93], the coverage of a room layout model [81], or the number of valid meta-model/role model matches [111].
Additionally, some metrics are related to the completeness or correctness of the recommendation approach, such as how complete a set of quick fixes is [79, 119], the validity of quick fixes or co-evolution actions (they completely remove an error) [38,79,119,130], or the impact of quick fixes (number of problems removed or introduced by their application) [93,119].

6 Discussion

This section discusses the results of our systematic mapping in relation to the three RQs posed in the introduction. Section 6.1 answers RQ1 (*"In which ways can recommender systems assist in the different tasks within MDE processes?"*), Section 6.2 answers RQ2 (*"Which recommendation techniques are most commonly used to support MDE tasks, and how are recommenders for MDE evaluated?"*), and Section 6.3 answers RQ3 (*"What are the main opportunities in recommender sys-*

tems for MDE solutions?"). Finally, Section 6.4 discusses the threats to the validity of our study.

6.1 RQ1: In which ways can recommender systems assist in the different tasks within MDE processes?

As discussed in Section 5.1, existing RSs for MDE target five main purposes: complete, create, find, repair and reuse. These tasks can be performed over models, meta-models, transformations or code generators.

The graphic in Figure 9 shows the number of approaches per purpose, stratified by the artefact type. It can be observed that the majority of approaches focus on completion and repair (together, 73.4% of the approaches), followed by reuse (10.9%), find (7.8%), other purposes (4.7%) and create (3.2%).

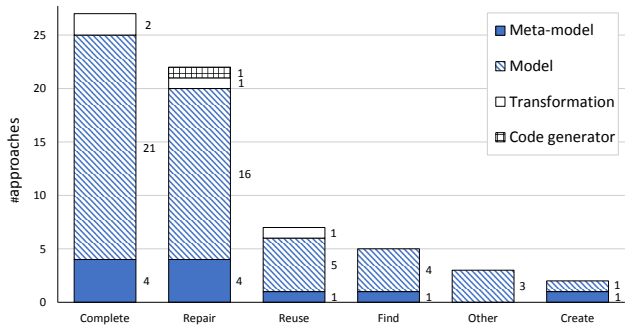

Fig. 9 Distribution of approaches by recommendation purpose.

As Figure 10 shows, most recommenders work over *models* (76.5%), followed by *meta-models* (15.6%), *transformations* (6.2%) and *code generators* (1.6%).

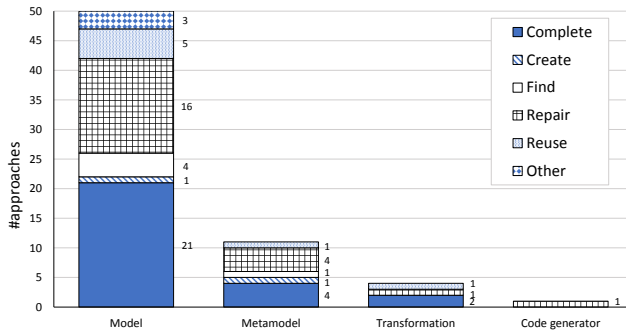

Fig. 10 Distribution of approaches by type of artefact.

Recommenders with the purpose of completing artefacts typically help in the development of models. For this purpose, some approaches transform partial models into a constraint satisfaction problem or logic programming to obtain a syntactically correct model conformant to its meta-model and integrity constraints [40, 82, 86, 92, 126, 127, 130]. This may involve adding many elements to the partial model. Instead, other approaches provide finer-grained recommendations for a step-by-step construction of a model. These recommendations are based on similar existing models [34–37, 132], model libraries [49], model histories [68, 69], knowledge bases [5, 6], or a static analysis of the language meta-model [99]. Since meta-models are also models, some approaches can be applied on both of them. Recommenders to complete transformations suggest mappings between source and target elements [10, 52].

Recommenders in repair approaches mainly consider models as well. These recommendations assist in repairing inconsistent models using a variety of techniques, such as rules [38, 91, 108], guidelines [60] or reinforcement learning [11, 53]. Sometimes, model repair occurs on specific contexts, like meta-model/model coevolution [8, 38] or conflict resolution in model versioning [21]. There is less support to repair meta-models and OCL constraints [12, 28], transformations [117–119], and models within code-generation activities [79].

In our study, we have identified numerous language-independent approaches [11, 34–36, 53, 68, 69, 82, 86, 91, 93, 98, 99, 108, 123–127, 130], but most RSs are specific for a modelling language. Figure 11 shows the targeted languages for the language-dependent cases. Most are widely used languages, like UML or process modelling notations, and there are RSs for both structural models (e.g., class diagrams) and behavioural ones (e.g., process models, sequence diagrams, and state machines).

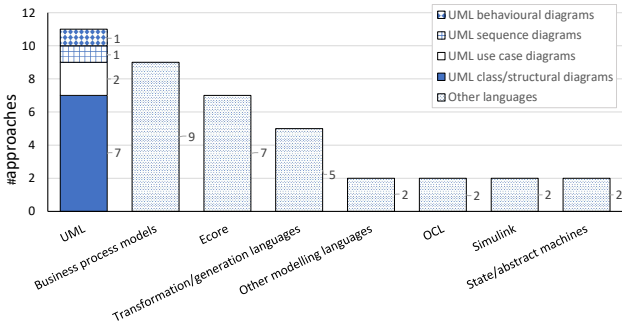

Fig. 11 Distribution of approaches by targeted language.

It is worth mentioning that there is tool support for 99.2% of the approaches, though some of them (37.2%) are prototypes. This demonstrates the feasibility of developing RSs for MDE tasks, but more effort may be needed to increase the number of mature, fully-developed tools. As Section 5.2 mentions, most recommenders help in the modelling activity on user demand,

but proactive approaches that monitor the user activity to update the recommendations are not uncommon.

6.2 RQ2: Which recommendation techniques are most commonly used to support MDE tasks, and how are recommenders for MDE evaluated?

Figure 12 shows the recommendation methods used by the studied approaches. Most RSs for MDE are knowledge-based (47%), followed by content-based (19.6%), hybrid (11.8%) and based on collaborative filtering (7.8%). Among the hybrid approaches, two are content-based and social-based, one is content-based and knowledge-based, and three combine content-based techniques with collaborative filtering. The bar *Other Methods* refers to ad-hoc methods different from the previous classical recommendation algorithms. Interestingly, there are more RSs applying ad-hoc methods (11.8%) than collaborative filtering (7.8%). Only one approach is extensible on the recommendation methods and therefore it may potentially support any of them [34–36].

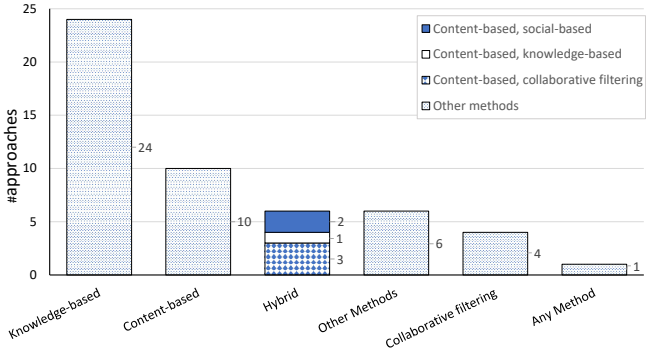

Fig. 12 Distribution of approaches by recommendation method.

Most of the information to build personalised recommendations is collected implicitly. Only 13 works consider explicit preferences of users, and all but one of those cases use implicit information as well.

An interesting question concerns the relation between recommendation methods and modelling purposes. Table 10 classifies the approaches along these two dimensions (cf. Tables 3 and 6). We can see that content-based methods have been used mostly to complete artefacts, but also to find and reuse them; collaborative filtering has been applied to find, repair and other purposes; knowledge-based RSs have been extensively used to complete and repair artefacts, as well as for every recommendation purpose in our classification except finding; and other ad-hoc methods have targeted complete and repair.

Table 10 Number of approaches grouped by recommendation purpose and method.

	Content-based	Collab. filtering	Knowledge-based	Hybrid	Other	Any
Complete	6	0	10	4	2	1
Create	0	0	2	0	0	0
Find	2	1	0	1	0	0
Repair	0	1	12	1	4	0
Reuse	3	0	1	1	0	1
Other	0	2	1	0	0	0

If we look at the recommendation purpose, we observe that recommenders for completion have used all considered recommendation methods (especially knowledge- and content-based) but collaborative filtering. Creation tasks have only been approached using knowledge-based methods. Recommenders for finding artefacts use collaborative filtering, content-based or hybrid algorithms, but not knowledge-based. Repair has been resolved mostly using knowledge-based methods, but also using collaborative filtering, ad-hoc and hybrid (content-based plus knowledge-based) methods. Finally, reuse has been tackled by content-based, knowledge-based and hybrid (content-based plus collaborative-filtering) methods.

Regarding evaluation, only 32 out of the 51 approaches (62.7%) have been evaluated. Offline experiments are the most frequent kind of evaluation [10–12, 21,24,27,31,49,53,64,68,69,75,79,81,82,92,93,99,111, 119, 123–125, 130]. This may be due to the difficulty to find a relevant number of users with the required level of expertise in modelling and willing to participate in online experiments or user studies. Moreover, some recommenders are implemented for very specific tools developed within research labs, sometimes prototypically, and therefore the tools are neither mainstream nor have a vast number of users. Therefore, while some approaches have been evaluated by means of user studies [10, 16, 26, 27, 37, 40, 50–52, 71–73, 100, 101, 103, 119], they involve small groups of participants (ranging from 2 to 44), typically students [10,26,37,40,73,100,101], developers/modellers [52,103,119] or more rarely employees [16,27]. Online experiments are very scarce [5,6,38].

Offline experiments require data, which sometimes come from public repositories [10,49,64,81,123,130] or companies [31,75]. However, in contrast to the programming field, it is difficult to have access to modelling artefacts, especially from industrial projects. For this reason, many authors resort to synthetic datasets created, e.g., via mutation or systematic generation [24, 79, 81, 93, 117–119]. In other cases, the authors evaluate their

proposal using artefacts from other papers [24, 68, 69] or analytically via case studies.

Some of the revised RSs have been evaluated using domain-independent metrics applicable to general RSs, specifically ranking accuracy metrics (mainly precision, recall and F-measure), time metrics and usage satisfaction collected via questionnaires. The advantage is that these metrics are standard and well accepted. As an example, to measure usability (a dimension of usage satisfaction), one could use de-facto standard questionnaires like the System Usability Scale (SUS) [20]. Instead, or in addition, some evaluations have considered metrics specific to MDE tasks – like the number of fixed/violated OCL constraints in a model – or domain-specific notions of completeness or correctness. These metrics are defined ad-hoc for each case.

Finally, we discuss whether some kinds of recommendation tasks are evaluated more than others. We have found that all RSs with the purpose of finding artefacts or fragments have been evaluated (100%), followed by completion tasks (73.9% of the RSs helping in completion tasks have been evaluated), repair tasks (61.1%), and create (50%) and reuse tasks (50%). RSs targeting repair have been mostly evaluated offline, while recommenders for other purposes have been evaluated using a wider variety of methods.

6.3 RQ3: What are the main opportunities in recommender systems for MDE solutions?

This section analyses gaps in the current research, resulting from an analysis of the coverage of the feature model by the proposals. Then, we identify opportunities based on an analysis of the different dimensions of the classification we propose, using both insights from the reviewed papers and our own experience.

Our analysis of the state-of-the-art reveals some gaps in the targeted tasks and artefacts. Most approaches focus on models, a handful on meta-models, very few on transformations, and hardly any on code generators. However, given that MDE fosters the automated processing of models, RSs for transformations and code generators (e.g., recommending completions of the code generation template; suggesting template fragments; or helping to repair faulty generators) would be very useful for the community. Similarly, the purpose of most RSs is completing and repairing models; however, there are few recommendation approaches for finding relevant artefacts, reusing them in a given context, and creating artefacts from scratch. For the latter case, we envision RSs proposing initial artefact templates out of higher-level descriptions, maybe defined using natural language. Finally, RSs for structural diagrams are more numerous than those for behavioural diagrams. Developing further RSs for behavioural diagrams would reveal whether behaviour and structure may require different recommendation methods, whether similarity-based recommendation is enough for behavioural diagrams, or whether behavioural diagrams would benefit from semantic comparison (e.g., based on execution) to generate recommendations.

Many of the studied papers present RSs for a specific language or tool (cf. Tables 4 and 5). Such recommenders tackle a single problem and are "hard-wired" into the systems they were designed for. Hence, an open line of research is devising solutions that allow adapting the recommendation algorithms, the users' preferences or the evaluation metrics to the users' needs. In this respect, a reference architecture for intelligent modelling assistance was proposed in [90], and one step in this direction is Hermes [34–36], since this framework permits integrating RSs into tools as well extending the framework with new recommendation methods.

In contrast to the field of programming, one of the main barriers when building RSs for MDE activities is the lack of data that can be used for training the recommenders. There are several initiatives to create repositories of modelling artefacts, both in the MDE [39, 76, 114] (some listed in Table 9) and BPMN communities [43]. Moreover, dedicated model search engines have been recently proposed [76], which can be used to create datasets of modelling artefacts. However, more efforts to make artefacts public and accessible are required.

Related to the previous point, we are recently witnessing the proposal of low-code development platforms for specific domains, like the creation of data analysis workflows (e.g., RapidMiner[3]), chatbots (e.g., Dialogflow[4]) or event-driven applications (e.g., Node-Red[5]). These platforms are cloud-based, making it easier for users without a technical background to construct applications by means of graphical languages and forms. Low-code platforms free the user from installing the development tool and deploying the defined applications, since they are used in a web browser. Frequently, low-code platforms form ecosystems where the models created by all users are stored in the platform's repository. This availability of data and users makes low-code platforms the ideal scenario for creating recommender systems, as shown in [55].

As we discussed in Section 6.2, a large percentage of the RSs in MDE are knowledge-based or content-based. This differs from the predominance of collabora-

[3] `https://rapidminer.com/`
[4] `https://cloud.google.com/dialogflow`
[5] `https://nodered.org/`

tive filtering and hybrid approaches in the RSs research field [15], where e-commerce (e.g., Amazon, Zalando), leisure (e.g., Netflix, Spotify), tourism (e.g., Booking, Yelp) and social networks (e.g., Facebook, Twitter) are the most widely addressed domains. It is in these domains where large communities of users provide feedback – mainly in the form of numeric ratings and textual reviews – which is exploited to find user similarities valuable for generating effective personalised recommendations. Following this trend, there are plenty of opportunities to research on how to exploit further collaborative filtering in MDE, for instance, via model and code sharing platforms. Moreover, the few revised works that apply collaborative filtering to MDE neglect long-term users' preferences, hence this stands as a problem worth investigating as well.

Regarding the evaluation of recommendations, Section 6.2 showed that only two approaches [5, 6, 38] report online experiments in real settings, and only one of them [38] uses A/B testing, as commonly done in the online evaluation of information retrieval and filtering approaches. This evaluation methodology not only allows assessing the performance of certain recommendation functionality with users at a large scale, but also its real effectiveness in a non-controlled scenario where contextual conditions arise. In contrast, most approaches were evaluated via offline experiments. In these cases, using public datasets and following standard evaluation methodologies are essential to ensure reproducibility and ease advances in the field. Except for a few cases [27,119,123], we have observed a general lack of reproducibility of the reported experiments. In line with the current open science movement [83], we believe that disclosing replication packages (containing the raw data and all necessary scripts for their analysis) is the way forward in this area. By making datasets available, the creation of new RSs as well as their comparison and improvement are facilitated.

In addition to generic recommendation accuracy and system performance metrics (cf. Table 7), we envision the formal definition and generalization of metrics oriented to particular MDE tasks (i.e., complete, create, find, repair and reuse) as a relevant research challenge. As our study reveals, the literature already presents ad-hoc metrics, such as the number of model editing operations to evaluate model completion [72, 99] or the number of constraint violations to assess the correctness of model repair [93]. However, there is room for designing and reporting more general, well established task-specific measures that would allow comparing distinct recommendation methods.

Related to user experience, an important success factor of RSs is how they integrate within the MDE tool [1, 89, 90]. For notations in the business process modelling domain, some studies investigate how to present recommendations [66], and surveys on the preferred ways to display recommendations in graphical modelling have been conducted as well [36]. However, more usability studies are required to understand the most effective, user-friendly ways to present recommendations for different styles of modelling languages and tasks.

With respect to other research trends on RSs, we highlight recommendation explainability [133] and group-oriented recommendation [80] as two directions of potential interest which, according to our review, have not been addressed yet in the MDE area, but are being extensively investigated by the RS community. On the one hand, explaining to the user the reasons for which recommendations are presented, as well as the potential benefits of the recommendations for the task at hand, can increase the user engagement and trust on the system, among other aspects [133]. On the other hand, there are cooperative tasks and environments that provide recommendations to a group of people, and consequently have to take individual preferences and constraints into account. In this context, the chosen methods for aggregating user models and generating consensus recommendations have to be complemented with an appropriate (collaborative) evaluation [80].

Lastly, although it is out of the scope of this study, we want to mention an open research issue related to the development process of RSs. We have observed that most RSs have been developed by hand from scratch, and very few works have investigated the application of MDE to assist in the design, implementation and evaluation of a RS for a given problem. The development of a RS and its integration into a tool undoubtedly requires a high effort, as noted in [90]. This makes the construction of RSs for DSLs – which typically have a smaller user community than languages like UML – less cost-effective. Therefore, methods for automating the construction of RSs for modelling languages, like those proposed in [7, 129], could be very useful for the MDE community.

6.4 Threats to validity

Some factors may threat the validity of our study. First, we might have missed some papers due to the query we have used. To mitigate this threat, we tested several versions of the query, confirming that papers we knew were relevant appeared in the query results.

A related threat is that some relevant papers might not be indexed in the databases considered for our

query. To mitigate this risk, we performed a final process of snowballing [138] to consider further relevant papers not included in the query results.

In the screening process, we might have erroneously left some relevant papers out. To mitigate this risk, each paper was independently checked by the four authors of the study, and was added to the second screening phase if one of them considered it relevant. In this second phase, it was read in full detail.

Finally, there is a thin line dividing the systems that can be considered to provide recommendations, with respect to others that just offer some kind of automated analysis. This situation is exacerbated by the fact that some systems rely on highly specialized algorithms which are non-standard in the RS literature. In our review, we included those systems that provide recommendations and assistance to the user to choose a small set of items over a large set of possibilities, or which consider implicit or explicit users' preferences. When in doubt, we included the system in the study.

7 Summary

In this paper, we have presented a systematic mapping review of existing research works on RSs for MDE. We have classified those works along four main dimensions (domain, tooling, recommendation and evaluation) characterised by means of feature models.

The review has allowed answering three research questions. First, we have seen that current RSs mainly target model completion and repair. Second, the most used recommendation methods in MDE are knowledge-based and content-based. Finally, we have identified research gaps and opportunities in the area, like implementing RSs to help in developing transformations and code generators, finding and reusing artefacts, and creating artefacts from scratch. We encourage the community to pick these challenges to improve the current MDE practice and tooling.

Acknowledgements We thank the reviewers for their useful comments. This work has been funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement n° 813884 (Lowcomote [134]), by the Spanish Ministry of Science (projects MASSIVE, RTI2018-095255-B-I00, and FIT, PID2019-108965GB-I00), and by the R&D programme of Madrid (project FORTE, P2018/TCS-4314).

References

1. S. Abrahão, F. Bourdeleau, B. H. C. Cheng, S. Kokaly, R. F. Paige, H. Störrle, and J. Whittle. User experience for model-driven engineering: Challenges and future directions. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS*, pages 229–236. IEEE Computer Society, 2017.
2. Acceleo. `https://www.eclipse.org/acceleo/`, 2020.
3. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
4. G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*, pages 217–253. Springer, 2011.
5. H. Agt-Rickauer, R. Kutsche, and H. Sack. Automated recommendation of related model elements for domain models. In *6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Revised Selected Papers*, volume 991 of *CCIS*, pages 134–158. Springer, 2018.
6. H. Agt-Rickauer, R. Kutsche, and H. Sack. DoMoRe - A recommender system for domain modeling. In *6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 71–82. SciTePress, 2018.
7. L. Almonte, I. Cantador, E. Guerra, and J. de Lara. Towards automating the construction of recommender systems for low-code development platforms. In *1st LowCode Workshop (LowCode@MoDELS)*, pages 66:1–66:10. ACM, 2020.
8. F. Anguel, A. Amirat, and N. Bounour. Hybrid approach for metamodel and model co-evolution. In *5th IFIP TC 5 International Conference on Computer Science and its Applications (CIIA)*, pages 563–573. Springer, 2015.
9. E. R. Aquino, P. de Saqui-Sannes, and R. A. Vingerhoeds. A methodological assistant for use case diagrams. In *8th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 227–236. SciTePress, 2020.
10. I. Avazpour, J. Grundy, and L. Grunske. Specifying model transformations by direct manipulation using concrete visual notations and interactive recommendations. *Journal of Visual Languages and Computing*, 28:195–211, 2015.
11. A. Barriga, A. Rutle, and R. Heldal. Improving model repair through experience sharing. *Journal of Object Technology*, 19(2):13:1–21, 2020.
12. E. Batot, W. Kessentini, H. A. Sahraoui, and M. Famelis. Heuristic-based recommendation for metamodel - OCL coevolution. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 210–220. IEEE Computer Society, 2017.
13. B. Baudry, S. Ghosh, F. Fleurey, R. B. France, Y. L. Traon, and J. Mottu. Barriers to systematic model transformation testing. *Commun. ACM*, 53(6):139–143, 2010.
14. A. Bellogín, I. Cantador, and P. Castells. A comparative study of heterogeneous item recommendations in social systems. *Information Sciences*, 221:142–169, 2013.
15. S. Berkovsky, I. Cantador, and D. Tikk. *Collaborative Recommendations: Algorithms, Practical Challenges And Applications*. World Scientific, 2018.
16. S. bin Abid, V. Mahajan, and L. Lucio. Machine learning for learnability of MDD tools. In *31st International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 355–468, 2019.

17. S. Bobek, M. Baran, K. Kluza, and G. J. Nalepa. Application of bayesian networks to recommendations in business process modeling. In *Workshop AI Meets Business Processes co-located with AI*IA*, volume 1101 of *CEUR Workshop Proceedings*, pages 41–50, 2013.
18. M. Borg, K. Wnuk, B. Regnell, and P. Runeson. Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. *IEEE Transactions on Software Engineering*, 43(7):675–700, 2017.
19. M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice, Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.
20. J. Brooke et al. SUS-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
21. P. Brosch, M. Seidl, and G. Kappel. A recommender for conflict resolution support in optimistic model versioning. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA Companion*, pages 43–50. ACM, 2010.
22. R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(Supplement 32):175–186, 2000.
23. R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-adapted Interaction*, 12(4):331–370, 2002.
24. C. Cai, J. Sun, and G. Dobbie. Automatic B-model repair using model checking and machine learning. *Automated Software Engineering*, 26(3):653–704, 2019.
25. G. C. Cawley and N. L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.
26. T. Cerqueira, F. Ramalho, and L. B. Marinho. A content-based approach for recommending UML sequence diagrams. In *28th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 644–649, 2016.
27. S. R. Chowdhury, F. Daniel, and F. Casati. Recommendation and weaving of reusable mashup model patterns for assisted development. *ACM Transactions on Internet Technology*, 14(2-3):21:1–21:23, 2014.
28. R. Clarisó and J. Cabot. Fixing defects in integrity constraints via constraint mutation. In *11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 74–82. IEEE Computer Society, 2018.
29. J. de Lara and H. Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In *5th International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2002.
30. M. C. de Oliveira, D. Freitas, R. Bonifácio, G. Pinto, and D. Lo. Finding needles in a haystack: Leveraging co-change dependencies to recommend refactorings. *Journal of Systems and Software*, 158, 2019.
31. S. Deng, D. Wang, Y. Li, B. Cao, J. Yin, Z. Wu, and M. Zhou. A recommendation system to facilitate business process modeling. *IEEE Transactions on Cybernetics*, 47(6):1380–1394, 2017.
32. A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.
33. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering (ICSE)*, pages 411–420. ACM, 1999.
34. A. Dyck, A. Ganser, and H. Lichter. Enabling model recommenders for command-enabled editors. In *1st International Workshop on Model-driven Engineering By Example (MDEBE@MoDELS)*, volume 1104 of *CEUR Workshop Proceedings*, pages 12–21, 2013.
35. A. Dyck, A. Ganser, and H. Lichter. A framework for model recommenders - requirements, architecture and tool support. In *2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 282–290. SciTePress, 2014.
36. A. Dyck, A. Ganser, and H. Lichter. On designing recommenders for graphical domain modeling environments. In *2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 291–299. SciTePress, 2014.
37. A. Elkamel, M. Gzara, and H. Ben-Abdallah. An UML class recommender system for software design. In *13th IEEE/ACS International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE Computer Society, 2016.
38. H. Florez, M. E. Sánchez, J. Villalobos, and G. Vega. Coevolution assistance for enterprise architecture models. In *6th International Workshop on Models and Evolution (ME@MoDELS)*, pages 27–32. ACM, 2012.
39. R. B. France, J. M. Bieman, S. P. Mandalaparty, B. H. C. Cheng, and A. C. Jensen. Repository for model driven development (remodd). In *34th International Conference on Software Engineering (ICSE)*, pages 1471–1472. IEEE Computer Society, 2012.
40. H. Garbe. Intelligent assistance in a problem solving environment for UML class diagrams by combining a generative system with constraints. In *eLearning*. IADIS, 2012.
41. M. Gasparic and A. Janes. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software*, 113:101–113, 2016.
42. P. Gomes. Software design retrieval using bayesian networks and wordnet. In *7th European Conf. on Advances in Case-Based Reasoning (ECCBR)*, volume 3155 of *Lecture Notes in Computer Science*, pages 184–197. Springer, 2004.
43. A. Großkopf, J. Brunnert, S. Wehrmeyer, and M. Weske. Bpmncommunity.org: A forum for process modeling practitioners - A data repository for empirical BPM research. In *Business Process Management Workshops, BPM*, volume 43 of *Lecture Notes in Business Information Processing*, pages 525–528. Springer, 2010.
44. E. Guerra, J. de Lara, M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Automated verification of model transformations based on visual contracts. *Autom. Softw. Eng.*, 20(1):5–46, 2013.
45. A. Gunawardana and G. Shani. Evaluating recommender systems. In *Recommender Systems Handbook*, pages 265–308. Springer, 2015.
46. I. Guy. Social recommender systems. In *Recommender Systems Handbook*, pages 511–543. Springer, 2015.
47. S. Hayashi, P. YiBing, M. Sato, K. Mori, S. Sejeon, and S. Haruna. Test driven development of UML models with SMART modeling system. In *7th International Conference on The Unified Modelling Language: Modelling Languages and Applications (UML)*, volume 3273 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2004.
48. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *26th Interna-*

tional Conference on the World-Wide Web (WWW), pages 173–182, 2017.
49. L. Heinemann. Facilitating reuse in model-based development with context-dependent model element recommendations. In *3rd International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pages 16–20. IEEE, 2012.
50. T. Hornung, A. Koschmider, and G. Lausen. Recommendation based process modeling support: Method and user experience. In *27th International Conference on Conceptual Modeling (ER)*, volume 5231 of *Lecture Notes in Computer Science*, pages 265–278. Springer, 2008.
51. T. Hornung, A. Koschmider, and A. Oberweis. A recommender system for business process models. *Information Technology & Systems*, 2009.
52. J. Huh, J. C. Grundy, J. G. Hosking, K. N. Li, and R. Amor. Integrated data mapping for a software metatool. In *20th Australian Software Engineering Conference (ASWEC)*, pages 111–120. IEEE Computer Society, 2009.
53. L. Iovino, A. Barriga, A. Rutle, and R. Heldal. Model repair with quality-based reinforcement learning. *Journal of Object Technology*, 19(2):17:1–21, 2020.
54. D. Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006. `http://alloytools.org/`.
55. D. Jannach, M. Jugovac, and L. Lerche. Supporting the design of machine learning workflows with a recommendation system. *ACM Trans. Interact. Intell. Syst.*, 6(1):8:1–8:35, 2016.
56. D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems - An Introduction*. Cambridge University Press, 2010.
57. J. Jézéquel, B. Combemale, O. Barais, M. Monperrus, and F. Fouquet. Mashup of metalanguages and its implementation in the Kermeta language workbench. *Software and Systems Modeling*, 14(2):905–920, 2015.
58. H. Jiang, J. Zhang, X. Li, Z. Ren, D. Lo, X. Wu, and Z. Luo. Recommending new features from mobile app descriptions. *ACM Transactions on Software Engineering and Methodology*, 28(4):22:1–22:29, 2019.
59. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.
60. F. Kahloun and S. A. Ghannouchi. Improvement of quality for business process modeling driven by guidelines. In *22nd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES)*, volume 126 of *Procedia Computer Science*, pages 39–48. Elsevier, 2018.
61. K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
62. S. Kelly and J. Tolvanen. *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley, 2008.
63. H. Khider, S. Hammoudi, A. Benna, and A. Meziane. Social business process model recommender: An MDE approach. In *5th International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 106–113. IEEE, 2018.
64. H. Khider, S. Hammoudi, and A. Meziane. Business process model recommendation as a transformation process in MDE: conceptualization and first experiments. In *8th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 65–75. SciTePress, 2020.
65. M. C. Kim and C. Chen. A scientometric review of emerging trends and new developments in recommendation systems. *Scientometrics*, 104(1):239–263, 2015.
66. K. Kluza, M. Baran, S. Bobek, and G. J. Nalepa. Overview of recommendation techniques in business process modeling. In *Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9)*, volume 1070 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
67. B. P. Knijnenburg and M. C. Willemsen. Evaluating recommender systems with user experiments. In *Recommender Systems Handbook*, pages 309–352. Springer, 2015.
68. S. Kögel. Recommender system for model driven software development. In *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 1026–1029. ACM, 2017.
69. S. Kögel, R. Groner, and M. Tichy. Automatic change recommendation of models and meta models based on change histories. In *10th Workshop on Models and Evolution (ME@MoDELS)*, volume 1706 of *CEUR Workshop Proceedings*, pages 14–19, 2016.
70. Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 77–118. Springer, 2015.
71. A. Koschmider, T. Hornung, and A. Oberweis. Recommendation-based editor for business process modeling. *Data & Knowledge Engineering*, 70(6):483–503, 2011.
72. T. Kuschke and P. Mäder. RapMOD - in situ autocompletion for graphical models: poster. In *39th International Conference on Software Engineering (ICSE), Companion Volume*, pages 303–304. IEEE Computer Society, 2017.
73. T. Kuschke, P. Mäder, and P. Rempel. Recommending auto-completions for software modeling activities. In *16th International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, volume 8107 of *Lecture Notes in Computer Science*, pages 170–186. Springer, 2013.
74. A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, volume 17, page 1, 2001.
75. Y. Li, B. Cao, L. Xu, J. Yin, S. Deng, Y. Yin, and Z. Wu. An efficient recommendation method for improving business process modeling. *IEEE Transactions on Industrial Informatics*, 10(1):502–513, 2014.
76. J. A. H. López and J. S. Cuadrado. MAR: a structure-based search engine for models. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 57–67. ACM, 2020.
77. P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.
78. S. Maki, S. Kpodjedo, and G. E. Boussaidi. Context extraction in recommendation systems in software engineering: A preliminary survey. page 151–160, USA, 2015. IBM Corp.
79. S. Mani, V. S. Sinha, P. Dhoolia, and S. Sinha. Automated support for repairing input-model faults. In

25th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 195–204. ACM, 2010.
80. J. Masthoff. Group recommender systems: Combining individual models. In *Recommender Systems Handbook*, pages 677–702. Springer, 2011.
81. P. Matikainen, P. M. Furlong, R. Sukthankar, and M. Hebert. Multi-armed recommendation bandits for selecting state machine policies for robotic systems. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4545–4551. IEEE, 2013.
82. S. Mazanek and M. Minas. Business process models as a showcase for syntax-based assistance in diagram editors. In *12th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, volume 5795 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2009.
83. D. Méndez, D. Graziotin, S. Wagner, and H. Seibold. Open science in software engineering. In *Contemporary Empirical Methods in Software Engineering*, pages 477–501. Springer, 2020.
84. G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.
85. MOF 2.5.1. `https://www.omg.org/mof/`, 2016.
86. N. Moha, S. Sen, C. Faucher, O. Barais, and J. Jézéquel. Evaluation of Kermeta for solving graph-based problems. *International Journal on Software Tools for Technology Transfer*, 12(3-4):273–285, 2010.
87. F. U. Muram, B. Gallina, and L. G. Rodriguez. Preventing omission of key evidence fallacy in process-based argumentations. In *11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 65–73. IEEE Computer Society, 2018.
88. K. Muslu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Speculative analysis of integrated development environment recommendations. In *27th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 669–682. ACM, 2012.
89. G. Mussbacher, B. Combemale, S. Abrahão, N. Bencomo, L. Burgueño, G. Engels, J. Kienzle, T. Kühne, S. Mosser, H. A. Sahraoui, and M. Weyssow. Towards an assessment grid for intelligent modeling assistance. In *23rd International Conference on Model Driven Engineering Languages and Systems, Companion Proceedings*, pages 48:1–48:10. ACM, 2020.
90. G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean, J. Jézéquel, T. Kühne, S. Mosser, H. A. Sahraoui, E. Syriani, D. Varró, and M. Weyssow. Opportunities in intelligent modeling assistance. *Software and Systems Modeling*, 19(5):1045–1053, 2020.
91. N. Nassar, H. Radke, and T. Arendt. Rule-based repair of EMF models: An automated interactive approach. In *10th International Conference on Theory and Practice of Model Transformation (ICMT)*, volume 10374 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2017.
92. A. Nechypurenko, E. Wuchner, J. White, and D. C. Schmidt. Applying model intelligence frameworks for deployment problem in real-time and embedded systems. In *Models in Software Engineering, Workshops and Symposia at MoDELS'06, Reports and Revised Selected Papers*, volume 4364 of *Lecture Notes in Computer Science*, pages 143–151. Springer, 2006.
93. P. Neubauer, R. Bill, T. Mayerhofer, and M. Wimmer. Automated generation of consistency-achieving model editors. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 127–137. IEEE Computer Society, 2017.
94. P. T. Nguyen, J. D. Rocco, D. D. Ruscio, L. Ochoa, T. Degueule, and M. D. Penta. FOCUS: a recommender system for mining API function calls and usage patterns. In *41st International Conference on Software Engineering (ICSE)*, pages 1050–1060. IEEE / ACM, 2019.
95. P. T. Nguyen, J. D. Rocco, D. D. Ruscio, and M. D. Penta. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*, 161, 2020.
96. X. Ning, C. Desrosiers, and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*, pages 37–76. Springer, 2015.
97. OCL. `http://www.omg.org/spec/OCL/`, 2014.
98. M. Ohrndorf, C. Pietsch, U. Kelter, and T. Kehrer. ReVision: a tool for history-based model repair recommendations. In *40th International Conference on Software Engineering (ICSE), Companion Proceeedings*, pages 105–108. ACM, 2018.
99. T. Pati, S. Kolli, and J. H. Hill. Proactive modeling: a new model intelligence technique. *Software and Systems Modeling*, 16(2):499–521, 2017.
100. S. Paydar and M. Kahani. A semantic web enabled approach to reuse functional requirements models in web engineering. *Automated Software Engineering*, 22(2):241–288, 2015.
101. S. Paydar and M. Kahani. A semi-automated approach to adapt activity diagrams for new use cases. *Inf. Softw. Technol.*, 57:543–570, 2015.
102. M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 1999.
103. A. Pescador and J. de Lara. DSL-maps: from requirements to design of domain-specific languages. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 438–443. ACM, 2016.
104. K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering, EASE*, Workshops in Computing. BCS, 2008.
105. K. Petersen, S. Vakkalanka, and L. Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.
106. L. Quijano-Sánchez, I. Cantador, M. E. Cortés-Cediel, and O. Gil. Recommender systems for smart cities. *Inf. Syst.*, 92:101545, 2020.
107. QVT 1.3. `http://www.omg.org/spec/QVT/`, 2016.
108. F. Rabbi, Y. Lamo, I. C. Yu, and L. M. Kristensen. A diagrammatic approach to model completion. In *4th Workshop on the Analysis of Model Transformations (AMT@MoDELS)*, volume 1500 of *CEUR Workshop Proceedings*, pages 56–65, 2015.
109. F. Rabbi, Y. Lamo, I. C. Yu, and L. M. Kristensen. Diagrammatic development of domain specific modelling languages with webdpf. *International J. Inf. Syst. Model. Des.*, 7(3):93–114, 2016.
110. M. E. Rangiha, M. Comuzzi, and B. Karakostas. Role and task recommendation and social tagging to enable social business process management. In

BPMDS/EMMSAD@CAiSE, volume 214 of *Lecture Notes in Business Information Processing*, pages 68–82. Springer, 2015.
111. J. Reimann, M. Seifert, and U. Aßmann. On the reuse and recommendation of model refactoring specifications. *Software and Systems Modeling*, 12(3):579–596, 2013.
112. F. Ricci, L. Rokach, and B. Shapira, editors. *Recommender Systems Handbook*. Springer, 2015.
113. M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.
114. J. D. Rocco, D. D. Ruscio, L. Iovino, and A. Pierantonio. Collaborative repositories in model-driven engineering. *IEEE Softw.*, 32(3):28–34, 2015.
115. L. M. Rose, R. F. Paige, D. S. Kolovos, and F. Polack. The Epsilon generation language. In *4th European Conf. on Model Driven Architecture - Foundations and Applications (ECMDA-FA)*, volume 5095 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.
116. R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle. Teaching modelling literacy: An artificial intelligence approach. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS), Companion Proceedings*, pages 714–719. IEEE, 2019.
117. J. Sánchez Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL model transformations. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 146–155. IEEE Computer Society, 2015.
118. J. Sánchez Cuadrado, E. Guerra, and J. de Lara. AnATLyzer: an advanced IDE for ATL model transformations. In *40th International Conference on Software Engineering (ICSE), Companion Proceedings*, pages 85–88. ACM, 2018.
119. J. Sánchez Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL transformations with speculative analysis. *Software and Systems Modeling*, 17(3):779–813, 2018.
120. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *10th International Conference on the World-Wide Web (WWW)*, pages 285–295, 2001.
121. M. Savary-Leblanc. Improving MBSE tools UX with ai-empowered software assistants. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS), Companion Volume*, pages 648–652. IEEE, 2019.
122. D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.
123. Á. M. Segura and J. de Lara. Extremo: An eclipse plugin for modelling and meta-modelling assistance. *Science of Computer Programming*, 180:71–80, 2019.
124. Á. M. Segura, J. de Lara, P. Neubauer, and M. Wimmer. Automated modelling assistance by integrating heterogeneous information sources. *Computer Languages, Systems and Structures*, 53:90–120, 2018.
125. Á. M. Segura, A. Pescador, J. de Lara, and M. Wimmer. An extensible meta-modelling assistant. In *20th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 1–10. IEEE Computer Society, 2016.
126. S. Sen, B. Baudry, and H. Vangheluwe. Domain-specific model editors with model completion. In *Models in Software Engineering, Workshops and Symposia at MoDELS'07, Reports and Revised Selected Papers*, volume 5002 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2007.
127. S. Sen, B. Baudry, and H. Vangheluwe. Towards domain-specific model editors with automatic model completion. *Simulation*, 86(2):109–126, 2010.
128. Simulink. `https://www.mathworks.com/products/simulink.html`, 2020.
129. C. D. Sipio, D. D. Ruscio, and P. T. Nguyen. Democratizing the development of recommender systems by means of low-code platforms. In *1st LowCode Workshop (LowCode@MoDELS)*, pages 68:1–68:9. ACM, 2020.
130. F. Steimann and B. Ulke. Generic model assist. In *16th International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, volume 8107 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2013.
131. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework, 2^{nd} Edition*. Addison-Wesley Professional, 2008.
132. M. Stephan. Towards a cognizant virtual software modeling assistant using model clones. In *41st International Conference on Software Engineering: New Ideas and Emerging Results (NIER@ICSE)*, pages 21–24. IEEE / ACM, 2019.
133. N. Tintarev and J. Masthoff. Evaluating the effectiveness of explanations for recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):399–439, 2012.
134. M. Tisi, J. Mottu, D. S. Kolovos, J. de Lara, E. Guerra, D. D. Ruscio, A. Pierantonio, and M. Wimmer. Lowcomote: Training the next generation of experts in scalable low-code engineering platforms. In *STAF (Co-Located Events)*, volume 2405 of *CEUR Workshop Proceedings*, pages 73–78. CEUR-WS.org, 2019.
135. M. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden, and K. Matsumoto. Javawock: A Java class recommender system based on collaborative filtering. In *17th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 491–497, 2005.
136. UML 2.5.1. `https://www.uml.org/`, 2017.
137. S. Witt, S. Feja, A. Speck, and C. Hadler. Business application modeler: A process model validation and verification tool. In *IEEE 22nd International Requirements Engineering Conference (RE)*, pages 333–334. IEEE Computer Society, 2014.
138. C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE*, pages 38:1–38:10. ACM, 2014.
139. C. Wohlin, P. Runeson, P. A. da Mota Silveira Neto, E. Engström, I. do Carmo Machado, and E. S. de Almeida. On the reliability of mapping studies in software engineering. *Journal of Systems and Software*, 86(10):2594–2610, 2013.