Towards model reuse in low-code development platforms based on knowledge graphs

Conference Paper · November 2022

DOI: 10.1145/3550356.3561570

Ilirian Ibrahimi
Johannes Kepler University of Linz

Towards Model Reuse in Low-Code Development Platforms Based on Knowledge Graphs

Ilirian Ibrahimi
JKU Linz, SE Institute
Austria
CLMS UK
Athens, Greece

Dimitris Moudilos
CLMS UK
Athens, Greece

Abstract

Low-code Development Platforms (LCDP) are applications to provide fast full-stack application development. These platforms rely on models as their core artifacts in order to reduce the complexity of software development. Despite the growing need and importance of using models in low-code platforms, there is still limited support for model discovery and reuse, which leads to unnecessary repetitive work. Therefore, facilities for automated discovery and recommendation of relevant models and model fragments are desired. A prerequisite for producing relevant recommendations for adding new features to a model is a model repository equipped with an efficient query mechanism in combination with algorithms for processing the retrieved data. In this paper, we present an approach that enables model recommendations by initially converting and merging heterogeneous models into a homogeneous graph, which serves as the repository of our approach. Afterwards, the approach queries the repository for relevant matches and uses N-grams to produce recommendations for models under development. The current approach is demonstrated on the zAppDev LCDP, but conceptually it can be integrated on any LCDP and can be applied for the reuse of any graph-based model. We evaluated our approach by reconstructing four different models that do not exist in our repository with the support of our approach.

CCS Concepts: • **Software and its engineering → Reusability**.

Keywords: MDE, Low-code platform, Model reuse, Recommendation system, N-grams, Graph-repository, RDF

ACM Reference Format:
Ilirian Ibrahimi and Dimitris Moudilos. 2022. Towards Model Reuse in Low-Code Development Platforms Based on Knowledge Graphs. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3550356.3561570

1 Introduction

Low-code Development Platforms (LCDP) are applications to provide fast full-stack application development with no programming knowledge requirements. All the necessary technical configurations, well-structured code generation, application deployment on the cloud, etc. will be done automatically [24, 27]. LCDP use models as their core artifacts and thus builds on top of them everything else automatically by using principles like high-level programming abstraction and model-driven engineering [8, 16]. The main goal of the LCDP is to build rapid enterprise applications as fast and as easily as possible. Thus, it is inevitable that the usage and importance of LCDPs are increasing continually [23, 25]. According to Gartner [26], in 2024 LCDPs will be responsible for the majority of the application development activities.

Different LCDP systems within similar domains tend to share models or some of their fragments (e.g., retail systems tend to share model fragments related to the management of customers, products, orders, and payments), which in the absence of effective discovery and reuse mechanisms are wastefully re-invented from scratch. This can hamper both productivity and feature completeness. As such, facilities for automated discovery and recommendation of relevant models (or model fragments) to be reused through semantic analysis of models are much desired. This model reuse task is performed by tools known as Modeling Assistants(MAs) [22]. MAs provide the modelers with helpful modeling suggestions by comparing the model under construction (or parts of the model) with a set of previously constructed models.

Hence, this paper presents an approach that enables model reuse by *i)* converting heterogeneous models to a homogenous graph format, *ii)* merging the homogenous graphs to a single graph which will serve as the *Knowledge graph* of the approach, and *iii)* getting the class names of the model under construction as input and uses *N-grams* to predict the next modeling step.

The novelty of this paper is the approach that facilitates model reuse by using a graph-based repository that conceptually can persist different level and language-agnostic models and provide model recommendations by using N-grams.

The current approach is integrated into the zAppDev[1] LCDP and is used for the reuse of different model within the zAppDev LCDP.

The paper is organized as follows: we will start by outlining the importance of a model reuse approach. For this, in Section 2 we will provide some background information relevant to the approach we have developed followed by a motivational example. Afterwards, in Section 3 we will explain the approach on how it persists models and how it provides recommendations. Following this, in Section 4, we will outline the tool support for our approach. In Section 5 we will depict the evaluation results of the approach and will explain them. Lastly, in Section 6 we will provide some related works to our approach, and in Section 7 we will present the conclusion of this work.

2 Background and Running Example

Fig. 1 represents a general overview of a model reuse approach. In this running example, we are building a Trip model. This "Trip" model should provide all the information for a trip from point A to point B. So far, we know that our model should contain a "Trip" class that has at least the attributes id, name, and numberOfAdults. And we know that a "Trip" class may be connected with a "Leg" class which represents a part of the trip. For example, let's say we have a Trip From Greece to London. That could consist of 2 Legs: 1. El. Venizelos - Heathrow (airplane), 2. Heathrow - London (train). The "Leg" class has the attributes id, TravelDate, ArrivalDate, etc. Finally, we know that a "Trip" class may have a one-to-many connection to the "Leg" class.

Let's assume that we have a repository where we can persist all the models we have created before, that are well-constructed by using probably the knowledge of domain experts and in the meantime developed by modeling experts. Hence, when the modeler attempts to create a model like the one depicted in Fig. 1, based on the current state of this model, the reuse approach should be able to find in the repository similar models with this model under construction, and by comparing the state of the model under construction with the similar model in the repository, the approach should be able to recommend to the user a class "Fare" which may be connected to the already created classes "Trip" and "Leg". There may be also information related to the connection between the recommended class and the existing ones (e.g. association, aggregation, composition, etc.)

Based on the state of the model under construction, the approach should also be able to reuse the information persisted in the repository related to the attributes of any given class and predict these relevant attributes to the user. Referring to the model under construction in Fig. 1. and the information available in the repository, our approach should also be able to recommend the attributes "idFare", "totalPrice", "taxAmount" etc. for the class "Fare", or attributes like "travelDate", "arrivalDate", etc. for the already existing class "Trip".

Motivated by this model reuse idea, we have created an approach that is capable to persist models in a repository and reuse the information they contain to support the modeler with useful modeling recommendations during the modeling process.

We believe, that it is important to preliminary explain the technologies we have used for developing this approach and the reasons that drove us to these decisions.

2.1 Background Information

To provide a more accurate and relevant model reuse approach, the need for more data is necessary. As much data having available increases the possibility to find something worth to be reused when creating new models. Although, the continually increasing amount of data directs us to the problem of how to manage this data. How should the data be persisted and how can they be discovered best? Consequently, scalability is a very important issue.

Due to [11] after conducting a performance comparison between relational databases and graph databases when handling large-scale social data, the author concluded that graph databases perform better on large-scale data and have some advantages over relational databases. Also, the comparative analysis of relational and non-relational databases in the context of performance in web applications conducted by Franczek et al [18] concluded that non-relational databases perform better when reading data. Thus, we decided to use graphs instead of relational databases.

Since there are also many graph databases available [6], since it has to scale[2] when using at least thousands of models, and since it has to be a good fit for the web [12] (because low-code platforms are cloud-based) we selected RDF (Resource Description Framework) since it is de facto the W3C standard since 1999[3]. An RDF graph is a collection of the subject, predicate, and object RDF triples and is used for representing data on the Web.

Concerning the graph discovery issue, we selected to proceed with SPARQL, the standard query language for data represented in a subject, predicate, object triple format (and even more). And finally, we selected to use TDB[4] - the RDF tailored-made repository as a repository backend for our approach.

[1]https://zappdev.com

[2]https://www.w3.org/wiki/LargeTripleStores

[3]https://www.w3.org/TR/2004/REC-rdf-concepts-20040210

[4]https://jena.apache.org/documentation/tdb

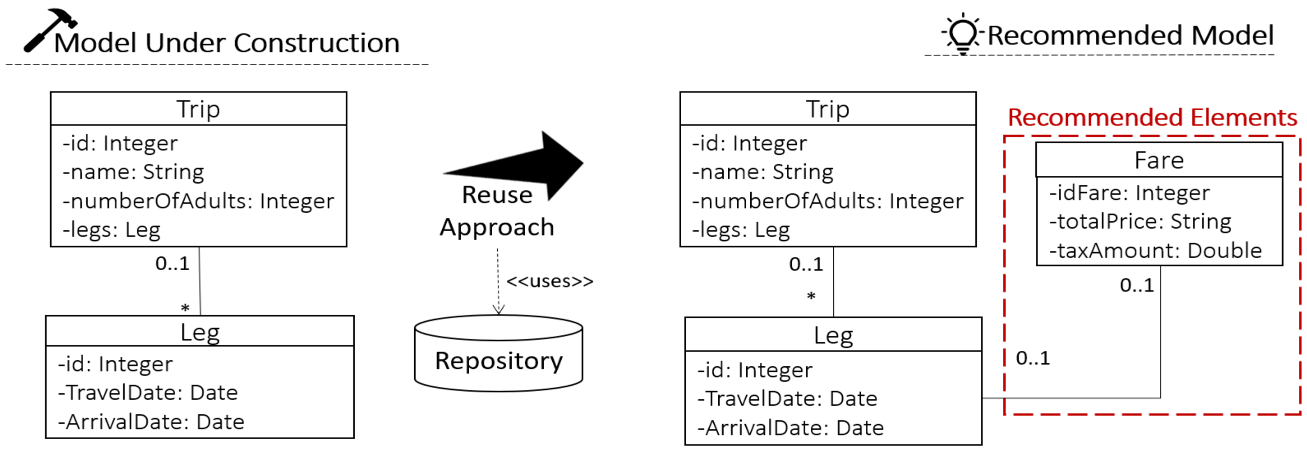

Figure 1. Running Example: Trip

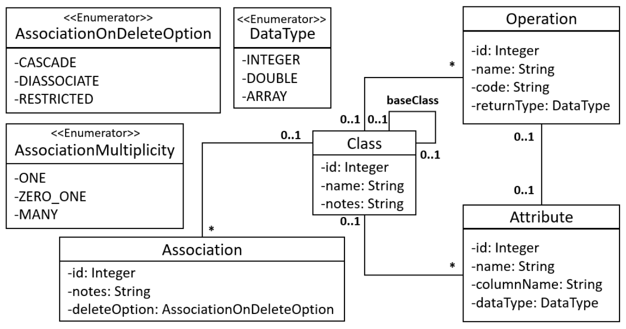

Figure 2. BO metamodel notations

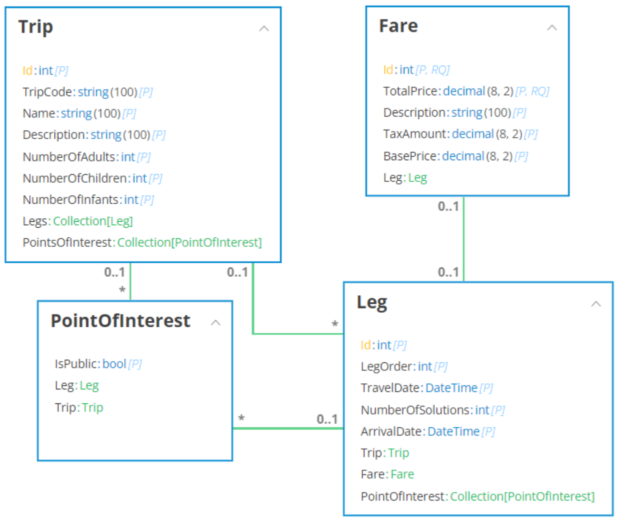
Trip
Id: int [P]
TripCode: string (100) [P]
Name: string (100) [P]
Description: string (100) [P]
NumberOfAdults: int [P]
NumberOfChildren: int [P]
NumberOfInfants: int [P]
Legs: Collection[Leg]
PointsOfInterest: Collection[PointOfInterest]
Fare
Id: int [P, RQ]
TotalPrice: decimal (8, 2) [P, RQ]
Description: string (100) [P]
TaxAmount: decimal (8, 2) [P]
BasePrice: decimal (8, 2) [P]
Leg: Leg
PointOfInterest
IsPublic: bool [P]
Leg: Leg
Trip: Trip
Leg
Id: int [P]
LegOrder: int [P]
TravelDate: DateTime [P]
NumberOfSolutions: int [P]
ArrivalDate: DateTime [P]
Trip: Trip
Fare: Fare
PointOfInterest: Collection[PointOfInterest]
0..1
*

Figure 3. An excerpt of the BO model named "Trip"

2.2 Business Object Models

Although the general concepts of reusing models from a graph-based repository can be also applied to other graph-based models e.g. XML, JSON, EMF, etc. in this work we have demonstrated the performance of this approach on the zAppDev LCDP models a.k.a business object models (BOs). BOs essentially are units of the model that encapsulate a single self-contained notion of the problem domain. These will include one or more classes, associations between these classes, and may also include operations that define the behavior of these classes. Fig. 2 depicts the notations of the zAppDev BO metamodel. And in Fig. 3 we depict an excerpt of a zAppDev model called "Trip".

3 Approach

This Section explains in detail the proposed model reuse approach.

3.1 Model Reuse Approach

In Fig. 4 are presented the steps our proposed approach takes to realize model discovery and reuse. As described in [19] the model reuse cycle is consisting of these four steps: *Abstraction*, *Selection*, *Specialization*, and *Integration*. Hence, we will explain the approach in the context of how it fulfills these model reuse steps.

Abstraction: This is one of the key elements of the reuse cycle. Thus, the abstraction process as described in step 1 of Fig. 4 is performed by converting heterogeneous models to RDF graphs. As outlined in the previews part of this section, we aim to use graph-repositories instead of relational ones, precisely we will use RDF as a graph for our approach. For

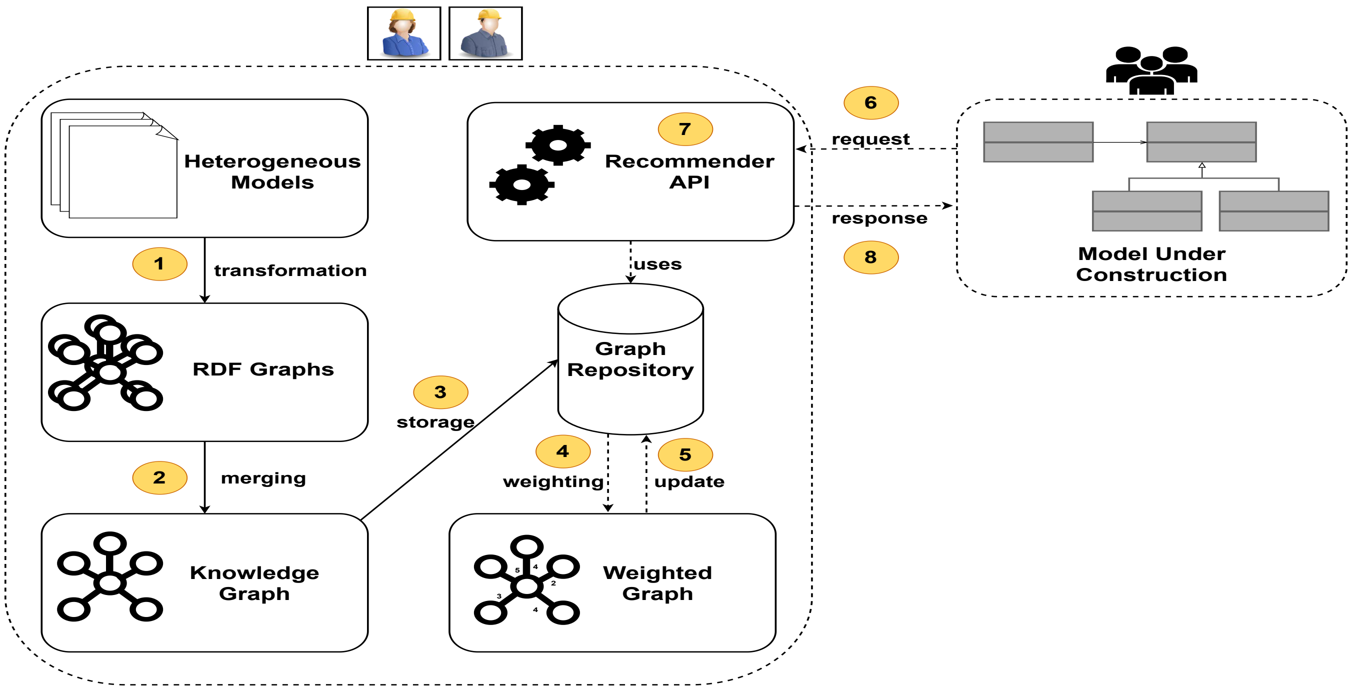

Figure 4. Overview of the model reuse approach

instance, in a particular case of abstracting zAppDev models (BOs), our approach performs the adjusting (e.g. adding URIs) and converting zAppDev into RDF/XML files. Afterward, as described in step 2, all these RDF files are merged into a single graph which in the end will serve as a knowledge base for our approach. As described in step 3, to be able to query and use the data of the merged RDF, it has to be persisted in a graph database, which in our case is the TDB. In order to use N-grams in the latter steps, we weighted this graph based on term (occurrence) frequency, e.g. we queried the entire repository to count how much any class B has occurred in relation to class A, this index has been added as the respective weight between class A and B. For the sake of clarity, let's assume class B occurred 5 times in relation to class A, then the connection between A and B would be weighted with 5 (A-5-B). The weighting process is presented as step 4 in Fig. 4. Finally, to keep these weights persistent in our graph, the entire graph has to be updated in the TDB - repository with the weights retrieved from the last step. In Fig. 4 this step is represented as step 5.

In conclusion, RDF serves as the abstraction format for our approach. So far, zAppDev models, Ecore models, and MoDisco reverse-engineered models [7] can be abstracted to the repository of our approach and can be reused, although, conceptually any other graph-based model can be converted to an RDF graph and reused by the proposed approach.

Selection: As mentioned before, we will use SPARQL for querying the repository. The first step of the selection part is getting the data from the model under construction, as depicted in Fig. 4 step 6 "request" part. If the approach will be triggered to provide recommendations for relevant classes from our repository e.g. the user triggers the approach after clicking any class or space on the modeling canvas, then the approach gets the data extracted from the model under construction and use them as an input for the SPARQL query for relevant class recommendations. We have distinguished the selection for class recommendations in **domain-specific** and **non-domain specific**. Since the same classes within different domains may have different connections e.g. a class "Manager" in a bank domain may be connected with classes like "Bank", "Client", "Account", etc, while a class "Manager" within a hospital model domain may be connected with classes like "Hospital", "Department", "Doctor", "Nurse", etc. To automatically determine the domain of the model under construction, we need to extract at least the name of two classes of that model. Next, we can use SPARQL to query our repository for a model name - which also determines the domain of the model - by using the names of the classes within the model under construction. When the given two or three classes of the model under construction belong to any model within the repository, then we can conclude that this model is the one that is domain-relevant to the model under construction and we have to look there for potential model elements to be reused. When the model under construction has only one class, then the approach will get its name and query the repository if there is such a model name there and get this as the model for providing domain-relevant recommendations.

In case the approach uses a non-domain-specific query then the approach searches for class recommendations in the entire repository.

Now, having found the relevant domain we proceed with finding potential model elements to be reused. Inspired by the work of Agt-Rickauer et al [2] we decided to use N-grams as a prediction algorithm. In our approach, we used 1-grams, 2-grams, and 3-grams. All of them are in domain-relevant and non-domain-relevant contexts. Thus, after the user triggers the approach for getting recommendations from a given class "Class1" (step 6 in Fig. 4), first it defines the relevant domain model (for domain-relevant queries), then it looks for all connected classes to "Class1" within that model (or in the entire repository for non-domain-relevant queries). The approach returns a list of all connected classes to "Class1" (when using 1-grams) ranked based on their term frequency. The classes that already are exiting in the model under construction will be removed while the others will be recommended to the user in a ranked way. The process is the same when using 2-grams and 3-grams, except the relation will be checked also for "Class2" and "Class3" respectively.

For a better understanding, let's refer to Fig. 5. As depicted we want to get recommendations about the modeling step after we have created the classes "Trip" and "Leg". If the approach is triggered from the "Trip" class, it checks for any "Trip" class within the repository, same if the approach will be triggered from the "Leg" class, since these classes are connected in the same model the result will be the same regardless from which of these class we want to get recommendations. After finding the "Trip" (or "Leg") class it performs N-grams to get all the related classes to Trip - Leg (2-grams in this case). The approach finds all connected classes with the relationship Trip - Leg within the repository and returns a list of those classes ranked based on their term -frequencies. Class "Fare" has the highest weight and is ranked as the first in the recommendation list. One of the most important model elements are also the model attributes (also shown in Fig. 1). The approach is also capable to find relevant model attributes for a given class. First, it queries the repository for the class whom we aim to find relevant attributes, then it queries all its attributes and returns them to be processed. The already existing attributes on the class under construction will be removed from the recommendation list, and the remaining ones will be recommended to the user to be reused.

Specialization: With specialization here, we mean to adopt for reuse the abstract model elements to a specific model format i.e. the format of the model under construction. In our context, it means that we just have to get the parameters i.e. the recommendations provided by the approach and use these parameters for the integration process. In Fig. 4 the specialization process happens in step 7.

Integration: After the approach has determined possible model elements that can be recommended and reused based

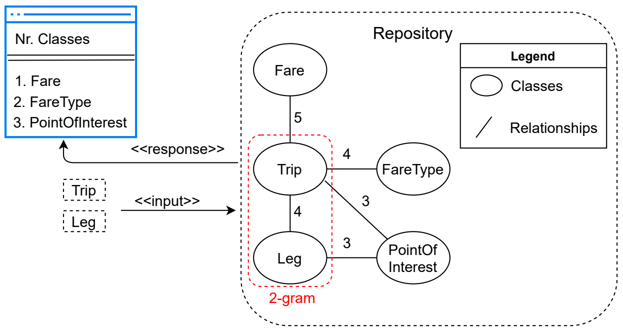

Figure 5. Extracting N-grams from the repository

on the inputs received from the state of the model under construction, the approach processes with the integration part. Currently, this approach is integrated into the zAppDev LCDP and is known as BORA. BORA is exposed as a REST API so other tools can access BORA's endpoints in order to get recommendations during the modeling process. In Fig. 4 the integration part is presented in step 8.

In section 4 will we explaining more about the integration process and tool support.

3.2 Handling Inexact Matchings

To get matches from the repository even if the user has made any typo or the class name of the model under construction is not the same as any class on the repository, we have added the Levenstein distance [20] to the approach so that it retrieves matches from the repository even if they differ for 2 characters. To match also word synonyms or words linked using conceptual-semantic and lexical relations to the name of the class of the model under construction we have also integrated WordNet[5] synonyms.

4 Tool Support

We implemented the above-mentioned approach by developing a tool called BORA (**B**usiness **O**bject **R**euse **A**pproach).

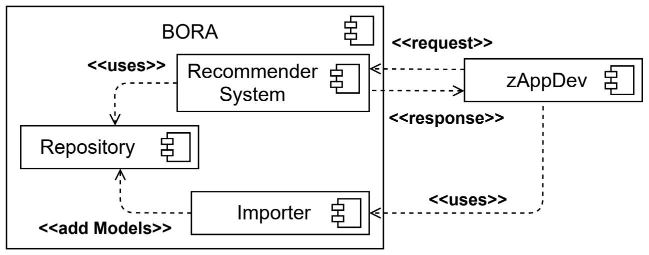

Figure 6. Architecture of BORA's integration into zAppDev.

As depicted in Fig. 6, zAppDev access BORA's functionalities by triggering any of its endpoints e.g. onegram (String entity), onegramdomainrelevant(String cl, String domain),

[5] https://wordnet.princeton.edu

twogram(String class1, String class2), etc. Depending on the endpoint, BORA's related recommendation function queries BORA's repository by triggering the relevant SPARQL query for that endpoint. The query results will be then processed by the recommendation function and will provide recommendations to zAppDev. In case users of zAppDev (or any other LCDP) wants to persist any model to BORA's repository so they can reuse it afterwards they can do so by triggering the relevant endpoint for inserting models which will instantiate the Importer class of BORA. Finally, the Importer class will persist the model to the repository by converting it to RDF, merging to the RDF-graph in the repository, and training again the repository in order to update the weights.

From the zAppDev perspective, when the users want to use BORA for getting modeling recommendations, as depicted in Fig. 7 they have just to click the "Business Object Suggester" button on the modeling canvas and BORA will provide recommendations to the user. After the user has selected the desired recommended classes/attributes, they will be automatically integrated into the zAppDev platform. In the class integration part, not only the classes will be integrated, but also the respective class attributes, connection type, connection name, and also the multiplicity that the recommended and selected class has to the class from which recommendations are triggered. An overview of a case on how BORA is used on zAppDev for class recommendations is depicted in Fig. 7. In a) is presented the current state of the model under construction for which the user selects any class (up to three mostly since we are provided no more the 3-grams) and triggers the recommendation button. In b) BORA provides recommendations and the user can select multiple of them. And in c) after the user has selected the desired recommendations and has pressed the "OK" button, the selected classes will be integrated on the modeling canvas together with all the information available in the repository. Depending on how many classes a user selects to ask for recommendations, the respective N-gram endpoint will be triggered, i.e. if the user selects one class, 1-gram endpoint will be triggered, if he selects two or three classes then the 2-gram or 3- gram endpoint respectively will be triggered. If the user asks for recommendations by not selecting any class, he will receive recommendations for "Island" classes within the respective domain. And finally, if the user presses the "Suggest attributes" button next to the "Business Object Suggester", he will receive recommendations for adding attributes within that class.

5 Evaluation

In this section, we will be explaining the evaluation of BORA by outlining the research questions that framed the evaluation process, the annotations used on the evaluation process, the evaluation process itself, the evaluation results, and finally, we will explain the results retrieved by our evaluation process.

5.1 Research Questions

In order to determine the evaluation process, the relevant metrics, and the evaluation methodology, we had to answer these research questions:

1. How effective is BORA by recommending model classes[6] during the model process of new models?
   - How effective are the domain-specific and non-domain-specific N-grams in recommending classes?
   - How effective are 1-grams, 2-grams, and 3 grams in recommending classes?
2. How effective is BORA by covering domain related model classes from the repository in order to provide recommendations?

5.2 Metrics

Before explaining the evaluation process, we will introduce the metrics we have used during the process. Let's refer to the model we aim to reconstruct from scratch as M. The annotations we have used during the evaluation process are TP (True Positives) - all recommendations that are the same or synonyms to the classes of M, FP (False Positives) - all recommendations that are not the same or synonyms to the classes of M, TN (True Negatives) - all model classes that are not the same or synonyms to the classes of M and are not recommended, and False Negatives (FN) - all model classes that are the same or synonyms to the classes of M but are not recommended. The metrics we have used for evaluating the approach are the common Precision, Recall, and F-measure [17] (since precision and recall have the same value of 0.5, we will be using F1-score). For evaluation of the ranking of the approach we used MAP (Mean Average Precision) [5]. Whereas, in order to see how many domain-relevant classes BORA was able to find from the repository and provide those for recommendations, we measured this with the metric coverage@N as given in [13].

- **Precision** is the report of the true positive recommendations to the total items recommended. Precision states the accuracy of recommendations:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

- **Recall** is the report of the true positive recommendations to the total items available in the model under construction. It shows how many model under construction classes could be recommended:

[6]Even though BORA is capable of reusing not only model classes, but also class attributes, attribute datatypes, class relationships, relationship names, and relationship multiplicity, we have evaluated only the class recommendations because the remaining entities will be integrated automatically after the user selects any recommended class.

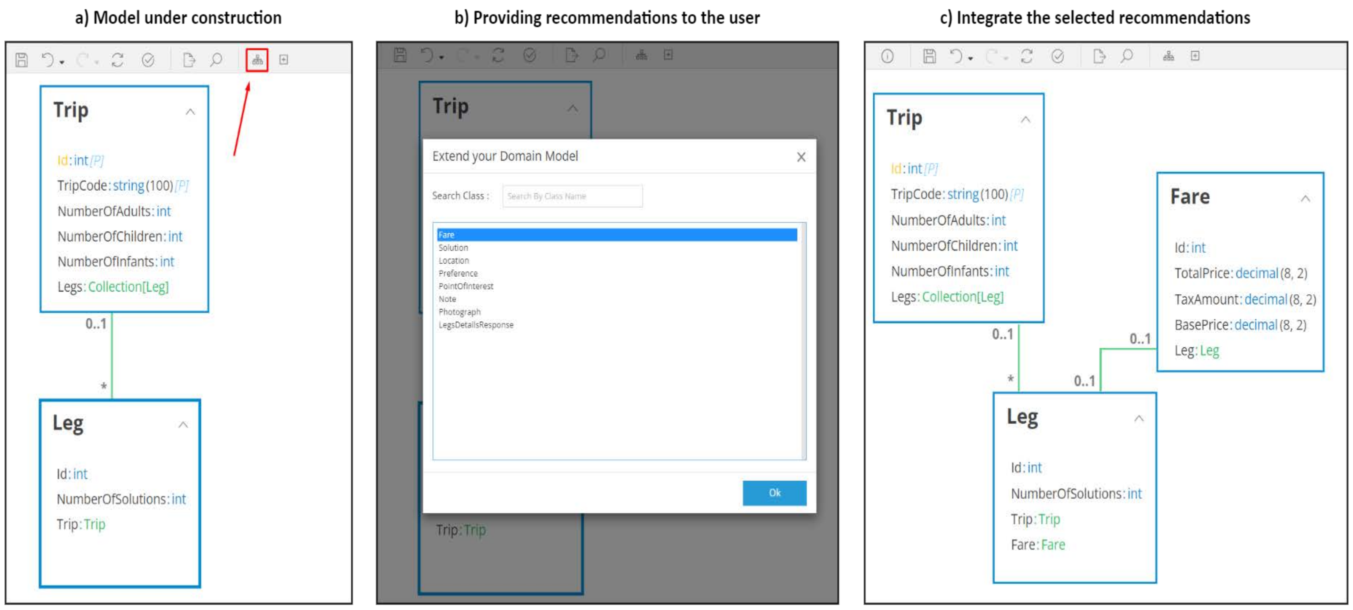

Figure 7. A screenshot of BORAs performace on zAppDev

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

- **F-measure** is a ratio that presents the harmonic average of precision and recall:

$$F_{measure} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

- **MAP** is the arithmetic mean of the average precision values for an information retrieval system over a set of n query topics:

$$MAP = \frac{1}{n} \sum_{n} APn \tag{4}$$

- **coverage@N** this metric measures the percentage of items recommended to projects, where I is the set of all items available for recommendation and P is the set of projects, as shown in eq. 5. In our case, instead of Projects we recommend just classes.

$$coverage@N = \frac{| \cup_{p \in P} \cup_{r=1}^{N} REC_r(p)|}{|I|} \tag{5}$$

5.3 Evaluation Methodology

In order to build and test BORA during the developing phase, we used 668 zAppDev models[7] which was converted and merged into a single RDF graph, and which finally was weighted based on term frequencies (class occurrence). This RDF graph served as a knowledge base for BORA to perform model recommendations and reuse. Although to answer the posed research questions, we selected four different models which are not in our repository (Knowledge base). Two of these models belong to the "Trip" domain and the remaining two belong to the "Order" domain. To evaluate BORA we tried to re-construct these four models from scratch by using the recommendations provided by BORA for each modeling step. By iterating through each class of the considered model, we calculated for each step the Precision, Recall, and F1 score for each iteration, and finally, we outlined the average as **"Avg."** and the standard deviation as **"SD"** of the results for each iteration step. Since BORA provides ranked results based on term frequencies, we have evaluated also the ranking process. To evaluate BORA's ranking, we have calculated the average precision for each iteration of the modeling step and finally calculated the mean of all the average precisions - which by definition gives the Mean Average Precision metric (RQ1). Attached to the MAP average we also outlined the standard deviation of MAP calculated through the re-construction process of the new models. Since BORA is a reuse-based recommendation approach, we calculated how many domain-relevant classes could BORA utilize from the repository in order to re-construct the new models. With utilization we concretely mean 1. how much could BORA explore from the repository in order to provide recommendations, it is important to know if BORA provides relevant recommendations then how many relevant classes do we

[7]Due to industrial property rights, these documents are not publicly available. However, we managed to get 543 Ecore models from the Maven repository (https://mvnrepository.com/), converted them to RDF, merged, and weighted the RDF graph so BORA can be tested by using this repository of Ecore models for free. BORA is available on GitHub at https://github.com/iliriani/BORA_Ecore

have on the repository, or if BORA doesn't provide any relevant recommendation then maybe there is nothing relevant to find in the repository? We named this evaluation metric coverage@N (RQ2). The evaluation results are depicted in Table 1

5.4 Results Overview

Based on the results depicted in Table 1, we will explain the posed RQs:

- **Answer to RQ1:** As shown in Table 1 precision is higher by domain-specific N-grams compared to the non-domain-specific ones. That's because the recommendation results for domain-specific N-grams are filtered to the particular model domain, the non-domain relevant recommendations will be removed. For the same reason, on the other side, the recall is higher in non-domain-specific cases as shown in the figure. More specifically, we can see that the precision is almost always highest for 1-grams domain-specific N-grams. That's because the models we tried to re-construct have a relatively small size (from 7 to 13 classes). When performing 2-grams and 3-grams on these models often occurred that the N-gram exceeded the size of models we aim to re-construct and didn't give any results. Also, the F1 score is almost the highest for 1-grams domain-specific due to the size of the models we aimed to reconstruct. As a conclusion for the recommendation part, we can say that our approach has promising results for recommending model classes when constructing models that are not in our repository.
  As depicted in Table 1, the ranking of BORA measured by Mean Average Precision (MAP) is more accurate on N-grams domain-specific cases. In general, domain-specific 3-grams have the highest MAP because they take 3 classes in comparison, as shown in the figure domain-specific 3-grams have a MAP of 1 (except in cases where no domain could be found that contains those 3 classes as in Model II and IV). The reason why the Precision for 3-grams is lower than the MAP is that there are also non-relevant classes recommended after the first relevant ones. Thus as in the case of Model I, we may have a MAP of 1 since all the relevant recommendations are the first ones, but we can have a lower precision (0.3667) because after the relevant recommendation we have some non-relevant ones. Concludingly for ranking of BORA we can say that BORA has a promising accuracy for its recommendations.
- **Answer to RQ2:** As depicted in Table 1, we can see that BORA was capable to discover and has recommended in most cases every domain-relevant class from the repository. It was able to recall an average of 95.96% of the existing relevant model classes from the repository, where in the repository we have 24 classes belonging to the "Trip" domain and 18 classes belonging to the "Order" domain. In most cases, we have a coverage@N of 1.

All the above-mentioned conclusions about the evaluation results can be seen more clearly in Fig. 8 where we outlined a comparison of the domain-specific and non-domain specific N-grams on all the evaluated metrics based on the results depicted in Table 1 (except coverage@N because it is always 1 except for Model III in domain-specific 3-gram).

5.5 Threats to Validity

In the following, we discuss the threats that may affect the validity of the findings of our experiment. The first threat that could impact the validity of the evaluation results is the models we choose for evaluation. Since BORA is N-gram-based, the size, the structure, and the domain of the models considered for evaluation significantly impact the evaluation results. To mitigate this threat, we considered four different models for evaluation, these four models belong to two different domains and the models of the same domain have different sizes and structures. And the second important factor that can threaten the validity is synonyms we considered as matches. For this, when BORA did not give exact matches, we selected by a consensus which of the algorithm's outputs can be considered as synonyms and treated them as true positives.

6 Related Works

Compared to other approaches, this is the only graph-based search approach that is capable to specify the relevant domain when given two different class names and producing modeling recommendations by using N-grams. It also differs from the other works in the aspect that it can reuse the knowledge of the language and level-agnostic models. It also is capable to reuse knowledge not only for class recommendations but also the information related to class attributes, attribute names and data types, relationship names, relationship multiplicities, etc. Although, we will explain briefly some related work to ours:

Di Rocco et al. [15] present MemoRec, a metamodel recommendation approach based on collaborative filtering. MemoRec encodes different metamodels in a graph representation, compares their similarity to the model under construction, and provides modeling recommendations retrieved after the metamodel and a model fragment comparison process.

In [28] Weyssow et al. present an approach based on a pre-trained language model for providing metamodel concepts recommendations. This approach gets lexical and structural information from metamodels, use these pieces of information to train a deep neural language model, and provides

Table 1. Results overview of BORA evaluation process

Domain	Model	Relevance	N-grams	**Precison**		**Recall**		**F1**		**MAP**		**coverage@N**
				Avg.	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	
Trip	Model I	Domain specific	1-gram	0.5014	0.1997	0.5333	0.0667	0.4810	0.1126	0.7597	0.2985	1.0000
			2-gram	0.5232	0.2280	0.6375	0.0650	0.5452	0.1916	1.0000	0.0000	1.0000
			3-gram	0.3667	0.1886	0.4667	0.0471	0.3867	0.1603	1.0000	0.0000	1.0000
		Non-domain specific	1-gram	0.3766	0.2520	0.5667	0.0816	0.4001	0.2019	0.5353	0.2923	1.0000
			2-gram	0.2550	0.1816	0.6571	0.0904	0.3377	0.1966	0.6130	0.2338	1.0000
			3-gram	0.0788	0.0530	0.5000	0.0000	0.1362	0.0713	0.4641	0.3224	1.0000
	Model II	Domain specific	1-gram	0.3841	0.2310	0.3625	0.2274	0.3566	0.2289	0.5309	0.2673	1.0000
			2-gram	0.2476	0.1524	0.2222	0.0000	0.2095	0.0762	0.5792	0.2542	1.0000
			3-gram	Not Applicable - no common domain where found								
		Non-domain specific	1-gram	0.2197	0.1737	0.2500	0.0500	0.2054	0.1094	0.3134	0.1718	1.0000
			2-gram	0.2292	0.1865	0.2716	0.0552	0.2172	0.1206	0.4435	0.4256	1.0000
			3-gram	0.0988	0.0660	0.2222	0.0786	0.1262	0.0643	0.3522	0.3219	1.0000
Order	Model III	Domain specific	1-gram	0.7108	0.3372	0.3333	0.1179	0.3728	0.1018	0.7917	0.2732	1.0000
			2-gram	0.1250	0.0000	0.4000	0.0000	0.1905	0.0000	0.4444	0.1288	1.0000
			3-gram	1.0000	0.0000	0.2500	0.0000	0.4000	0.0000	1.0000	0.0000	0.0476
		Non-domain specific	1-gram	0.3021	0.1362	0.2917	0.1382	0.2556	0.0839	0.7477	0.3099	1.0000
			2-gram	0.0993	0.0210	0.4800	0.0980	0.1646	0.0346	0.3512	0.1316	1.0000
			3-gram	0.2479	0.0894	0.3125	0.1672	0.2323	0.0185	0.3102	0.0400	1.0000
	Model IV	Domain specific	1-gram	0.8471	0.3059	0.1500	0.0972	0.2046	0.0623	0.9083	0.1833	1.0000
			2-gram	Not Applicable - no common domain where found								
			3-gram	Not Applicable - no common domain where found								
		Non-domain specific	1-gram	0.3856	0.3007	0.1458	0.0551	0.1688	0.0632	0.5823	0.2221	1.0000
			2-gram	0.3512	0.2235	0.2197	0.0582	0.2233	0.0555	0.5820	0.1983	1.0000
			3-gram	0.3187	0.2616	0.2000	0.1000	0.1787	0.0531	0.5704	0.1912	1.0000

modeling recommendations provided by this trained deep neural language model.

Also in [14], Di Rocco et al. present MORGAN, a GNN-based recommender system for facilitating the modeling process by assisting the specification of metamodels and models. The (meta) models are encoded in a graph-based format and afterward, a graph kernel function processes the graphs' information to provide model recommendations.

Angel et al. [4, 21] present Extremo - a heterogeneous modeling assistant. Extremo persists heterogeneous models into a single data model and provides different queries that allow the users to search for potential artifacts that can be reused.

In [9] Burgueño presents a framework that uses NLP techniques to process text documents, creates word embeddings and persists them on an NLP model, and provides model recommendations based on that model. This framework also considers the users' previous interaction with the model, what they selected and what they rejected. Similarly, in [10] Capuano et al. present a UML class recommender that learns from code repositories.

Henning Agt-Rickauer et al. [2] present an approach for domain modeling recommendation namely DoMoRe. DoMoRe uses a large-scale network of semantic-related terms extracted from different knowledge bases [1] to provide modeling recommendations during the modeling process.

Also, Lissete et al. [3] provides a model recommendation system for LCDP. The low-code user provides a meta-model for recommendation to the system. Then the user uses a textual DSL to determine which of the meta-model elements play the roles of users, items, and item-features. The DSL also specifies the number of recommended items that have to be offered to the developer, the recommendation method, format, etc.

7 Conclusion

In this paper, we have presented a Business Object Reuse Approach (BORA) that persists the models in an RDF/XML format and uses this as a knowledge base for recommending relevant model elements that can be reused during the construction of a new model. It uses SPARQL as a query language and N-grams as the prediction algorithms.

As the evaluation process revealed, BORA's performance seems to be promising by reusing the knowledge of previous models in order to construct new ones. BORA is not limited to reusing only the information related to the classes, it can also reuse the attributes of any class, the connection type, connection multiplicity, etc. The evaluation results outlined

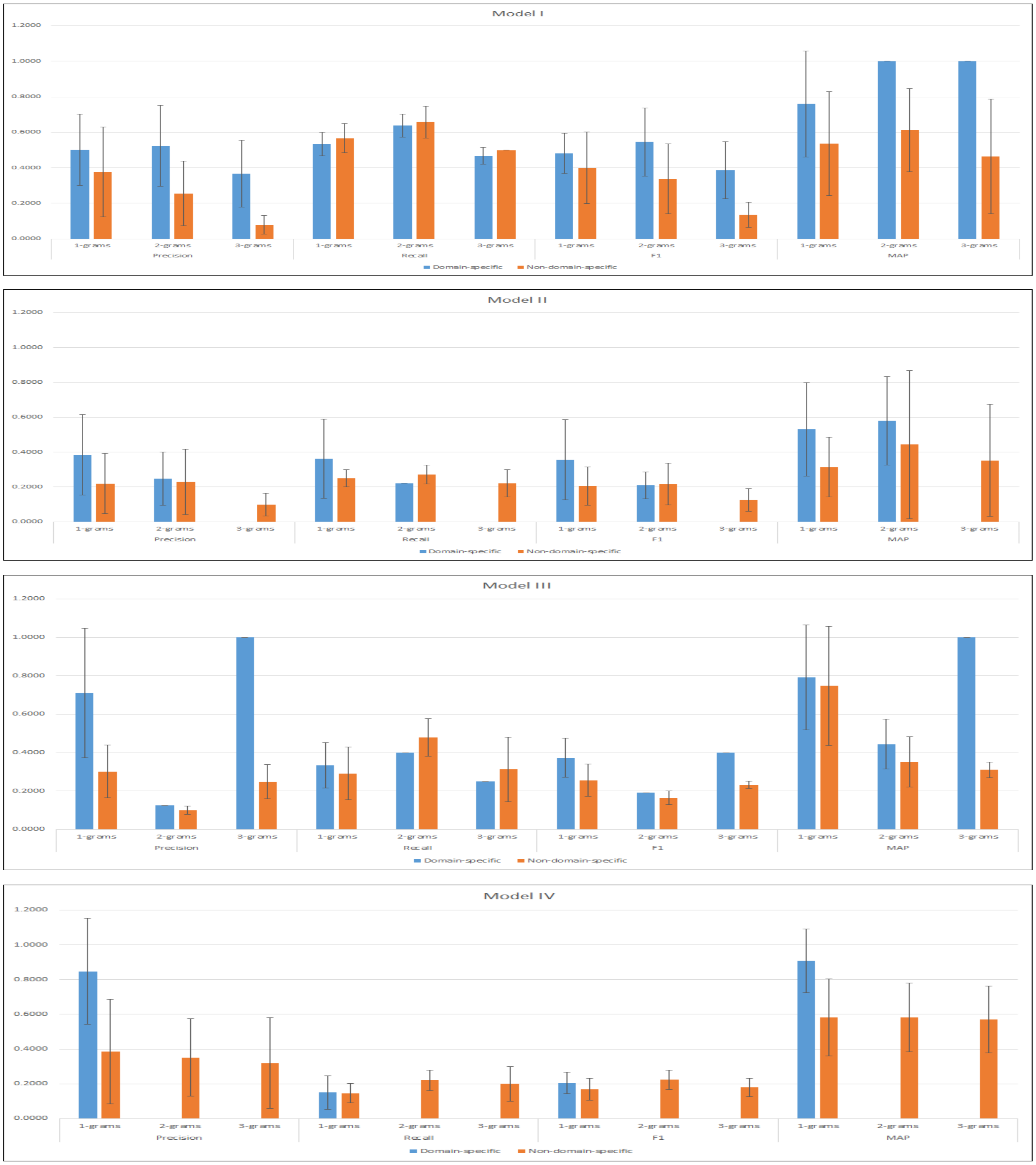

Figure 8. N-grams metrics comparison

that BORA can provide useful modeling recommendations when such relevant knowledge is existent in the repository.

BORA is available as a REST API and is integrated into the zAppDev LCDP.

8 Acknowledgments

This project has received funding from the EU Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 813884.

References

[1] Henning Agt and Ralf-Detlef Kutsche. 2013. Automated Construction of a Large Semantic Network of Related Terms for Domain-Specific Modeling. In *CAiSE*.

[2] Henning Agt-Rickauer, Ralf Detlef Kutsche, and Harald Sack. 2018. DoMoRe – A recommender system for domain modeling. *MODELSWARD 2018 - Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development* 2018-January, January (2018), 71–82. https://doi.org/10.5220/0006555700710082

[3] Lissette Almonte. 2020. Towards automating the construction of recommender systems for model-driven engineering. (2020).

[4] Mora Segura Ángel, Juan de Lara, Patrick Neubauer, and Manuel Wimmer. 2018. Automated modelling assistance by integrating heterogeneous information sources. *Computer Languages, Systems and Structures* (2018). https://doi.org/10.1016/j.cl.2018.02.002

[5] Steven M. Beitzel, Eric C. Jensen, and Ophir Frieder. 2009. *MAP.* Springer US, Boston, MA, 1691–1692. https://doi.org/10.1007/978-0-387-39940-9_492

[6] Amitabha Bhattacharyya and Durgapada Chakravarty. 2020. (Graph Database: A Survey). *2020 International Conference on Computer, Electrical and Communication Engineering, ICCECE 2020* (2020). https://doi.org/10.1109/ICCECE48148.2020.9223105

[7] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. 2014. MoDisco: a Model Driven Reverse Engineering Framework. *Information and Software Technology* 56, 8 (Aug. 2014), 1012–1032. https://doi.org/10.1016/j.infsof.2014.04.007

[8] Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. 2022. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling* Lcd (2022). https://doi.org/10.1007/s10270-021-00964-0

[9] Loli Burgueño, Robert Clarisó, Shuai Li, Sébastien Gérard, Jordi Cabot, Loli Burgueño, Robert Clarisó, Shuai Li, Sébastien Gérard, and Jordi Cabot An Nlp-based. 2021. An NLP-based architecture for the autocompletion of partial domain models To cite this version : HAL Id : hal-03010872 A NLP-based architecture for the autocompletion of partial domain models. (2021).

[10] Thibaut Capuano, Houari Sahraoui, Benoit Frenay, and Benoit Vanderose. 2022. Learning from Code Repositories to Recommend Model Classes. *Journal of Object Technology* 21, 3 (July 2022), 3:1–11. https://doi.org/10.5381/jot.2022.21.3.a4 The 18th European Conference on Modelling Foundations and Applications (ECMFA 2022).

[11] Yaowen Chen. 2016. Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data. (2016), 82.

[12] Philippe Cudré-Mauroux and Sameh Elnikety. 2011. Graph data management systems for new application domains. *Proceedings of the VLDB Endowment* 4, 12 (2011), 1510–1511. https://doi.org/10.14778/3402755.3402810

[13] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Riccardo Rubei. 2021. Development of recommendation systems for software engineering: the CROSSMINER experience. *Empirical Software Engineering* 26, 4 (2021). https://doi.org/10.1007/s10664-021-09963-7

[14] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Phuong T. Nguyen. 2021. A GNN-based Recommender System to Assist the Specification of Metamodels and Models. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 70–81. https://doi.org/10.1109/MODELS50736.2021.00016

[15] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Alfonso Pierantonio. 2022. MemoRec: a recommender system for assisting modelers in specifying metamodels. *Software and Systems Modeling* (2022). https://doi.org/10.1007/s10270-022-00994-2

[16] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling* (2022). https://doi.org/10.1007/s10270-021-00970-2

[17] Cyril Goutte and Eric Gaussier. 2005. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. *Lecture Notes in Computer Science* 3408, April (2005), 345–359. https://doi.org/10.1007/978-3-540-31865-1_25

[18] Adam Grzech, Leszek Borzemski, Jerzy Świątek, and Zofia Wilimowska. 2016. Preface. *Advances in Intelligent Systems and Computing* 430, October (2016), V–vi. https://doi.org/10.1007/978-3-319-28561-0

[19] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger. 2015. Reuse in model-to-model transformation languages: are we there yet? *Software and Systems Modeling* (2015). https://doi.org/10.1007/s10270-013-0343-7

[20] Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady* 10 (1965), 707–710.

[21] Ángel Mora Segura and Juan de Lara. 2019. EXTREMO: An Eclipse plugin for modelling and meta-modelling assistance. *Science of Computer Programming* (2019). https://doi.org/10.1016/j.scico.2019.05.003

[22] Gunter Mussbacher, Benoit Combemale, Jörg Kienzle, Silvia Abrahão, Hyacinth Ali, Nelly Bencomo, Márton Búr, Loli Burgueño, Gregor Engels, Pierre Jeanjean, Jean Marc Jézéquel, Thomas Kühn, Sébastien Mosser, Houari Sahraoui, Eugene Syriani, Dániel Varró, and Martin Weyssow. 2020. Opportunities in intelligent modeling assistance. *Software and Systems Modeling* June (2020). https://doi.org/10.1007/s10270-020-00814-5

[23] Why Read, This Report, and Key Takeaways. 2016. The Forrester Wave ™ : Low-Code Development. (2016).

[24] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020* August (2020), 171–178. https://doi.org/10.1109/SEAA51224.2020.00036

[25] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan De Lara, Esther M Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. 2019. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019) (CEUR Workshop Proceedings (CEUR-WS.org))*. Eindhoven, Netherlands. https://hal.archives-ouvertes.fr/hal-02363416

[26] Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis. 2019. Licensed for Distribution Magic Quadrant for Enterprise Low-Code Application Platforms. (2019), 1–34. https://www.gartner.com/doc/reprints?id=1-1ODOM46A{&}ct=190812{&}st=sb

[27] Robert Waszkowski. 2019. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* 52, 10 (2019), 376–381. https://doi.org/10.1016/j.ifacol.2019.10.060

[28] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. 2021. Recommending Metamodel Concepts during Modeling Activities with Pre-Recommending Metamodel Concepts during Modeling Activities with Pre-Trained Language Models. April (2021).