Noname manuscript No.
(will be inserted by the editor)

Model Discovery and Reuse by Knowledge Graphs for Low-Code Development Platforms: The zAppDev Case Study

Ilirian Ibrahimi · Luca Berardinelli · Yannis Zorgios

Received: date / Accepted: date

Abstract *Context*: Low-code Development Platforms (LCDPs) are tools that facilitate the rapid development of full-stack applications using models as their primary building blocks. However, there is limited support for discovering and reusing models in LCDPs, leading to unnecessary repetition of work. There is a high demand for tools that can automatically discover and recommend relevant models and model fragments to address this issue.

Contribution: This paper presents an extended version of a discovery and recommendation approach for LCPD using a model repository based on knowledge graphs. It expands the recommendation scope based on class name similarities by including attribute similarities. Accordingly, we propose an extended case study on reusing zAppDev models, emphasizing the emerging need for a model reuse approach.

Evaluation: Our proposed approach is evaluated using two methodologies. First, we regenerate from scratch four different models not present in our repository with the support of our approach. Second, we perform 10-fold cross-validation on 100 zAppDev models randomly selected from the repository. The evaluation revealed that recommendations based on attribute similarities outperformed those based on class name similarities.

✉ Ilirian Ibrahimi
Johannes Kepler University, Linz, Austria
E-mail: ilirian.ibrahimi@jku.at

✉ Luca Berardinell
Johannes Kepler University, Linz, Austria
E-mail: luca.berardinelli@jku.at

Yannis Zorgios
CLMS, Andrea Papandreo 19, Athens, Greece
E-mail: yz@clmsuk.com

Keywords MDE, Low-code platform, Model reuse, Recommendation system, Knowledge graphs, RDF, N-grams.

1 Introduction

Low-code Development Platforms (LCDPs) are tools that allow users to create full-stack applications without necessarily needing programming skills. Technical configurations, generating structured code, and automatically deploying applications to the cloud are just some tasks LCDPs can handle [27,31]. LCDPs use models as their primary building blocks and utilize high-level programming abstraction and model-driven engineering to construct applications efficiently [16,8]. The main goal of LCDPs is to make it easy and fast to build enterprise applications, and their usage and importance are continually growing [26,29]. According to Gartner [30], LCDPs will play a significant role in most application development activities by 2024.

LCDPs within similar domains often have models or parts of models in common (e.g., retail systems often have model fragments for managing customers, products, orders, and payments). Without effective discovery and reuse mechanisms, these models or fragments may be recreated from scratch, reducing productivity and limiting the completeness of features. To address these issues, it is desirable to have tools that can automatically discover and suggest relevant models or model fragments for reuse through semantic analysis of models. Modeling Assistants (MAs) [25] are tools that can help in model reuse. MAs offer modeling suggestions to modelers by comparing (parts of) the current model with a collection of previously built models.

In our previous works, we proposed a solution to reuse cross-language LCDP models through model slicing [20], and an approach to reuse models through modeling recommendations [21]. In this work, we extend [21] by:

1. Explaining in detail all the approach's steps, making explicit its workflow and the abstraction format of the knowledge graph
2. Extending the querying and recommendation capabilities based on class attributes,
3. Evaluating the new capabilities by creating four different models following recommendations triggered by class attributes.
4. Comparing class-name and class-attribute similarity-based recommendations,
5. Evaluating the complexity of our approach in terms of executed steps to recall a whole model from the repository.

The new extended approach is available in the zAppDev LCDP[1] for the zAppDev developers.

The paper is organized as follows: we start by outlining the importance of a model reuse approach. For this, in Section 2, we provide some background information relevant to our developed approach, followed by a motivational example. In Section 3, we reveal the need for model reuse in LCDPs and introduce the foundational technologies required to implement it. Afterward, in Section 4, we explain the approach on how it persists models and how it provides recommendations. Following this, in Section 5, we outline the tool support for our approach. In Section 6, we depict the evaluation results of the approach and explain them. Lastly, in Section 7, we provide some related works to our approach, and in Section 8, we conclude the paper.

2 Background and Motivational Example

In order to clarify the aim of this publication, we present a running example in Figure 1 related to model reuse on LCDPs.

We assume that the user wants to create a domain model regarding tourism. This model should contain information about points of interest, photographs, routes, fees, etc. Thus, the user starts by creating a model class `Photograph` with the attributes Id, Name, and Description and a class `PointOfInterest` containing the attributes Id, IsPublic, and Photographs. And finally, the user connects these classes with a one-to-many relationship. We consider this current model under construction

[1] `https://zappdev.com`

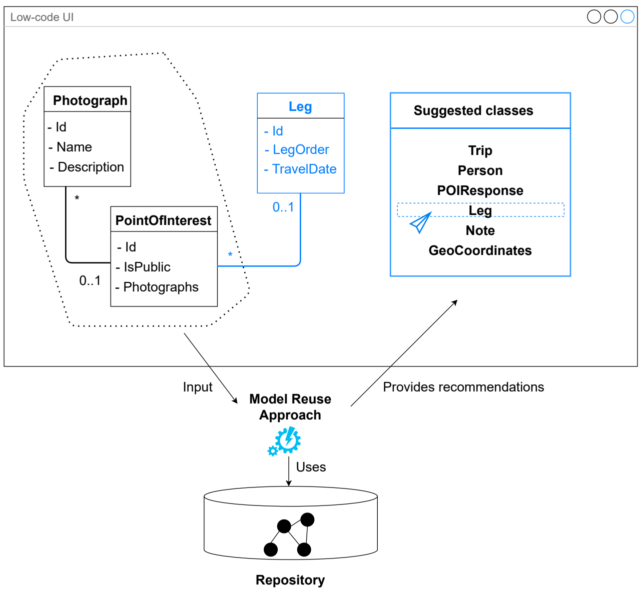


Fig. 1 Running Example: Tourism.

state as the point where the user asks for modeling support.

Based on the information retrieved from the model under construction, the reuse approach should be able to find, among others, a model related to the trip domain in the repository, which contains relevant tourism information. The approach retrieves, if any, one or more existing and valid relevant models. One of them is presented in Figure 2. By comparing the state of the tourism model under construction with the relevant model(s) in the repository provided in Figure 2, the approach recommends to the user a list of classes that can be relevant to the tourism domain, like `Trip`, `Person`, `POIResponse`, `Leg`, `Note`, and `GeoCoordinates`. In our example, we assume the user selects the class `Leg` to be integrated within the model under construction. Thus, the class `Leg` will be integrated together with its one-to-many association with the class `PointOfInterest`. In this context, a `Leg` refers to a specific segment of a trip. For example, if someone is traveling from Pristina to New York, the trip could be divided into two legs: one from Pristina to Istanbul and another from Istanbul to New York. Also, all the attributes related to the `Leg` class within the repository, i.e., Id, LegOrder, TravelDate, will be automatically integrated into the modeling canvas.

3 Enabling Model Reuse in LCPD

Based on the motivational example, we believe that an approach that can store and reuse different models would be a valuable asset for LCDPs. This approach

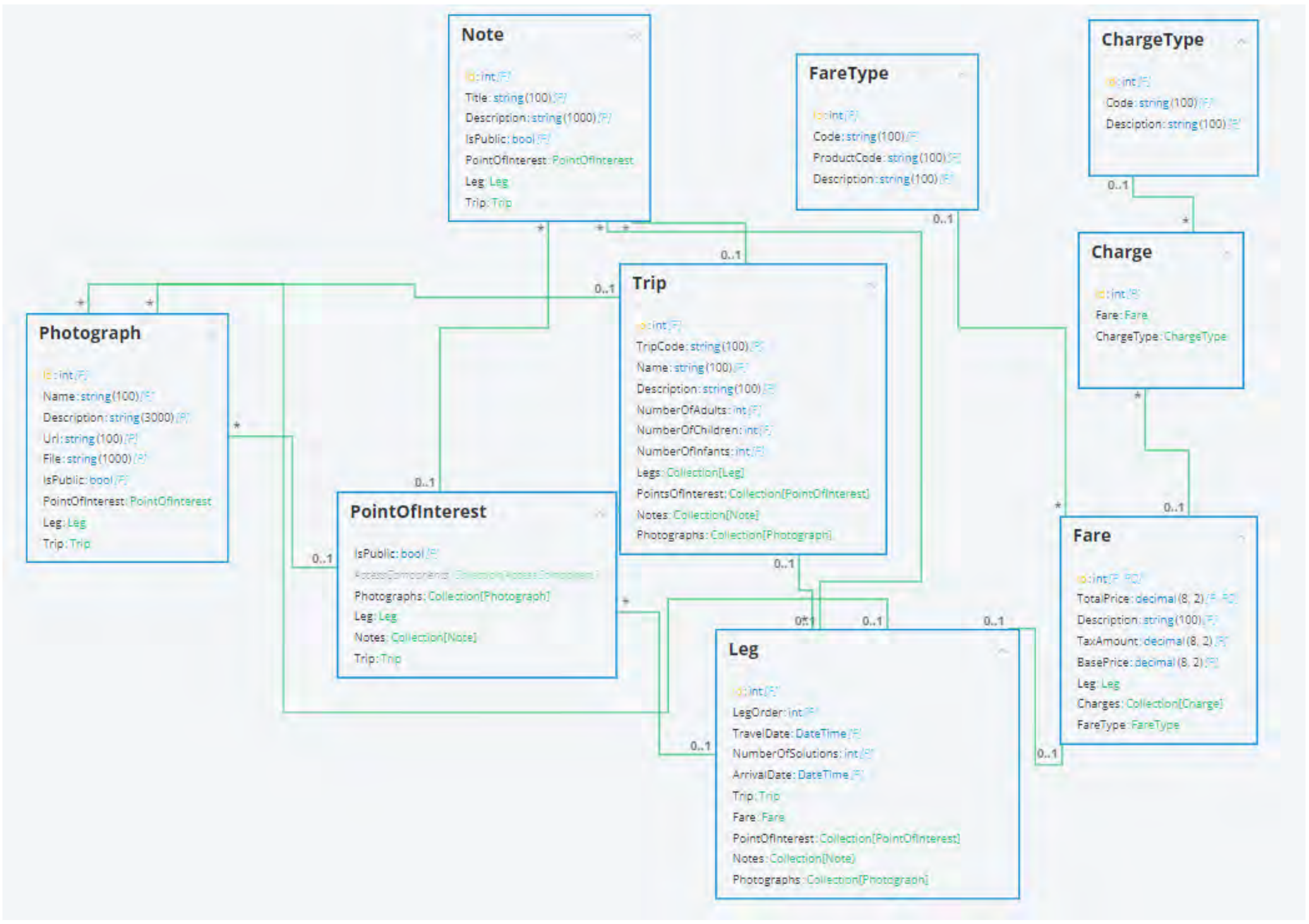


Fig. 2 An existing Trip model within the repository.

would facilitate the modeling process on an LCDP by providing useful modeling recommendations to users.

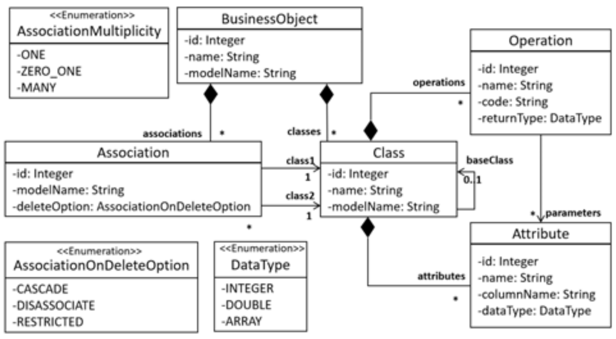


Fig. 3 The BO metamodel.

In the following, we will introduce the technologies that are suitable for implementing model discovery and reuse functionalities in LCDPs. We will also demonstrate how an exemplary LCDP, zAppDev, can benefit from these functionalities. Finally, we will outline the research questions related to this topic.

3.1 Technologies

Next, we present the key technologies that form the foundation of such an approach, i.e., graph databases, RDF, and the zAppDev LCDP.

3.1.1 Graph Databases and RDF

In order to improve the accuracy and relevancy of model reuse, more data is required. The greater the amount of data available, the more likely it is that something valuable can be found and reused when creating new models. However, the constantly increasing amount of data raises the issue of how to manage it efficiently. How should the data be stored, and how can it be easily accessed? Therefore, scalability is a crucial concern.

According to a performance comparison between relational and graph databases in handling large-scale social data [11], graph databases perform better and have certain advantages over relational databases. Additionally, a study by Franczek et al. [19] found that non-relational databases are more efficient when reading data in the context of web applications. For these reasons, we have used graph databases rather than relational ones.

There are several graph databases available [6], but in order to ensure scalability when using thousands of models and compatibility with the web [13], we have selected the Resource Description Framework (RDF), which has been the W3C standard since 1999[2]. RDF graphs consist of subject, predicate, and object triples and are used to represent data on the Web.

To address the issue of graph discovery, we have chosen to use SPARQL [18], the standard query language for data represented in subject, predicate, and object format. As a repository backend for our approach, we have selected TDB[3], a repository specifically designed for RDF data.

3.1.2 zAppDev and Business Object Models

The zAppDev LCDP is a web-based, model-driven development environment allowing developers of any technology and proficiency level to easily create, edit and reuse models of software artifacts (e.g., database models, business logic models, user interface models, and more), covering the complete application development lifecycle while having total control of the process.

Model reuse from a graph-based repository is applicable to any graph-based models, like XML, JSON, and EMF. In this paper, we demonstrate the performance of a discovery and recommendation approach on BO models, namely *business object* models (BOs). BOs encapsulate a single self-contained notion of the problem domain. These include a set of (hierarchical) classes, associations between classes, attributes, and parameterized operations defining classes' behavior. Figure 3 depicts an excerpt of the zAppDev BO metamodel. It is worth noting that this metamodel is introduced here for explanatory purposes. Such metamodel is used here to represent data structure manipulated by a zAppDev built-in C#-based API that programmatically manipulates BO models[4]. The model presented in Figure 2 shows a Trip BO model within the BO model editor.

The need for a model reuse approach is evident once inspecting zAppDev BO models. Figure 4 shows an excerpt of an *AccessComponent* BO model. In this model, three out of four classes are reused from other BO models (e.g., `SystemDetail` and `SystemInterfaceDetails` classes from a SystemDetails model, and `Location` class from a Location model). The newly created class is the `AccessComponent`.

3.2 Research Questions

In the outlined running example, this paper aims to answer the following research questions:

- **RQ1:** *How many classes are reused within a given dataset of an LCDP?*

By answering this RQ, we want to quantitatively assess if and how many classes are reused to construct new ones, given a dataset of models of an LCDP.

- **RQ2:** *What is the structure of the distinct classes?*

A *distinct class* refers to a class that is counted only once, even if it appears multiple times in a repository. By answering this RQ, we want to assess if the reused classes are islands or related to each other when used within different projects (models).

3.3 Quantitative Analysis of Model Reuse in zAppDev

In order to estimate the extent of model reuse in zAppDev, we collected 667 models from 19 real industrial projects covering different domains (expenses, logistics, finances, flight company services, etc.).

From a technical perspective, inspecting the XML serialization of BO models is illuminating. An excerpt is presented in Listing 1. The model root and each model element, grouped by `<Classes>` and `<Associations>` tags, declare the source BO model via the *ModelName* attribute[5]. If the root's and element's *ModelName* values match, the model element is created in the same model. Otherwise, it is imported from other models. For instance, Listing 1 shows associations imported together with their class ends (e.g., *Class1* and *Class2* attributes) from other models, marked with red rectangles (e.g., *AccessComponentFunction*), within the AccessComponent BO model.

By leveraging matches based on the *ModelName* attribute, it is possible to calculate model reuse metrics, like the number of i) *distinct classes*, *reused classes* (i.e., used out of the defining model), *connected* and *isolated classes* (depending on their participation in at least one association or not, respectively).

A knowledge graph is generated from the BO models dataset, and a new backend function has been implemented to calculate reused, connected, and island class metrics.

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

[3] `https://jena.apache.org/documentation/tdb`

[4] The zAppDev LCPD does not provide a metalanguage and then a metamodel artifact.

[5] The BusinessObject root element uses *Model_Name*. We ignore this syntactical difference for the sake of readability.

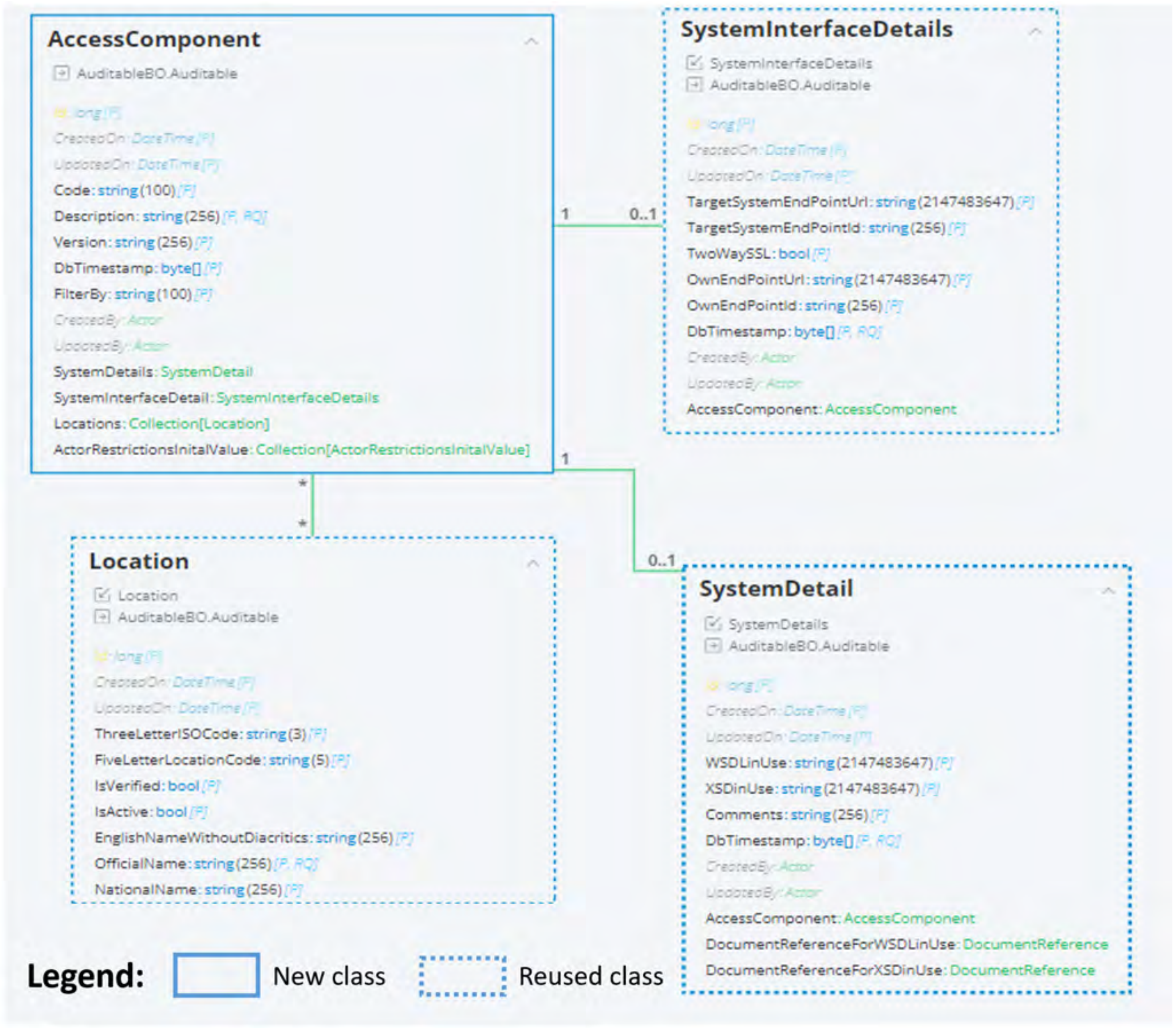

AccessComponent
AuditableBO.Auditable
long [P]
CreatedOn: DateTime [P]
UpdatedOn: DateTime [P]
Code: string(100) [P]
Description: string(256) [P, RQ]
Version: string(256) [P]
DbTimestamp: byte[] [P]
FilterBy: string(100) [P]
CreatedBy: Actor
UpdatedBy: Actor
SystemDetails: SystemDetail
SystemInterfaceDetail: SystemInterfaceDetails
Locations: Collection[Location]
ActorRestrictionsInitalValue: Collection[ActorRestrictionsInitalValue]
SystemInterfaceDetails
SystemInterfaceDetails
AuditableBO.Auditable
long [P]
CreatedOn: DateTime [P]
UpdatedOn: DateTime [P]
TargetSystemEndPointUrl: string(2147483647) [P]
TargetSystemEndPointId: string(256) [P]
TwoWaySSL: bool [P]
OwnEndPointUrl: string(2147483647) [P]
OwnEndPointId: string(256) [P]
DbTimestamp: byte[] [P, RQ]
CreatedBy: Actor
UpdatedBy: Actor
AccessComponent: AccessComponent
1
0..1
1
0..1
*
*
Location
Location
AuditableBO.Auditable
long [P]
CreatedOn: DateTime [P]
UpdatedOn: DateTime [P]
ThreeLetterISOCode: string(3) [P]
FiveLetterLocationCode: string(5) [P]
IsVerified: bool [P]
IsActive: bool [P]
EnglishNameWithoutDiacritics: string(256) [P]
OfficialName: string(256) [P, RQ]
NationalName: string(256) [P]
SystemDetail
SystemDetails
AuditableBO.Auditable
long [P]
CreatedOn: DateTime [P]
UpdatedOn: DateTime [P]
WSDLinUse: string(2147483647) [P]
XSDinUse: string(2147483647) [P]
Comments: string(256) [P]
DbTimestamp: byte[] [P, RQ]
CreatedBy: Actor
UpdatedBy: Actor
AccessComponent: AccessComponent
DocumentReferenceForWSDLinUse: DocumentReference
DocumentReferenceForXSDinUse: DocumentReference
Legend:
New class
Reused class

Fig. 4 Reused classes within the AccessComponent model.

3.4 Quantitative Results

Figure 5 reports the model metrics calculated over 667 BO models, allowing us to answer the RQs as follows:

RQ1: How many classes are reused within a given dataset of an LCDP? The dataset includes 667 BO models for a total of 2652 classes by counting how many times the *ModelName* attribute has been used within the `<Class>` tag in BO models. The so-called *distinct classes* are 1347, i.e., they are defined in one model and appear at least once in the given dataset, thus including also non-reused classes (i.e., distinct classes appearing exactly once in the repository). Therefore, we obtain that *some* distinct classes[6] have been reused outside the defining BO model, for a total of 1305 times. As a result, we obtain that 49% of the classes out of a total of 2652 are reused within different models.

RQ2: What is the structure of the distinct classes? We found out that 977 out of 1347 distinct classes (72%) are connected via associations, i.e., they are referenced either by *Class1* or by *Class2* attributes in the `<Association>` tag (cf. Listing 1).

[6] It is worth noting that it does not necessarily imply that *each* distinct class has been reused.

```

<BusinessObject Model_Name="AccessComponent" Model_Description="" Model_Creator="">
 <Associations>
  <Association ModelName="AccessComponent" Class1="DocumentReference" Class2="SystemDetails" Role1="
      DocumentReference">...</Association>
  <Association ModelName="AccessComponent" Class1="SystemInterfaceDetails" Class2="AccessComponent" Role1=
      "SystemInterfaceDetail"...></Association>
  <Association ModelName="AccessComponent" Class1="AccessComponent" Class2="Location" Role1="
      AccessComponents">...</Association>
  <Association ModelName=" SystemDetails" Class1="SystemInterfaceDetails" Class2="AccessComponent" Role1=
      "SystemDetails">...</Association>
  <Association ModelName="Location" Class1="Location" Class2="AccessComponent" Role1="Location">...</
      Association>
  <Association ModelName="SystemDetails" Class1="SystemDetail" Class2="AccessComponent" Role1="
      SystemDetail" Role2="AccessComponent" Multiplicity1="Many">...</Association>
  ...
 <Classes>
  <Class ModelName="AccessComponent" Name="AccessComponent" ...>
   <Attributes>
    <Attribute Name="Id" IsValueClass="false" Description="" DataType="long" ....
      MinLength="0" MaxLength="100" MinValue="" MaxValue="" Getter="" Setter="" />
    <Attribute Name="CreatedOn" IsValueClass="false" Description="" DataType="DateTime" ...</Attributes>
  </Class>
  <Class ModelName="SystemDetails" Name="SystemDetails" ShadowModel="SystemDetails"
   <Attributes>
    <Attribute Name="Id" IsValueClass="false" Description="" DataType="long" ... </Attributes>
  </Class>
</BusinessObject>
```

Listing 1 Excerpt of the AccessComponent BO model

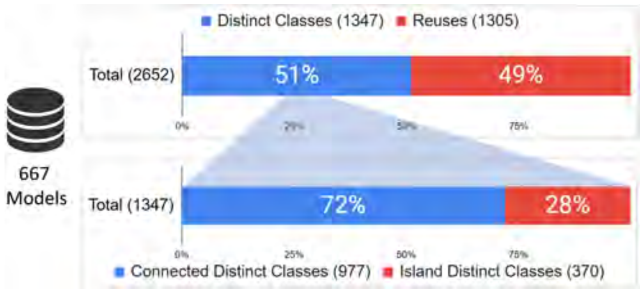


Fig. 5 Reused classes within the BO models.

3.5 Hypotheses and Considerations

Based on our findings and empirical results that answered our RQs, we can derive two hypotheses:

▲ **Hypothesis 1:** *A relevant number of the classes within a dataset of a given company are reused.*

▲ **Hypothesis 2:** *The majority of the classes within a dataset of a given company are connected.*

Half of the classes within different models constructed on a given LCDP from the same company (in our case, zAppDev) are reused to build models. Based on the answer to RQ2, we can hypothesize that the majority of classes within a dataset constructed by the same company are connected.

In conclusion, we can see that a huge amount of classes are reused within an LCDP, which shows the need for a model reuse approach. Also, since the research showed that most of the classes are connected, the reuse approach should consider class associations.

3.6 Threats to Validity

One potential threat that could significantly affect the results of this case study is the selection of models investigated. In our case study, we considered a dataset of BO models, which can vary significantly. To mitigate this potential threat to the validity of our case study, we used 667 different BO models created within 19 different projects.

4 Model Reuse Approach for LCDP

This section explains the main phases of a model reuse approach, extending the one presented in [21]. Its implementation for the zAppDev LCDP is called the Business Object Reuse Approach (BORA).

4.1 Usage Scenario

In Figure 6, we present the usage scenario of the proposed model reuse approach through a sequence diagram. The main actor is a citizen developer who starts modeling on an LCDP. She can ask for modeling support through the UI that, in turn, triggers the recommendation API providing the relevant information about the model under construction, i.e., classes and their attributes. Having lists of classes and attributes, the recommendation API queries the repository for classes connected to the classes of the model under construction. The connected classes will be returned to the recommendation API for processing and ranking. If no connected classes exist in the repository, the approach searches for classes declaring the same attributes of *classes* in the model under construction. If a class is found with the same attributes, the approach checks its connected classes and returns them to the recommendation API. The ranked list of connected classes will be provided to the citizen developer as a list of recommended classes. She can select them, and such user-selected classes will trigger another recommendation process. The latter will get the list of user-selected classes as input and queries the repository for related relevant information, i.e., their attributes, attribute data types, connection types, etc. Finally, all the user-selected classes, augmented with retrieved related information, will be integrated directly into the modeling canvas of the LCDP.

4.2 Model Reuse Approach Phases

A different perspective of the usage scenario depicted in Figure 6 is shown in the activity diagram in Figure 7 The latter better highlights how model reuse is realized through eight actions (①-⑨) over a client-server LCDP architecture. The actions implemented by LCDP engineers on the server side (①-⑤) realized a recommendation service that is invoked by citizen developers through the UI during modeling (⑥-⑨). In the following, we group such actions according to the model-reuse phases introduced by [22], i.e., *abstraction*, *selection*, *specialization*, and *integration* phases.

4.2.1 Abstraction

Steps ①-⑤ in Figure 7 realize the abstraction phase. In step ①, different graph-based models are parsed using appropriate model parsers for different model types (XML[7], Ecore[8], JSON[9], RDF[10]) and converted[11] to RDF graphs. In step ②, these graphs are merged into a single RDF graph using Apache Jena[12].

This merged RDF graph plays the role of the *knowledge graph*. It is persisted into a graph repository in step ③. To provide ranked modeling recommendations, the knowledge graph is weighted in step ④ by using *class term frequencies*, i.e., the number that presents the occurrence of class B after a class A will be inserted as the respective weight between these two classes. In step ⑤, the knowledge graph is updated by inserting the calculated weights on its edges.

4.2.2 Selection.

The selection phase is conducted in steps ⑥-⑦. The approach gets all relevant information from the model under construction, i.e., class names and class attributes, on the client side and generates a recommendation service request (⑥). The Recommender API is invoked and queries the repository if there is any relevant model element to be selected and recommended (⑦). The graph repository is an RDF-based weighted knowledge graph generated by the abstraction phase (①-⑤). Two selection algorithms are available, one based on class names and one based on class attributes. Based on the results, both classes and attributes can be recommended to the users through the LCDP UI. The following explains how the recommendation action work once requested by the citizen developer.

Class Recommendations Based on Class Names. As outlined in Figure 6, the approach checks the repository for model classes having the same names as the classes within the model under construction, using SPARQL queries.

To extract relevant modeling recommendations, the approach uses N-grams [12]. Currently, the approach uses 1-grams, 2-grams, and 3-grams if one, two, or three classes are selected on the model under construction to trigger the recommendation service. Figure 8 shows how recommendations (i.e., the response) are generated

[7] `https://docs.oracle.com/javase/8/docs/api/org/xml/sax/package-summary.html`

[8] `https://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html`

[9] `https://rdf4j.org/`

[10] `https://jena.apache.org/`

[11] In order to use the same SPARQL queries, the RDF source models has to be transformed to a specific RDF schema that is tailored for the zAppDev LCDP.

[12] `https://jena.apache.org/`

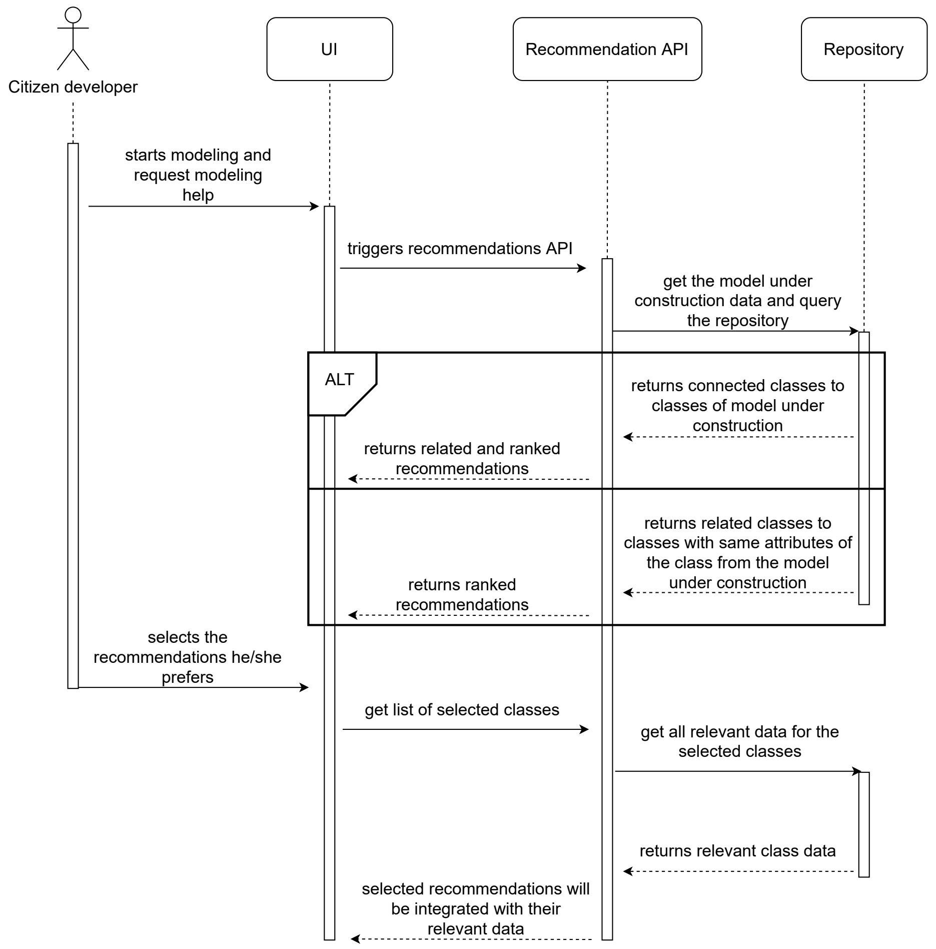

Citizen developer
UI
Recommendation API
Repository
starts modeling and request modeling help
triggers recommendations API
get the model under construction data and query the repository
ALT
returns connected classes to classes of model under construction
returns related and ranked recommendations
returns related classes to classes with same attributes of the class from the model under construction
returns ranked recommendations
selects the recommendations he/she prefers
get list of selected classes
get all relevant data for the selected classes
returns relevant class data
selected recommendations will be integrated with their relevant data

Fig. 6 Activity diagram of a recommendation scenario with an LCPD offering model-reuse by recommendation capabilities.

using 2-grams over the weighted knowledge graph by selecting two classes, `PointOfInterest` and `Photograph`, on the LCDP UI in Figure 1.

Existing models and recommended classes can both be classified as domain-relevant or non-domain-relevant. An existing model is considered *domain-relevant* if it contains *all* classes currently defined within the model under construction. If it does not, it is considered *non-domain-relevant.* Existing models without matching class names do not contribute to recommendations. The same classification applies to recommendations referring to such classes, i.e., domain-relevant recommendations include only domain-relevant classes, while non-domain-relevant recommendations do not. This scenario is shown in Figure 2. An existing Trip model is domain relevant for the Tourism model since the former defines all the classes currently included in the Tourism model under construction, i.e., `Photograph` and `PointOfInterest`. The approach queries the repository to get all pairs of `Photograph` and `PointOfInterest` classes and all related classes to these pairs, i.e., ends of association relationships.

If `Photograph-PointOfInterest` pairs cannot be found, the approach lessens from 2-grams to 1-grams for further querying. In our example, the approach query separately for the `Photograph` and `PointOfInterest` classes and checks for their related classes. Once all relevant classes are found, the approach responds to the users by recommending these classes ranked based on their term frequencies, prioritizing the domain-relevant recommendations.

Class Recommendations Based on Class Attributes. If the approach cannot provide any recommendation based on class names, class attributes are checked for similarity by counting the shared attributes amongst classes in the model under construction and the ones in the repository,

As shown in more detail in Figure 9, the approach initially takes as input the list of attributes of the class from which the recommendation service has been triggered (e.g., X, Y, and Z). Then, it queries the entire repository to get classes that contain these attributes. Since we have constructed the query to match any class

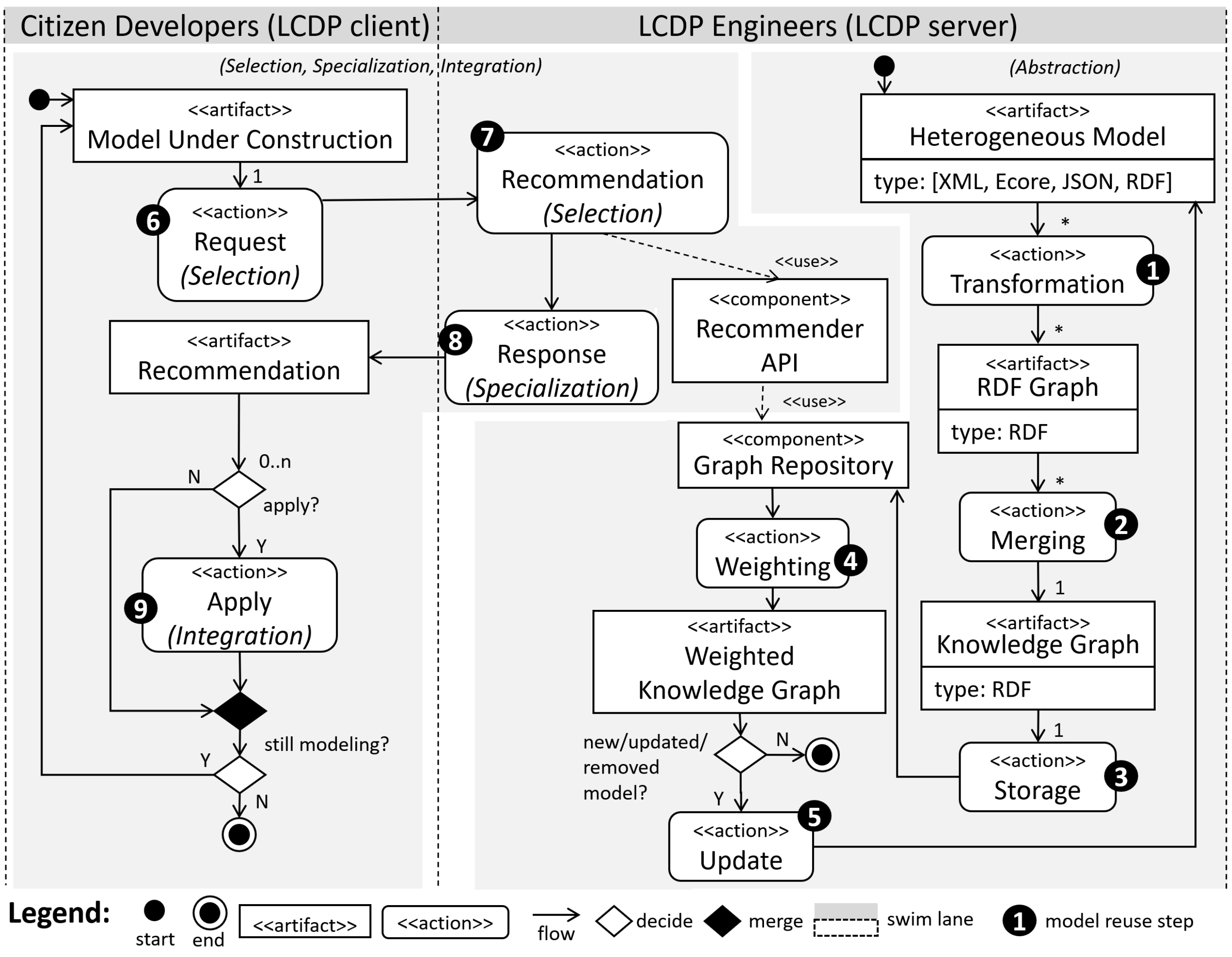

Citizen Developers (LCDP client)
LCDP Engineers (LCDP server)
(Selection, Specialization, Integration)
(Abstraction)
<<artifact>> Model Under Construction
<<action>> Request (Selection)
<<action>> Recommendation (Selection)
<<use>>
<<component>> Recommender API
<<action>> Response (Specialization)
<<artifact>> Recommendation
<<component>> Graph Repository
<<action>> Weighting
<<artifact>> Weighted Knowledge Graph
new/updated/ removed model?
<<action>> Update
0..n
apply?
<<action>> Apply (Integration)
still modeling?
<<artifact>> Heterogeneous Model
type: [XML, Ecore, JSON, RDF]
<<action>> Transformation
<<artifact>> RDF Graph
type: RDF
<<action>> Merging
<<artifact>> Knowledge Graph
type: RDF
<<action>> Storage
Legend:
start
end
<<artifact>>
<<action>>
flow
decide
merge
swim lane
model reuse step

Fig. 7 Overview of the model reuse approach.

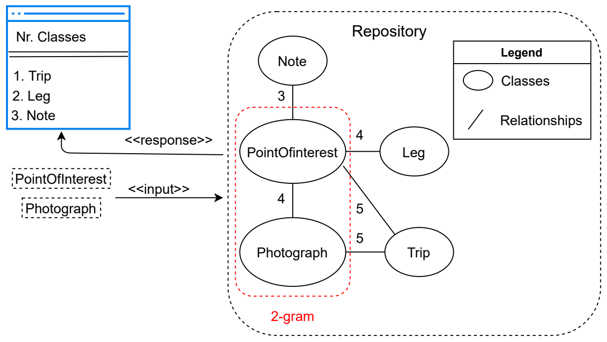


Fig. 8 Extracting 2-grams from the repository.

that contains Queries are built as logical disjunction prepositions of all involved attributes and then return the matched classes ranked based on the number of common attributes (i.e., ranked based on true arguments). Thereafter, the approach collects connected classes for each matched class in the repository. For instance, as depicted in Figure 9, `Class C3` contains three similar attributes X, Y, and Z w.r.t. `Class C1`. In the repository, `Class C2`, being directly connected to `Class C3`, is recommended for the next modeling step.

Class Attribute Recommendations Since all the model-relevant information is persisted in the Knowledge Graph, the approach can also recommend relevant class attributes during the modeling process.

While editing a class in the model under construction, the repository is queried for the same-named classes, and if any, all corresponding attributes are collected. The attribute list obtained from the repository is compared against already existing attributes from the currently modeled class, any matches are removed, and the remaining attributes are offered as recommendations to the user.

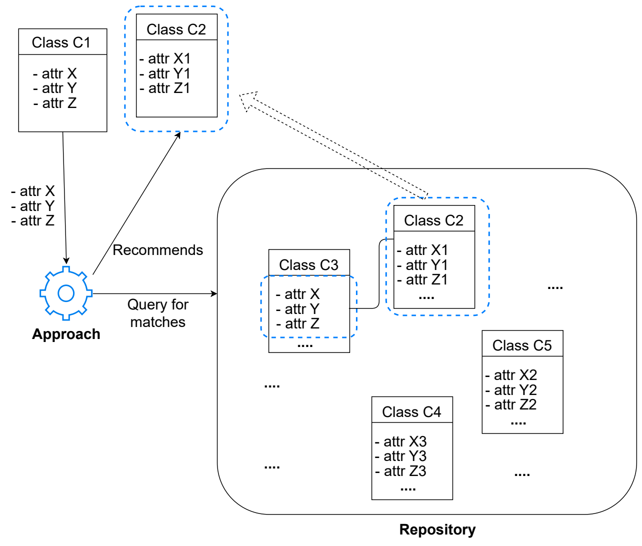
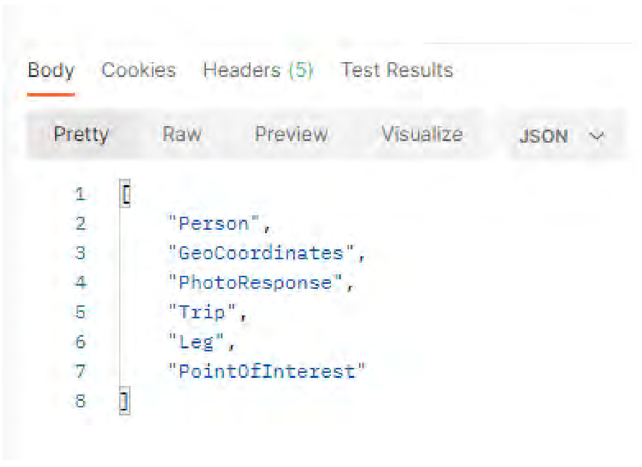


Fig. 9 Class recommendations from attributes.



```
[
    "Person",
    "GeoCoordinates",
    "PhotoResponse",
    "Trip",
    "Leg",
    "PointOfInterest"
]
```

Fig. 10 Specialization format of the model reuse approach.

4.2.3 Specialization

After the approach has selected the relevant model elements from the repository, a recommendation response has to be generated that is compatible with the target LCDP platform. The next step is, therefore, a specialization (8) that transforms the retrieved model elements into the specific format of the model under construction, making such model elements suitable for reuse. An example is shown in Figure 10. It depicts the recommendations for a `Photograph` class given as an array in JSON format generated from the RDF format leveraged by the graph-based repository.

4.2.4 Integration

The final step (9) is the integration of the selected modeling elements within the modeling canvas of the LCDP, thus leveraging UI-specific technologies. It happens when the citizen developer accepts the recommendation offered by the LCDP.

4.3 Handling Inexact Matchings

We leverage the Levenshtein distance [23] algorithm to find matches from the repository, even if there are typos or slight differences in the class or attribute name. This allows finding matches that differ by up to two characters. We also use WordNet[13] to find synonyms or related words to improve the matching process.

5 Model Reuse for zAppDev: the BORA Tool

The Business Object Reuse Approach (BORA) implements the model-reuse approach in Figure 7 for the zAppDev LCDP and BO models.

BORA is developed using Java Spring Boot and is exposed as a REST API to integrate within an LCDP. We integrated and used BORA into the zAppDev LCDP as a proof of concept.

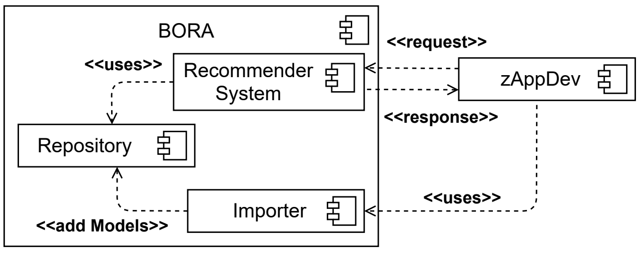


Fig. 11 The BORA architecture for zAppDev.

The overall BORA architecture for zAppDev is shown in Figure 11. The zAppDev LCDP accesses BORA's recommendation system API by triggering the relevant endpoints, for example:

– `onegram (String entity)` is used for getting class recommendations by considering only one class;
– `onegramdomainrelevant (String cl, String domain)` is used to get class recommendations for a given class by considering a specific domain;
– `twogram (String class1, String class2)` is used to get class recommendations by considering a class pair.

Based on the endpoint triggered from the LCDP, BORA triggers a SPARQL query over the repository for collecting relevant model elements. The chosen recommendation function processes the results, if any, and

[13] `https://wordnet.princeton.edu`

a list of model elements is sent as a response to the LCDP.

BORA allows importing models through the Importer component that persists the model to the repository by converting it to RDF, merging to the RDF-graph in the repository, and training the repository again to update the weights.

From the zAppDev perspective, when the users want to use BORA for getting modeling recommendations, as depicted in Figure 12 and in Figure 13, they have to click the "Business Object Suggester" button for class recommendations or the "Get attribute suggestion" button for attribute recommendation on the modeling canvas, and BORA provides recommendations to the user. After the user has selected the desired recommended classes or attributes, they are automatically integrated into the zAppDev platform.

When a recommended class is integrated, it automatically imports all its attributes and association information to the corresponding end (the class from which the recommendation service is triggered). Figure 12 shows how BORA has been integrated into the zAppDev LCDP. The following actions are illustrated:

a. Model under construction: The user asks for recommendations on the model under construction. Up to three classes can be selected at the same time. A dedicated button triggers the recommendation. 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, a 1-gram endpoint will be triggered; if he selects two or three classes, then the 2-gram or 3-gram endpoint respectively are invoked. it is worth noting that if the user asks for recommendations without selecting any class, recommendations for isolated classes are searched. In this particular case, BORA checks within the repository if there is any model containing the same name as the model under construction and returns all the isolated classes from that model.
b. BORA provides recommendations, and the user can select many of them.
c. After the user selects the desired recommendations and presses the "OK" button, the selected classes are integrated into the modeling canvas with all the relevant information in the repository.

The UI interaction is shown in Figure 13 for recommendations of class attributes.

6 Evaluation

This section provides an overview of the evaluation of BORA. It starts by identifying the research questions and then describes the metrics used. The evaluation is conducted using two different evaluation methodologies. The evaluation results are then presented, and the section concludes with a discussion of the insights gained from both evaluation methodologies.

6.1 Research Questions

The evaluation aims at answering these research questions:

- **RQ1: How does BORA perform on recommending modeling classes based on attribute similarities?** The goal is to assess how BORA performs in recommending relevant classes based on attribute similarities.
- **RQ2: Are class names or class attribute similarities more convenient for modeling recommendations?** The goal is to compare the performance of recommendations based on class names against class attributes and, thus, to determine whether priority shall be given first to classes or their attributes.
- **RQ3: How many steps does BORA need to recall a whole model from the repository?** Since BORA is meant to provide modeling recommendations iteratively, we can figure out how many iterations it would take to recall a complete model from the repository so the users can create a complete model only through the recommendations they get from BORA.

6.2 Metrics

It was necessary to determine the appropriate metrics and their combination to evaluate the BORA approach. The metrics are determined according to the following variables:

– M is a model that is reconstructed from scratch through BORA's recommendations.
– TP, true positives: all recommendations that match with the class names/class attributes in M.
– FP, false positives: all recommendations that don't match with the class names/class attributes in M.
– TN, true negatives: all model elements that don't match with the class names/class attributes in M and are not recommended.
– FN, false negatives: all class names/class attributes that match the model elements in M but are not recommended.

The metrics we used to evaluate the approach are the following:

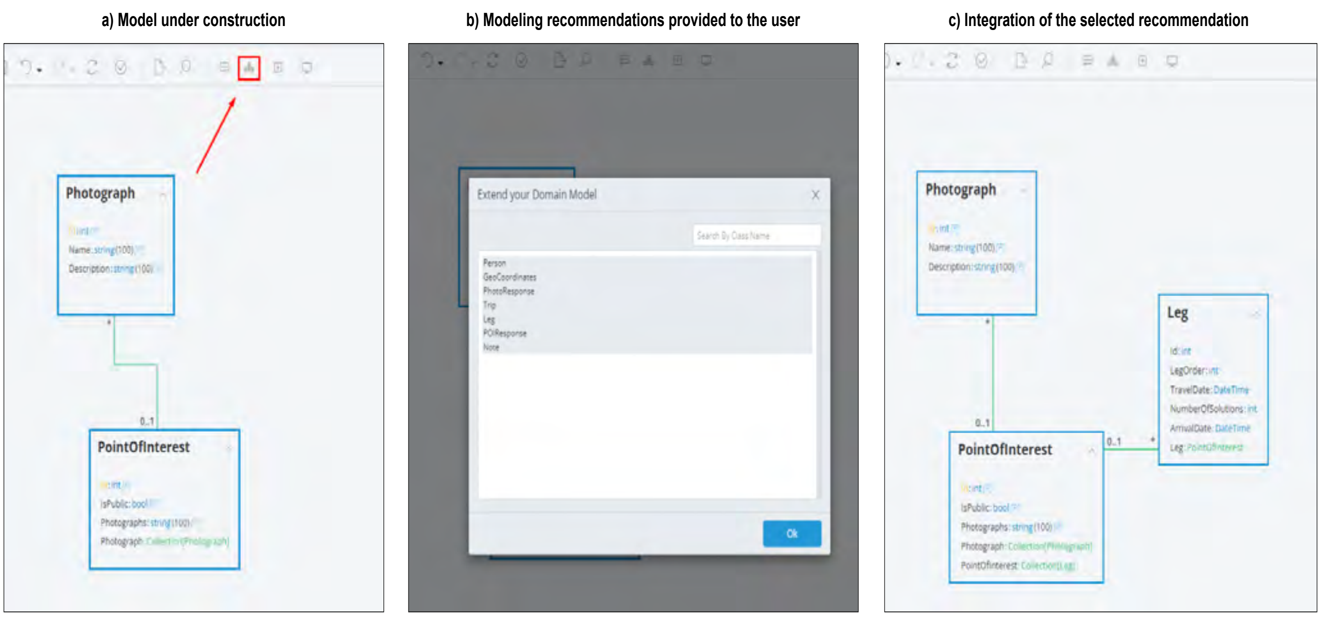

a) Model under construction
b) Modeling recommendations provided to the user
c) Integration of the selected recommendation
Photograph
PointOfInterest
Extend your Domain Model
Person
GeoCoordinates
PhotoResponse
Trip
Leg
POIResponse
Note
Ok
Leg
0..1

Fig. 12 A screenshot of BORA's class recommendation performance on zAppDev.

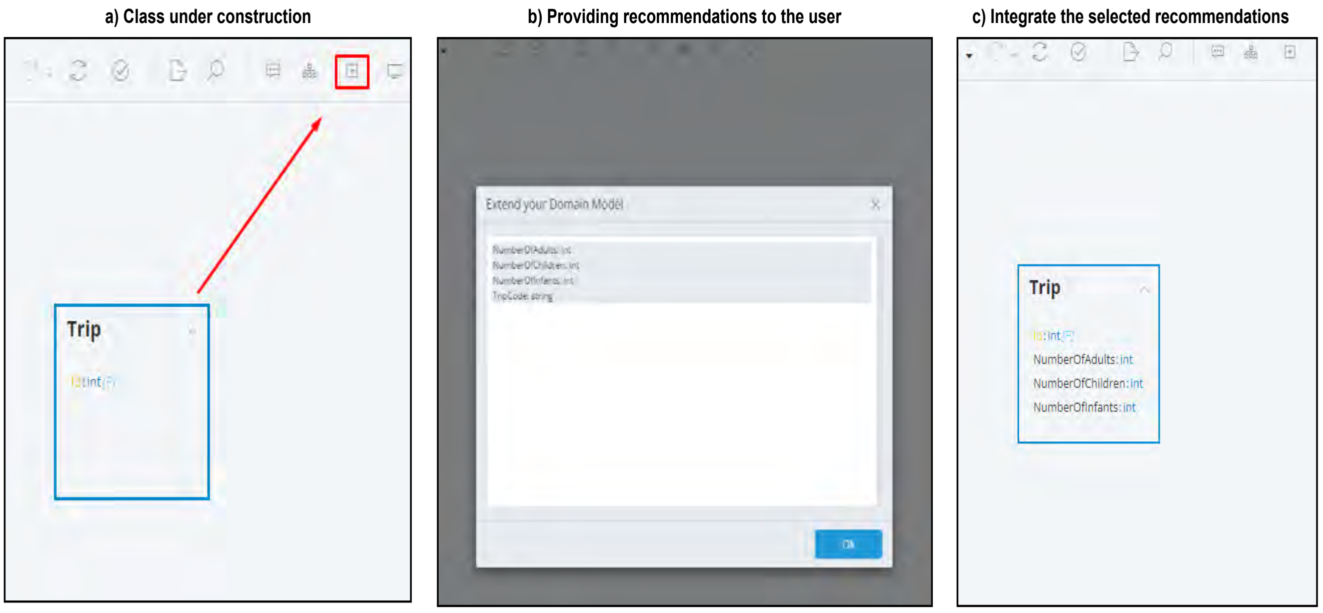


Fig. 13 A screenshot of BORA's attribute performance on zAppDev.

- **Precision** is the ratio between true positive recommendations and total recommended items. Precision states the accuracy of recommendations:

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

- **Recall** is the ratio between TP recommendations and the total items available in M, which can be true or false positives. It quantifies how many classes in M can be recommended:

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

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

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

It is worth noting since precision and recall have the same weight of 0.5, we use the $F1_{score}$, which is the specific case of the $F_{measure}$ metric when Precision and Recall are given the same weights (i.e., 0.5).

- **MAP** [5] is the arithmetic mean of the average precision values for an information retrieval system over a set of n recommendations:

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

  We use MAP to evaluate the approach's ranking.

- **coverage@N**, in [14], is the percentage of items $i \in I$ recommended to projects $p \in P$, where I is the set of all items available for recommendation and P is the set of projects. We reuse this metric and apply it to classes $c \in C$ rather than projects.

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

6.3 Evaluation Methodologies

The evaluation was carried out using two methodologies detailed in the following.

6.3.1 Evaluation Methodology A

This evaluation methodology has been presented in our previous work [21]. During the development phase of BORA, we collected 667 zAppDev models. Due to industrial property rights, the zAppDev models mentioned in this paper cannot be made available to the public. However, it is pertinent to note that a series of RDF models derived from the Ecore/EMF models have been diligently uploaded to our designated GitHub repository[14]Notably, the Ecore/EMF models themselves are openly and publicly accessible via the established Maven repository[15]. All 667 zAppDev models were transformed into RDF graphs and then combined into one single RDF graph. This graph was then weighted based on the frequency of terms, such as the occurrences of classes. This weighted RDF graph serves as the knowledge base for BORA to make model recommendations.

In order to address the research questions, four additional models were collected, two pertaining to the Trip domain and two to the Order domain. These models are also available on the paper's companion GitHub repository. However, they are not included in the generated repository as their sole purpose is for evaluation. We also used those four models to evaluate our previous work [21].

The evaluation results are summarized in Table 1. To evaluate BORA, we attempted to reconstruct these four models from scratch using only recommendations provided by BORA for each modeling step (i.e., repeating steps ⑥-⑨ in Figure 7). For each iteration through each class of the selected model , we calculated the Precision (Equation 1), Recall (Equation 2), and F1 score (Equation 3). Finally, we computed the average (Avg) and standard deviation (SD) for each metric.

Since BORA provides ranked results based on class occurrences (i.e., term frequencies), we also evaluated the ranking process. To evaluate BORA's ranking, we calculated the average precision for each iteration of the modeling step and then calculated the mean of all the average precisions to obtain the MAP (see Equation 4). In addition, we calculated the MAP standard deviation.

Since BORA is a reuse-based recommendation approach, we calculated how many domain-relevant classes BORA could utilize from the repository to reconstruct the new models. Utilization refers to the extent to which BORA could explore the repository to provide recommendations. When BORA provides relevant recommendations, it is important to know how many relevant classes are in the repository. Similarly, if BORA does not provide any relevant recommendations, it is important to confirm that nothing relevant can be found in the repository. This evaluation metric is coverage@N 5, first introduced by Di Rocco et al. in [14] and it is relevant for answering RQ1 and RQ2.

6.3.2 Evaluation Methodology B

Another evaluation methodology of BORA was conducted to eliminate any potential bias related to the evaluation methodology, such as the selection of test models, the consideration of synonyms as true positives, and manual evaluation.

For this reason, we randomly selected 100 zAppDev models and used 10-fold cross-validation (10% for testing and 90% for training) to outline the performance of BORA in different cases and compare the results for the recommendations generated from class name similarities and those from attribute similarities. Concerning the recommendations based on class name similarities, we used BORA's 1-grams, i.e., we consider the related classes of only these classes whose attributes match the given attributes as input. The 10-fold cross-validation results are given in Table. 2.

Moreover, to address RQ3, we have evaluated the approach by testing it on how many steps we can get to recommend/reuse an entire model within the repos-

[14] `https://github.com/iliriani/BORA_Ecore`
[15] `https://mvnrepository.com/`

itory. We started the reuse process from a given class within the model and designed the whole model by selecting the proposed recommendations. In addition, we have determined the reuse process based on the starting point since it can significantly impact the iteration required for getting a wholly recommended model from the repository. We assumed that if the starting point is a central class that is highly connected can retrieve much more recommendations that can be reused during the modeling process. Consequently, the model within the repository will be reused faster than starting the recommendation process by a less-connected starting point. Thus, we evaluated the required recommendations steps starting from the highest connected class in one case and the least connected class in another.

In Table. 2, we have depicted the precision/recall in each step until the entire model is recommended, i.e., we aggregated the recall through each iteration till we got a recall of 1. For this evaluation, we selected three different models within our repository. The chosen models are a model named Trip with 14 classes, a model named Expenses with 20 classes, and an Actor model with 16 classes.

6.4 Results Overview

- **Answer to RQ1:**
  Results from Evaluation methodology A:
  According to Table Table 1, the precision is very low, with a maximum of 0.02543 in Model III. This is because many classes have the same attributes, leading to many irrelevant recommendations. Another reason for the low precision value is that the class attributes were not very descriptive of the domain. The attributes of the models under construction were mostly general, such as name and date, and the classes of the model under construction did not contain many attributes. The lack of class attributes also contributes to the low recall value. Without representative class-domain attributes on both sides - the model under construction and the repository - finding relevant classes for recommendations is difficult.
  Consequently, since F1 is the harmonic function of Precision and Recall, its values are also relatively low. Additionally, because there were many domain-irrelevant recommendations, BORA's ranking is also relatively low.
  Concerning the coverage, based on the evaluation results, we can see that BORA could discover a lot of domain-related classes, with an average coverage@N of 0.89450. Although, during the evaluation methodology, we recognized that we have such a discrepancy between recall and coverage@N because the class attributes of the model under construction were not very domain descriptive. That means that most of the classes contained quite common attributes, and when looking for class recommendations from those attributes, we got mostly domain-irrelevant recommendations. On the other hand, there were only a few classes that contained domain-relevant attributes and from which BORA could discover a lot of relevant recommendations; nevertheless, even the domain-relevant recommendations were not the same as the classes of the model under construction. In conclusion, BORA could discover a lot of domain-relevant attributes for reuse, which gave a high coverage@N value. But, the recommended classes did not match any class of the model under construction; thus, we got a low recall value.
  Results from Evaluation methodology B: The evaluation results are significantly better when using real industrial models, as shown in TableTable. 2. In each iteration of the 10-fold cross-validation process, almost all metrics are higher than 0.6. Many of the test classes share many attributes with the classes from the training set, allowing the approach to extract many relevant classes for recommendations based on attribute similarities. As a result, the average precision is 0.66. The same attributes among different models also enable us to find many relevant classes within the repository for providing recommendations, resulting in a high average recall of 0.75. Since the F1 score is a harmonic function of precision and recall, we can see that the average F1 of 0.7 is also significantly high. Finally, since we have many true positives (TP), BORA's ranking performance is very high, with an average MAP of 0.71.

- **Answer to RQ2:**
  Results from Evaluation methodology A: Considering the evaluation results of BORA's recommendation performance based on class name similarity [21], with an overall average precision of 0.36517, recall of 0.35585, F1 value of 0.27585, MAP of 0.613306, and coverage@N of 0.9546[16], we can see that BORA's performance in those four models is significantly better when considering class names for modeling recommendations instead of the class attributes.
  Results from Evaluation methodology B: When considering a much larger dataset of real industrial mod-

[16] Since in the previous work we evaluated the metrics for each N-grams, and on three different models, here we outlined the average value of each metric throughout the entire evaluation table

Table 1 Evaluation process of class recommendations based on class attribute similarities

Reuse from repository evaluation - from class attributes										
Domain	Model	**Precison**		**Recall**		**F1**		**MAP**		**coverage@N**
		Avg.	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	**Avg.**	**SD**	
Trip	Model I	0.00523	0.00261	0.14286	0.05832	0.01009	0.00440	0.01194	0.00597	1.00000
	Model II	0.01556	0.01987	0.15000	0.13229	0.02717	0.03297	0.07422	0.13261	0.69565
Order	Model III	0.02543	0.02141	0.26190	0.17496	0.04635	0.03962	0.17332	0.16131	1.00000
	Model IV	0.02339	0.02187	0.12692	0.05867	0.03838	0.03391	0.11387	0.15351	0.88235
										(avg.) 0.89450

Table 2 Comparison of recommendations based on class names and attribute similarities.

Iterations	**Class names similariies**					**Attribute similarities**			
	Precision	**Recall**	**F1**	**MAP**		**Precision**	**Recall**	**F1**	**MAP**
1	0.5221	0.9363	0.6037	0.9494	**1**	0.7843	0.9096	0.8424	0.8313
2	0.5076	0.4783	0.4471	0.6214	**2**	0.8535	0.8549	0.8542	0.8683
3	0.2434	0.3217	0.2313	0.3973	**3**	0.6901	0.5733	0.6263	0.8215
4	0.3713	0.5146	0.3753	0.5625	**4**	0.7005	0.6422	0.6701	0.7345
5	0.3961	0.6705	0.4316	0.7097	**5**	0.6316	0.7410	0.6819	0.7309
6	0.2337	0.2679	0.2148	0.3279	**6**	0.6348	0.7910	0.7044	0.8417
7	0.5478	0.6637	0.5470	0.7355	**7**	0.6160	0.6939	0.6526	0.9052
8	0.6444	0.8303	0.6683	0.8630	**8**	0.6418	0.7460	0.6900	0.5677
9	0.2504	0.4333	0.2741	0.4667	**9**	0.4662	0.7407	0.5722	0.1573
10	0.3087	0.5715	0.3515	0.6047	**10**	0.6034	0.7787	0.6800	0.6549
Average	**0.4026**	**0.5688**	**0.4145**	**0.6238**		**0.6622**	**0.7471**	**0.6974**	**0.7113**

els as test/train models and performing 10-fold cross-validation, as presented in Table 2, we see that in each iteration, the metrics derived from the attribute similarities outperform those derived from class name similarities.

■ **Answer to RQ3:**
The evaluation results of the number of iterations required to completely reuse a model from the repository show that domain-specific N-grams almost always have higher precision than non-domain-specific ones. However, recall is usually higher for non-domain-specific N-grams. One crucial point outlined in Table 3 confirms the assumption that starting from the most connected class requires fewer iterations to completely recommend a model from the repository, except for the Actor model. In this model, fewer iterations are required to reuse a model from the repository when starting from the least connected class, named `ActorInboxMessage`, rather than starting from the highest connected class. The reason is that in the observed Actor model, the highest connected class, `ActorFunctionRestriction`, is not actually the most connected class among all Trip-related models. This means that there are other Actor-related models within the repository, and it was concluded that the class `Actor` - not in the observed Actor model - is much more connected than `ActorFunctionRestriction`. Thus, since the least-connected class in the observed Actor model, `ActorInboxMessage`, is connected with the `Actor` class from another Actor-related model, all Trip-related classes will be recommended directly on the next iteration.

In conclusion, BORA performs very well in reusing the knowledge of previous models in order to construct new ones, regardless of class names or class attributes will be considered. BORA also performs relatively well in supporting the citizen developers during the modeling process with domain-relevant recommendations. Finally, considering class attributes instead of class names

Table 3 Recall sum-up evaluation of BORA

Model	Relevance	Start point	n-gram	Iteration to completely get reused an model from the repository															
				1		2		3		4		5		6		7		8	
				Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall
Trip	Domain - specific	Most connected class	1-gram	0.545	1.000														
			2-gram	0.571	1.000														
			3-gram	0.550	1.000														
		Least connected class	1-gram	1.000	0.077	1.000	0.154	0.833	0.462	0.833	0.462	0.833	0.462	0.846	0.923	0.786	0.923	0.591	1.000
			2-gram	1.000	0.083	0.800	0.333	0.800	0.333	0.800	0.333	0.833	0.833	0.769	0.833	0.476	1.000		
			3-gram	0.750	0.273	0.750	0.273	0.750	0.273	0.818	0.818	0.750	0.818	0.474	1.000				
	Non - domain - specific	Most connected class	1-gram	0.481	1.000														
			2-gram	0.444	1.000														
			3-gram	0.393	1.000														
		Least connected class	1-gram	1.000	0.077	1.000	0.154	0.500	0.308	0.500	0.308	0.400	0.769	0.407	0.846	0.407	0.846	0.342	1.000
			2-gram	1.000	0.083	0.500	0.333	0.222	0.333	0.222	0.333	0.385	0.833	0.357	0.833	0.316	1.000		
			3-gram	0.429	0.273	0.176	0.273	0.176	0.273	0.400	0.909	0.345	0.909	0.289	1.000				
Expenses	Domain - specific	Most connected class	1-gram	0.800	1.000														
			2-gram	0.789	1.000														
			3-gram	0.778	1.000														
		Least connected class	1-gram	1.000	0.188	0.833	0.313	0.762	1.000										
			2-gram	0.667	0.133	0.789	1.000												
			3-gram	1.000	0.357	0.714	0.357	0.778	1.000										
	Non - domain - specific	Most connected class	1-gram	0.762	1.000														
			2-gram	0.789	1.000														
			3-gram	0.700	1.000														
		Least connected class	1-gram	0.750	0.188	0.600	0.375	0.696	1.000										
			2-gram	0.500	0.267	0.625	1.000												
			3-gram	0.833	0.357	0.833	0.357	0.833	0.357	0.667	1.000								
Actor	Domain - specific	Most connected class	1-gram	0.786	0.688	0.765	0.813	0.421	1.000										
			2-gram	0.750	0.800	0.405	1.000												
			3-gram	0.750	0.857	0.389	1.000												
		Least connected class	1-gram	0.500	0.063	0.432	1.000												
			2-gram	0.429	1.000														
			3-gram	0.333	0.071	0.500	0.143	0.389	1.000										
	Non - domain - specific	Most connected class	1-gram	0.786	0.688	0.765	0.813	0.333	1.000										
			2-gram	0.750	0.800	0.333	1.000												
			3-gram	0.765	0.929	0.318	1.000												
		Least connected class	1-gram	0.500	0.063	0.356	1.000												
			2-gram	0.349	1.000														
			3-gram	0.333	0.071	0.400	0.143	0.311	1.000										

for recommendations is more effective for providing modeling recommendations.

6.5 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 can significantly impact the validity of the evaluation results is the class attributes we selected for the first evaluation methodology. In order to mitigate this threat, we used at least three different and most relevant class attributes. By relevant, here we mean the most specific and descriptive attributes for that class. We did this by removing common class attributes like Id, Name, etc.

Another threat that could significantly validate our evaluation results is the models we considered for investigation. Considering models that may (not) be similar to other models in the repository can significantly change the evaluation results. To mitigate this threat, we selected four different models of two different domains. Moreover, we conducted another evaluation methodology by selecting 100 real industrial models and performing 10-fold cross-validation to calculate the metrics of interest.

7 Related Works

Di Rocco et al. [17] 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. Also, in [15], Di Rocco et al. present MORGAN, a GNN-based recommender system for facilitating the modeling process by assisting in 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. Compared to these approaches, BORA shares with them a graph-based repository, although BORA is search-based and uses N-grams for prediction, while MORGAN and MemoRec are ML-based. BORA can specify the relevant domain when given two different class names and produce modeling recommendations using N-grams. BORA, compared to these approaches, can determine the relevant modeling recommendation by considering the sequence of one, two, or three classes. Compared to MemoRec, BORA is not limited only to metamodel recommendations; it is level-agnostic in this context.

In [32], Weyssow et al. present an approach based on a pre-trained language model for providing metamodel concept recommendations. This approach gets lexical and structural information from metamodels, uses these information to train a deep neural language model, and provides modeling recommendations provided by this trained deep neural language model. The difference between our approach is that it relies on a knowledge graph that is able to abstract knowledge of different graph-based (meta) models and uses N-grams for producing model recommendations. However, the idea of weighting our knowledge graph with the textual embedding technique used in Weyssow et al. s. can be considered interesting future work.

Angel et al. [4,24] 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.

Compared with Extremo, which uses a data model for model persistence, BORA uses an RDF graph-based repository. Also, BORA is capable of not just querying the repository. Furthermore, it can provide recommendations based on N-grams by clicking the "get suggestions" button without requesting the user know anything about the approach. BORA is technology independent and can be used as a REST API.

Concerning model reuse, Bragilovski et al. [7] have recently published a framework that transforms models into ME-MAP [28] and uses an iterative greedy algorithm to match the user's query (query-by-example) with the models within a repository. The models will be ranked based on how much they match the model fragment given as a query by the user and will be returned as a query result.

BORA, compared to this framework, is different regarding model reuse. BORA queries the repository and uses N-grams to provide modeling recommendations steps that the user might not be aware of, whereas this framework is just query-by-example based, i.e., the user already knows what he/she is looking for.

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. The difference between Burgueño's framework is that it provides document-specific recommendations, i.e., it can suggest recommendations relevantly to the domain of the documents it received as input, whereas BORA uses a knowledge graph constructed by merging heterogeneous models and is capable of providing cross-domain recommendations. The source of the approach is also the difference between BORA and Capuano's approach. While Capuno et al. use the knowledge from code repositories to provide modeling recommendations, we use the knowledge of models.

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.

One of the differences between BORA and DoMoRe is that BORA uses the knowledge of the previously constructed model to predict recommendations, including model names, class names, class attributes, attributes names, attribute datatypes, class associations, etc. So BORA is model reuse-based. On the other hand, DoMoRe uses a large-scale graph of connected terms to predict another term. DoMoRe's repository can be integrated into BORA's repository in order to enhance the model class names recommendations.

Also, Lissete et al. [3] provide a model recommendation system for LCDPs. 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.

Even though we provide a modeling recommendations approach and Lissete a DSL for configuring modeling recommendations within an LCDP, our approach, by specifying the right paraments, can conceptually be integrated and used within this DSL,

8 Conclusion

This extended paper builds upon our previous work on BORA [21] (**B**usiness **O**bject **R**euse **A**pproach) by introducing class recommendations based on attribute similarities. Through a comprehensive survey conducted on a collection of LCDP models, we initially identified the pressing need for a model reuse approach. The survey results revealed that approximately half of the classes in these models are reused across different projects, highlighting the significance of our proposed solution.

In this paper, we have provided a more detailed explanation of BORA, shedding light on its inner workings and highlighting its key features. Notably, we have demonstrated that recommendations based on attribute similarities outperform class name-based recommendations regarding effectiveness. This finding emphasizes the importance of considering attribute similarities when suggesting class reuse to LCDP users.

Moreover, our evaluation process substantiated that BORA is an efficient tool for model reuse. By measuring the number of steps required for BORA to fully reuse a model from a repository, we established its efficacy in streamlining the reuse process. This efficiency offers tangible benefits to LCDP users, saving time and effort while promoting model reuse during the modeling process.

In summary, this extended paper contributes to advancing model reuse techniques by incorporating class recommendations based on attribute similarities into the BORA tool. By addressing the need for a model reuse approach, providing a comprehensive explanation of BORA, showcasing the superiority of attribute-based recommendations, and demonstrating its efficiency in practice, our research establishes BORA as a valuable tool for enhancing software development processes in LCDPs and fostering model reuse across projects.

Acknowledgements This project has received funding from the EU Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 813884, and by the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350.

References

1. H. Agt and R.-D. Kutsche. Automated construction of a large semantic network of related terms for domain-specific modeling. In *CAiSE*, 2013.
2. H. Agt-Rickauer, R. D. Kutsche, and H. Sack. 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):71–82, 2018.
3. L. Almonte. Towards automating the construction of recommender systems for model-driven engineering. 2020.
4. M. S. Ángel, J. de Lara, P. Neubauer, and M. Wimmer. Automated modelling assistance by integrating heterogeneous information sources. *Computer Languages, Systems and Structures*, 2018.
5. S. M. Beitzel, E. C. Jensen, and O. Frieder. *MAP*, pages 1691–1692. Springer US, Boston, MA, 2009.
6. A. Bhattacharyya and D. Chakravarty. (Graph Database: A Survey). *2020 International Conference on Computer, Electrical and Communication Engineering, ICCECE 2020*, 2020.
7. M. Bragilovski, R. Stern, and A. Sturm. How do I find reusable models? *Software and Systems Modeling*, 2023.
8. A. Bucaioni, A. Cicchetti, and F. Ciccozzi. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, (Lcd), 2022.
9. L. Burgueño, R. Clarisó, S. Li, S. Gérard, J. Cabot, L. Burgueño, R. Clarisó, S. Li, S. Gérard, and J. C. A. Nlp-based. 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. T. Capuano, H. Sahraoui, B. Frenay, and B. Vanderose. Learning from code repositories to recommend model classes. *Journal of Object Technology*, 21(3):3:1–11, July 2022. The 18th European Conference on Modelling Foundations and Applications (ECMFA 2022).
11. Y. Chen. Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data. page 82, 2016.
12. D. M. Christopher and S. Hinrich. Foundations of statistical natural language processing. 1999.
13. P. Cudré-Mauroux and S. Elnikety. Graph data management systems for new application domains. *Proceedings of the VLDB Endowment*, 4(12):1510–1511, 2011.
14. J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and R. Rubei. Development of recommendation systems for software engineering: the CROSSMINER experience. *Empirical Software Engineering*, 26(4), 2021.
15. J. Di Rocco, C. Di Sipio, D. Di Ruscio, and P. T. Nguyen. 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)*, pages 70–81, 2021.
16. D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 2022.
17. J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and A. Pierantonio. MemoRec: a recommender system for assisting modelers in specifying metamodels. *Software and Systems Modeling*, 2022.
18. B. DuCharme. *Learning SPARQL (free chapters)*. 2013.
19. A. Grzech, L. Borzemski, J. Świątek, and Z. Wilimowska. Preface. *Advances in Intelligent Systems and Computing*, 430(October):V–vi, 2016.
20. I. Ibrahimi and D. Moudilos. Model slicing on low-code platforms. In C. Dubois and J. Cohen, editors, *STAF 2022 Workshop Proceedings: 2nd International Workshop on Foundations and Practice of Visual Modeling (FPVM 2022), Nantes, France, July 5-8, 2022*, volume 3250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.
21. I. Ibrahimi and D. Moudilos. Towards model reuse in low-code development platforms based on knowledge graphs. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, page 826–836, New York, NY, USA, 2022. ACM.
22. A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger. Reuse in model-to-model transformation languages: are we there yet? *Software and Systems Modeling*, 2015.
23. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

24. Á. Mora Segura and J. de Lara. EXTREMO: An Eclipse plugin for modelling and meta-modelling assistance. *Science of Computer Programming*, 2019.
25. G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean, J. M. Jézéquel, T. Kühn, S. Mosser, H. Sahraoui, E. Syriani, D. Varró, and M. Weyssow. Opportunities in intelligent modeling assistance. *Software and Systems Modeling*, (June), 2020.
26. W. Read, T. Report, and K. Takeaways. The Forrester Wave ™ : Low-Code Development. 2016.
27. A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio. Supporting the understanding and comparison of low-code development platforms. *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, (August):171–178, 2020.
28. A. Sturm, D. Gross, J. Wang, and E. Yu. Means-ends based know-how mapping. *Journal of Knowledge Management*, 21(2):454–473, 2017.
29. M. Tisi, J.-M. Mottu, D. S. Kolovos, J. De Lara, E. M. Guerra, D. Di Ruscio, A. Pierantonio, and M. Wimmer. 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*, CEUR Workshop Proceedings (CEUR-WS.org), Eindhoven, Netherlands, July 2019.
30. P. Vincent, K. Iijima, M. Driver, J. Wong, and Y. Natis. Licensed for Distribution Magic Quadrant for Enterprise Low-Code Application Platforms. pages 1–34, 2019.
31. R. Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019.
32. M. Weyssow, H. Sahraoui, and E. Syriani. Recommending Metamodel Concepts during Modeling Activities with Pre- Recommending Metamodel Concepts during Modeling Activities with Pre-Trained Language Models. 2021.