"This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884"

Project Number: 813884

Project Acronym: Lowcomote

Project title: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms

Components for Mining Low-Code Engineering Repositories

Project GA: 813884

Project Acronym: Lowcomote

Project website: https://www.lowcomote.eu/

Project officer: Thomas Vyzikas

Work Package: WP4

Deliverable number: D4.5

Production date: 28 December 2022

Contractual date of delivery: 31 October 2022

Actual date of delivery: 28 December 2022

Dissemination level: Public

Lead beneficiary: Johannes Kepler University Linz

Authors: MohammadHadi Dehghani, Ilirian Ibrahimi

Contributors: Luca Berardinelli

Project Abstract

Low-code development platforms (LCDP) are software development platforms on the Cloud, provided through a Platform-as-a-Service model, which allows users to build completely operational applications by interacting through dynamic graphical user interfaces, visual diagrams, and declarative languages. They address the need of non-programmers to develop personalized software and focus on their domain expertise instead of implementation requirements.

Lowcomote will train a generation of experts that will upgrade the current trend of LCPDs to a new paradigm, Low-code Engineering Platforms (LCEPs). LCEPs will be open, allowing to integrate heterogeneous engineering tools, interoperable, allowing for cross-platform engineering, scalable, supporting very large engineering models and social networks of developers, smart, simplifying the development for citizen developers by machine learning and recommendation techniques. This will be achieved by injecting in LCDPs the theoretical and technical framework defined by recent research in Model Driven Engineering (MDE), augmented with Cloud Computing and Machine Learning techniques. This is possible today thanks to recent breakthroughs in the scalability of MDE performed in the EC FP7 research project MONDO, led by Lowcomote partners.

The 48-month Lowcomote project will train the first European generation of skilled professionals in LCEPs. The 15 future scientists will benefit from an original training and research programme merging competencies and knowledge from 5 highly recognized academic institutions and 9 large and small industries of several domains. Co-supervision from both sectors is a promising process to facilitate the agility of our future professionals between the academic and industrial worlds.

Deliverable Abstract

This report presents a collection of four different Low-code mining components that aim to facilitate the modeling processing on Low-codes. The first two are model state-based mining components, and the third is a modeling process-based mining component. This report initially presents a model reuse approach that facilitated the modeling process on a Low-code platform by providing modeling recommendations derived from a graph-based repository tailored for the actual state of the model under construction. Subsequently, this report presents another approach that enables model reuse based on natural language processing (NLP) and a graph-based repository. It aims to reuse Low-code models or model classes by requesting them in natural language. Then, the report presents a process mining as a service component for LCDPs based on recorded modeling events generated by users of graphical model editors. Finally, the deliverable presents the ongoing work for a modularization approach for monolithic data models.

Contents

1 Introduction

1.1 Context

Model-Driven Engineering (MDE) is a method in software engineering that focuses on (semi-)automatic development of software artifacts from high-level models [1]. This paradigm relies on using Domain-Specific Modelling Languages (DSMLs) to build such models using domain concepts. As a result, the implementation effort is reduced because the developer is isolated from the implementation details.

MDE methodology can be used to build platforms that produce actual software, such as Low-code Development Platforms (LCDPs). For instance, tools such as Google AppSheet[1], Amazon Honeycode[2], among others [2] are becoming popular given their advantage for rapid development of full-stack applications without requiring prior programming knowledge. Therefore, some of these platforms target the so-called *citizen developers* with no programming skills. Typically, LCDPs provide a user-friendly interface (e.g., graphical editors to describe the different parts of the application) and integration with third-party systems and cloud development facilities. Due to Gartner [3], in 2024, low-code platforms will be responsible for most application development activities.

The EU project *Lowcomote* aims to enhance LCDPs by transferring knowledge from MDE to improve low-code development and reach a more systematic low-code engineering. The main Research Objectives (ROs) of Lowcomote are:

- **RO1**: Enabling *Low-code Engineering of Large-Scale Heterogeneous Systems*, by smart development environments on the Cloud and precise integration of low-code languages with new domains.
- **RO2:** Developing a *Large-scale Repository and Services for Low-Code Engineering*, as a Cloud-based service able to handle a huge number of low-code artifacts, and automatically learn from them.
- **RO3:** Producing advancements in *Scalable Low-Code Artifact Management*, as new algorithms and reusable components.

This deliverable tackles RO2, which focuses on defining approaches for mining low-code engineering repositories to learn from existing models and interactions. The deliverable presents four contributions:

- An extension and update of our previous deliverable (D4.2) that enabled model recommendations by initially converting and merging heterogeneous models into a homogeneous graph, queries the repository for relevant matches and uses N-grams to produce recommendations for models under development. In this deliverable, to emphasize the emerging need for a model reuse approach, we have extended our previous approach by initially outlining a comparative study on reused models within different zAppDev models and expanding the recommendation scope by considering class-attributes similarities.
- Furthermore, in the context of learning from a knowledge graph and model reuse, this deliverable also presents an approach that supports the LCDP users during the modeling process by enabling them to reuse the information of previous models. The approach persists heterogeneous models in a knowledge graph, queries this graph using a natural language (NL) description, and returns the relevant information to the user.
- Then, this deliverable presents a process mining component that supports desktop and cloud-based LCDPs. It provides the ability to capture the user interaction events in a log file that can be later used to mine a process model, which is then used for many purposes, such as operation-based recommendations to the users and providing better user interfaces to users based on their observed behavior.
- Finally, this deliverable presents an extension of an existing modularization approach for monolithic Entity-Relationships (ER) data models [4] by proposing a formulation of the modularization problem into a reinforcement learning (RL) one.

1.2 Problem Statement

The role of modeling in an LCDP is crucial for developing the application. Models created by citizen developers are valuable assets for any LCPD, as well as the experience gained by citizen developers while creating such models. For these reasons, model reuse for providing assistance during the modeling process or when starting the modeling process from scratch, modeling process mining, and the quality of the created models are problems whose solution can promote the adoption of LCPDs and their sustainable adoption based on reusable models and modelers' experience as valuable assets.

On Model Reuse when modeling from scratch. Modeling from scratch can be difficult for novice LCDP users since it requires modeling experience on the one hand and domain-relevant knowledge on the other.

[1] https://about.appsheet.com/home/.

[2] https://www.honeycode.aws/.

Secondly, a non-well-structured model can lead to errors for upcoming automated steps. For instance, if the domain model contains errors, then the UI generated from the domain model and the DSL code will be prone to errors. And finally, even though modeling is easier than coding, re-inventing models from scratch can be time-consuming even for experienced modelers. Therefore, facilities that enable modeling support when starting from scratch by simply processing users' requests provided in their own natural language are highly demanded.

On Model Reuse during the modeling process. When modeling a domain on an LCDP, we can see many projects with similar domain models, and many model elements are the same among these various projects. For instance, retail systems tend to share model elements related to managing customers, products, orders, and payments). All these shared domain models or model elements are wastefully re-invented from scratch without effective discovery and reuse mechanisms. This can hamper both productivity and feature completeness. As such, facilities for automated discovery and recommendation of relevant models and model elements through semantic analysis of models of other low-code systems are much desired.

On Modeling Process Mining. Usually, LCDPs generally lack mechanisms to collect the modeling activity performed by the modelers. The logging information of the user's modeling behavior deserves appropriate mechanisms (e.g., modeling event generators and listeners) and format to reuse the logged information for further processing conveniently (e.g., model recommendations).

On Modularization of Data Models. We expect data models (e.g., entity-relationship models) to be defined by citizen developers. A monolithic data model can hinder the development of LCDP-based apps consuming such data. A modularization approach can help citizen developers deliver higher-quality data models and LCDP-based apps.

1.3 Contribution

The main contributions of this deliverable are:

- **Contribution on Model Reuse when modeling from scratch.** To facilitate the modeling process on LCDP, we propose two different services for model reuse. One service aims to support the citizen developer when they start modeling from scratch, where they can request what kind of model or model element they want in their natural language. If this model or model element exists in the repository, it will be given to the citizen developer.
- **Contribution on Model Reuse during the modeling process.** The other proposed service aims to support the citizen developer during the modeling process. Suppose the citizen developers want to get support for the subsequent modeling step. In that case, they can trigger our proposed service, which takes as input the relevant classes (or class attributes) from the domain model, and by using a knowledge graph and N-grams, it predicts the next modeling step.
- **Application of Operation-Based Mining Approach on Sirius-based and cloud-based graphical editors:** The approach for capturing, recording, and using operational data generated by system executions as process models presented in D4.2 are refined and applied to two different editors, namely the CAEX Modeling Workbench, an AutomationML [3] editor based on Eclipse EMF and Sirius, and the Ecore editor provided by Eclipse Theia.
- **Reinforcement Learning-based approach for modularization of monolithic data models**: We present the ongoing work on extending an existing modularization approach for monolithic data models based on genetic algorithms with reinforcement learning capabilities.

1.4 Structure of the Deliverable

The remainder of this deliverable is structured as follows. Section 2 describes model reuse on LCDP based on knowledge graphs. Section 3 presents model reuse on LCDP based on NLP algorithms. Section 4 introduces process mining as a service, while Section 5 presents the RL-based modularization approach for monolithic data models. Finally, Section 6 concludes the deliverable.

2 Model Reuse for LCDPs based on Knowledge-Graphs

Context: Low-code Development Platforms (LCDP) are applications to provide fast full-stack application development. These platforms rely on models as their core artifacts to reduce software development's complexity. Despite the growing need and importance of using models in LCDPs, 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

[3] www.automationml.org

repository equipped with an efficient query mechanism combined with algorithms for processing the retrieved data.

Contribution: In this Section, we present an extension of our previous deliverable (D4.3) that enabled model recommendations by initially converting and merging heterogeneous models into a homogeneous graph, queries the repository for relevant matches, and using N-grams to produce recommendations for models under development. In this work, to emphasize the emerging need for a model reuse approach, we have extended our previous approach by initially outlining a comparative study on reused models within different zAppDev models and expanding the recommendation scope by considering class-attributes similarities.

Evaluation: We evaluated our approach by reconstructing four different models that do not exist in our repository with the support of our approach. The evaluation is conducted for the class as well as attributes similarities. The current approach is demonstrated on the zAppDev LCDP, but conceptually it can be integrated into any LCDP and applied to reuse any graph-based model.

2.1 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 [5, 6]. LCDPs 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 [2, 7]. 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 [8, 9]. According to Gartner [3], in 2024, LCDPs will be responsible for most 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) [10]. 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, in this Section, we present an extension of our previous work[1] of 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 using *N-grams* to predict the next modeling step. Moreover, in this work, we initially present the emerging need for a model reuse approach from a study conducted on models used in the zAppDev LCDP to see how many classes are reused among different models.

In this work, we also extended our previous approach by considering class attributes for getting similar classes from the repository and providing modeling recommendations.

The novelty of the approach provided in this Section is twofold; first, the study conducted on LCDP models, which results outline the need for a model reuse approach. And the second contribution is the approach that facilitates model reuse by using a graph-based repository that can conceptually persist different level and language-agnostic models and provide model recommendations using N-grams [11]. The approach considers class names and class attributes for modeling recommendations.

The current approach is integrated into the zAppDev[4] LCDP and is used to reuse different models within the zAppDev LCDP.

The Section is organized as follows: we will start by outlining the importance of a model reuse approach. For this, in Subsection ,2.2 we will provide some background information relevant to the approach we have developed, followed by a motivational example. In Subsection 2.3, we outline a study about reused classes within some zAppDev models and the need for a model reuse approach. Afterward, in Subsection ,2.5 we will explain the approach on how it persists models and how it provides recommendations. Following this, in Subsection 2.6, we will outline the tool support for our approach. In Subsection 2.7, we will depict the evaluation results of the approach and will explain them. Lastly, in Subsection 2.8, we will provide some related works to our approach, and in Subsection 2.9, we will present the conclusion of this work.

2.2 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,

[4]https://zappdev.com

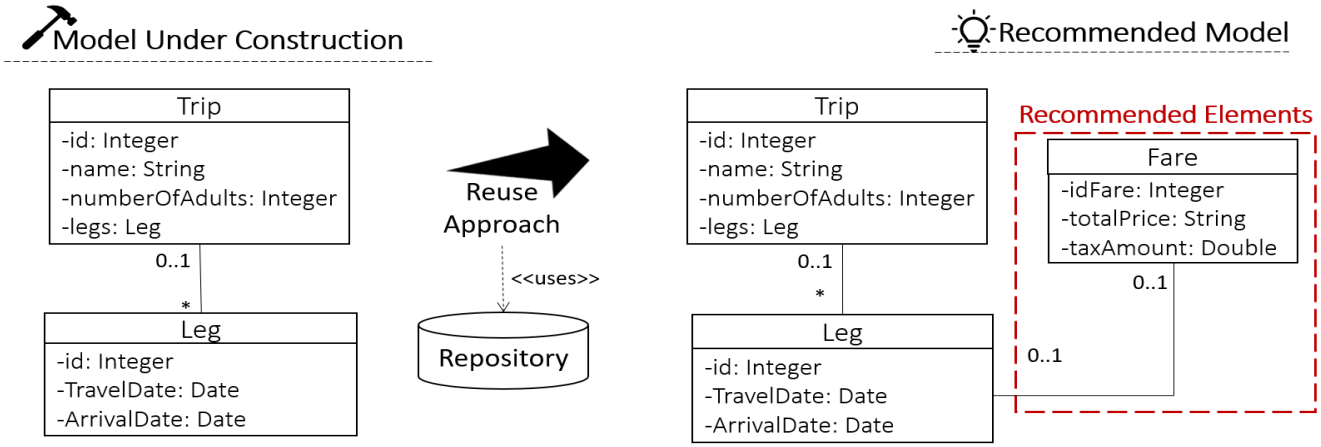

Figure 1: Running Example: Trip

Table 1: Reused classes among different zAppDev models

Total models	Total distinct classes	Connected distinct classes	Total classes among models	Repeated cross-model classes
667	1347	977	2652	2295

our model should contain a ”Trip” class with 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, like the model presented in Fig. 2. By comparing the state of the model under construction with a 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 also be 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 of persisting models in a repository and reusing the information they contain to support the modeler with useful modeling recommendations during the modeling process.

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

2.3 On the Need for a Model Reuse Approach

In this section, we will explain the importance of a model reuse approach based on a study conducted on some zAppDev models.

The need for a model reuse approach was evident after we discovered some zAppDev models. As depicted in Fig. 4, which presents a snapshot of a zAppDev model named AccessComponent, we can see that from the six classes shown here, four classes are reused from other models. For instance, the class ActorRestrictionsInitialValue is copied from the Actor model, MRFMessage class is copied from the MRFMessage model, etc. The only classes created directly in this [snapshot of the] model are the classes AccessComponent and DocumentReference. Hence, we studied how many classes are reused among some given zAppDev models. The results are presented in Table 1.

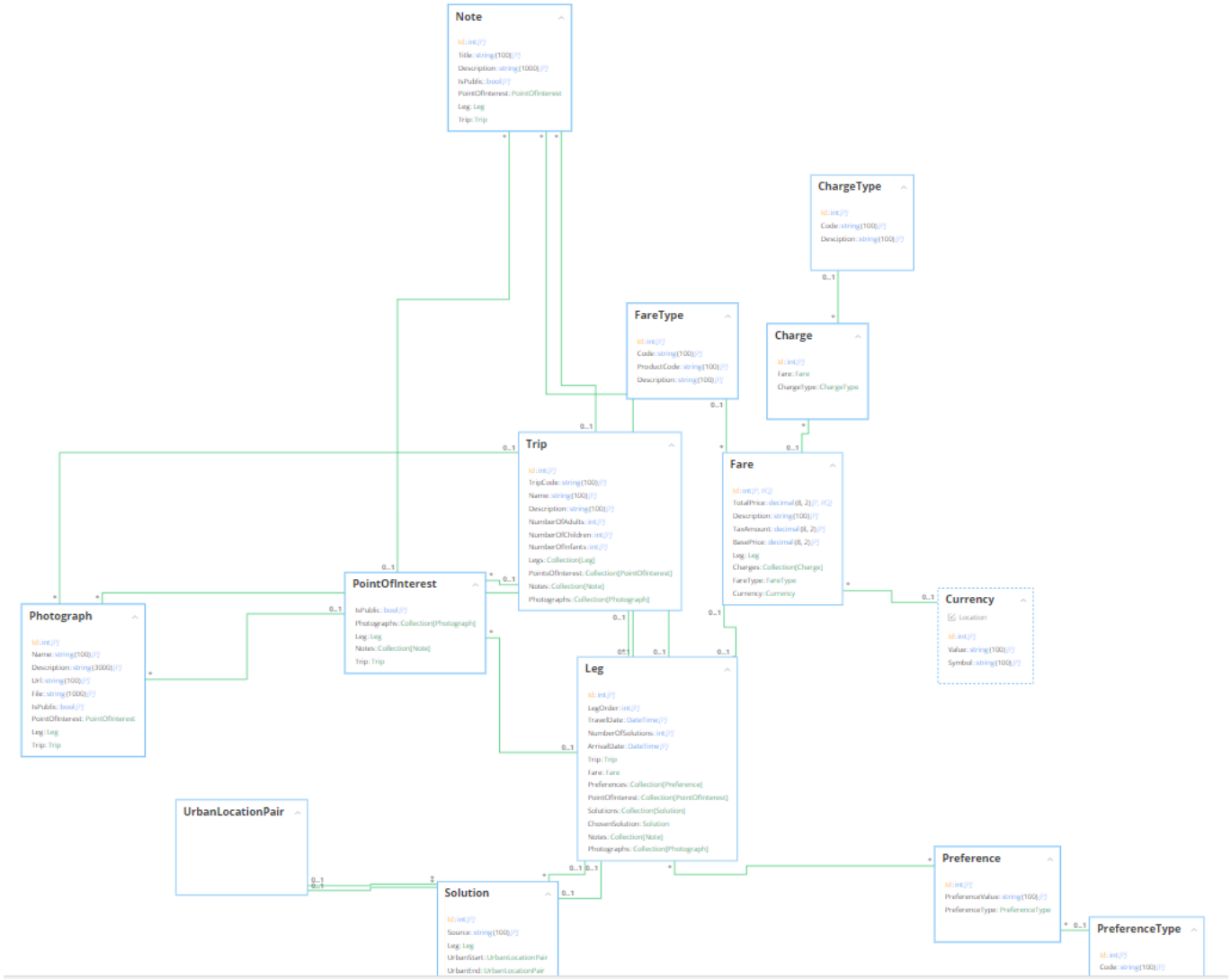

Figure 2: A Trip model within the repository.

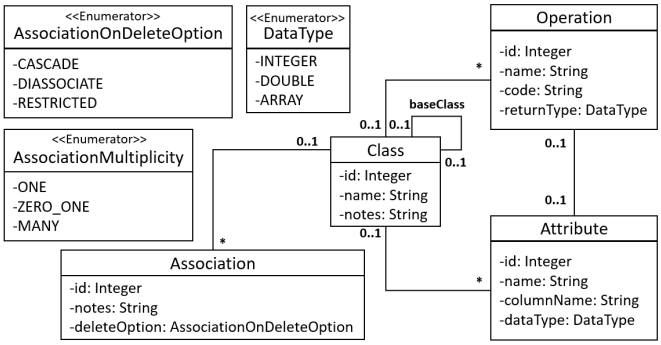

Figure 3: BO metamodel notations

We researched 667 different zAppDev models used in real industrial projects. We initially found that the total number of distinct classes used within these models is 1357. When we searched for the total number of classes within these models, we found 2652 classes. Concretely, this number represents how often the tag ¡Class¿ has been used within these zAppDev models, where ¡Class¿ represents a class within a zAppDev model. That means that there were many repeated (reused) classes among different models. Thus, we researched how many classes are reused among these models. By counting all classes whose source model was different from the model being discovered (as depicted in Fig. 4), we found that there are 2295 repeated (or reused) classes within the given dataset. So, in this research, we found that 87% of the classes are reused within different models, as also depicted in Fig. 5. And finally, as shown in Fig. 6, we also found that around 72% of the classes (977 classes) are connected to any other class. In conclusion, we can see that a huge number of classes are reused within an LCDP, which shows the need for a model reuse

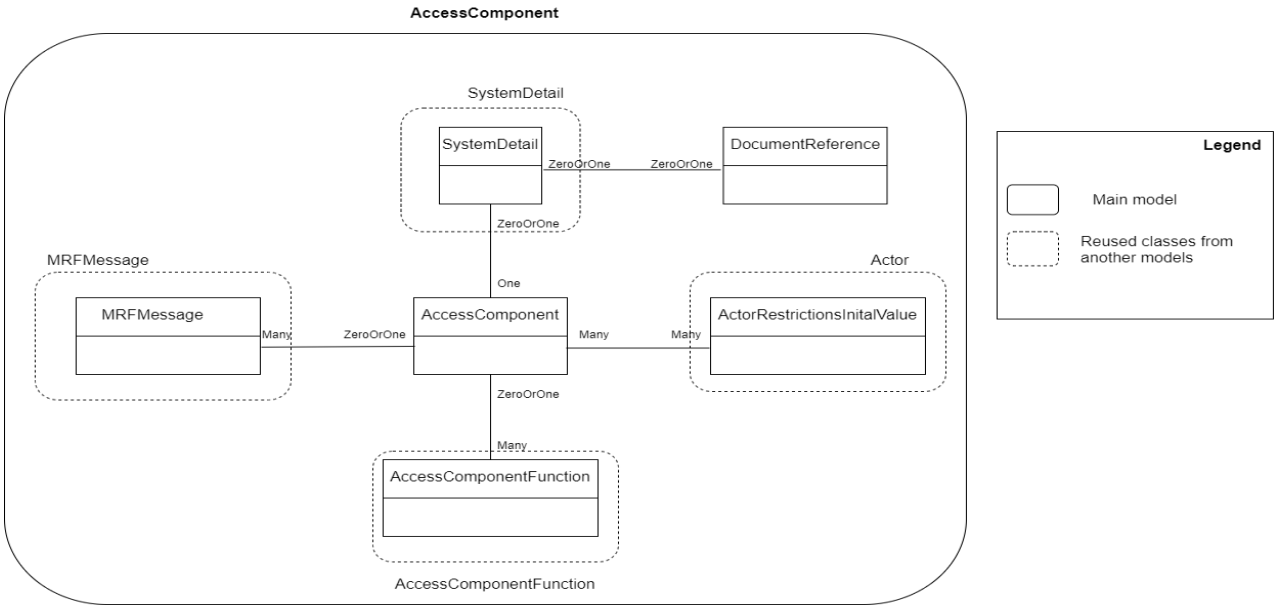

Figure 4: Reused classes within the AccessComponent model

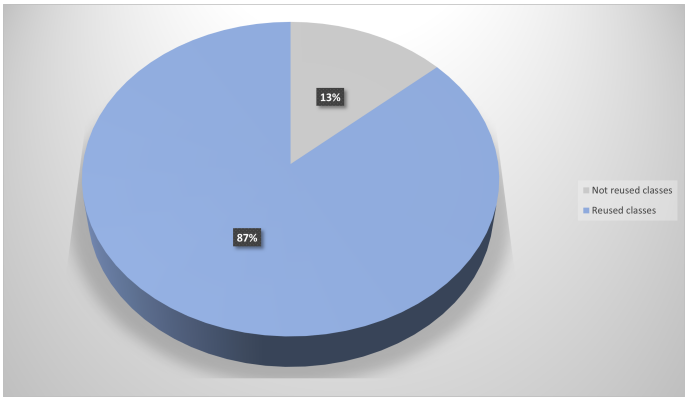

Figure 5: Reused classes within the zAppDev model

approach. Also, since the research showed that most classes are connected, the reuse approach should probably focus mostly on class connection.

2.4 Background Information

2.4.1 Data Storage

To provide a more accurate and relevant model reuse approach, the need for more data is necessary. As much data is available, it increases the possibility of finding something worth reusing when creating new

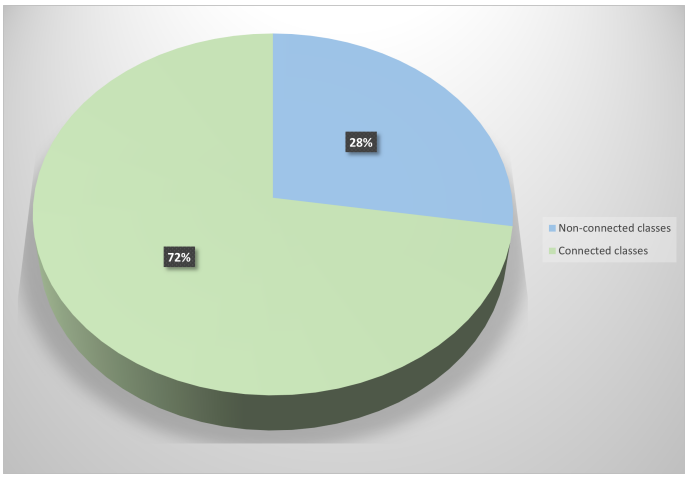

Figure 6: Connected classes within the zAppDev model

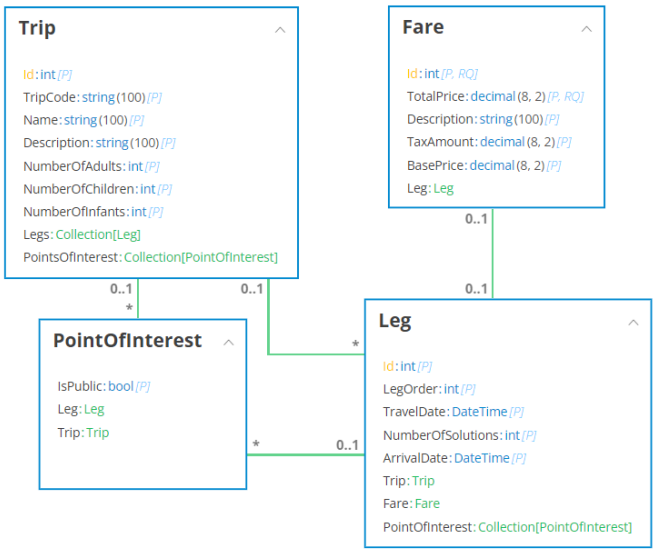

Figure 7: An excerpt of the BO model named "Trip"

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 it be discovered best? Consequently, scalability is a very important issue.

Due to [12] 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. [13] 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 [14], since it has to scale[5] when using at least thousands of models, and since it has to be a good fit for the web [15] (because low-code platforms are cloud-based) we selected RDF (Resource Description Framework) since it is de facto the W3C standard since 1999[6]. 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 [16], the standard query language for data represented in a subject, predicate, object triple format (and even more). And finally, we selected to use TDB[7] - the RDF tailored-made repository as a repository backend for our approach.

2.4.2 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. 3 depicts the notations of the zAppDev BO metamodel. And in Fig. ,7 we depict an excerpt of a zAppDev model called "Trip".

2.5 Approach

This Section explains in detail the proposed model reuse approach.

2.5.1 Process diagram of the approach

In Fig. 8, we presented the process diagram of our proposed model reuse approach. After the citizen developer starts modeling on an LCDP, she can ask for modeling support on the UI. The respective UI component will trigger the recommendation API and provide it with the relevant information on the model under construction, i.e., classes and class attributes. Having the list of the classes and attributes,

[5]https://www.w3.org/wiki/LargeTripleStores
[6]https://www.w3.org/TR/2004/REC-rdf-concepts-20040210
[7]https://jena.apache.org/documentation/tdb

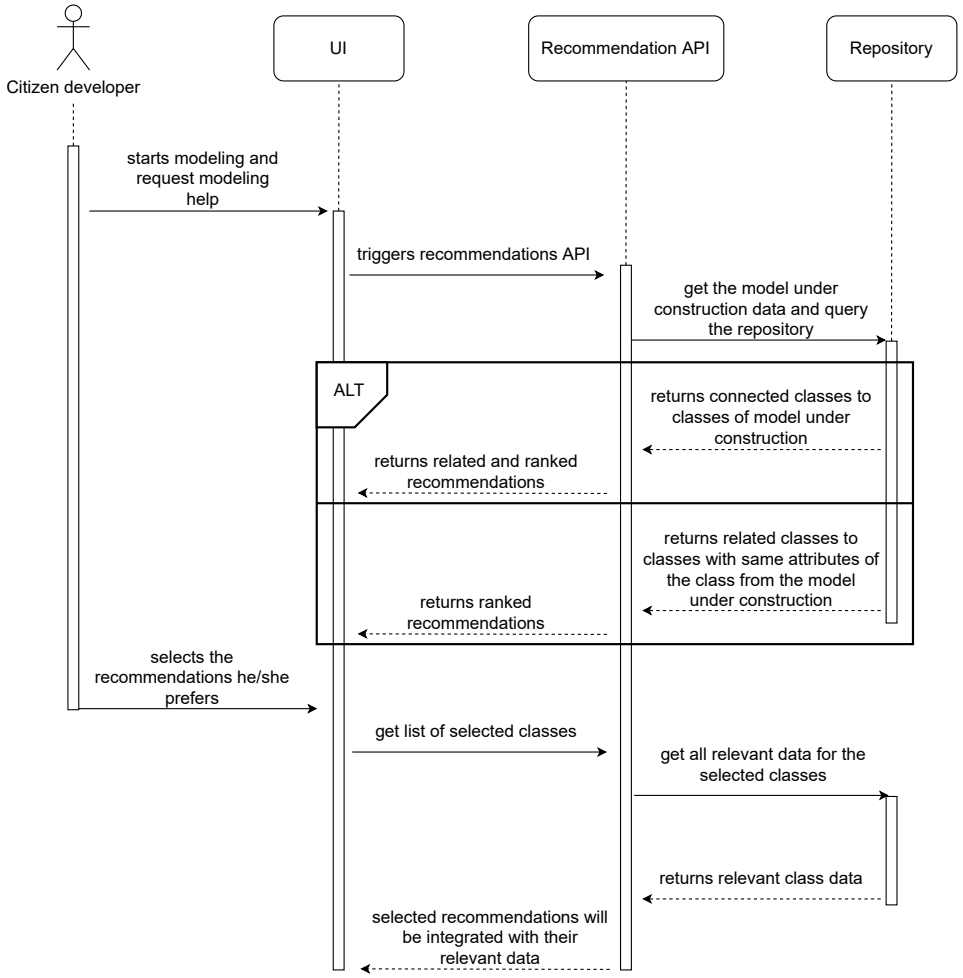

Figure 8: Process diagram of the proposed model reuse approach

the recommendation API will query the repository for classes that are connected to the classes of the model under construction. The connected classes will be returned to the recommendation API, which will be processed and ranked. In case no connected classes can be found on the repository, the approach checks for classes having the same attributes as the class of the model under construction from which the recommendation service has been triggered. If there is such a class with the same attributes, then the approach checks its connected classes and returns these 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 as the next step during the modeling process. The user-selected classes will trigger another process which will get as input the list of user-selected classes and queries for the repository for all relevant information for them, i.e., their attributes, attribute data types, connection types, etc. Finally, all the user-selected classes and their respective information will be integrated directly into the modeling canvas of the LCDP.

Next, we will explain in more detail how the recommendation API works and how it uses the information of previous models in order to provide recommendations for the next modeling steps

2.5.2 Model Reuse Approach

In Fig. 9 are presented the steps our proposed approach takes to realize model discovery and reuse. As described in [17], the model reuse cycle consists 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. ,9 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 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

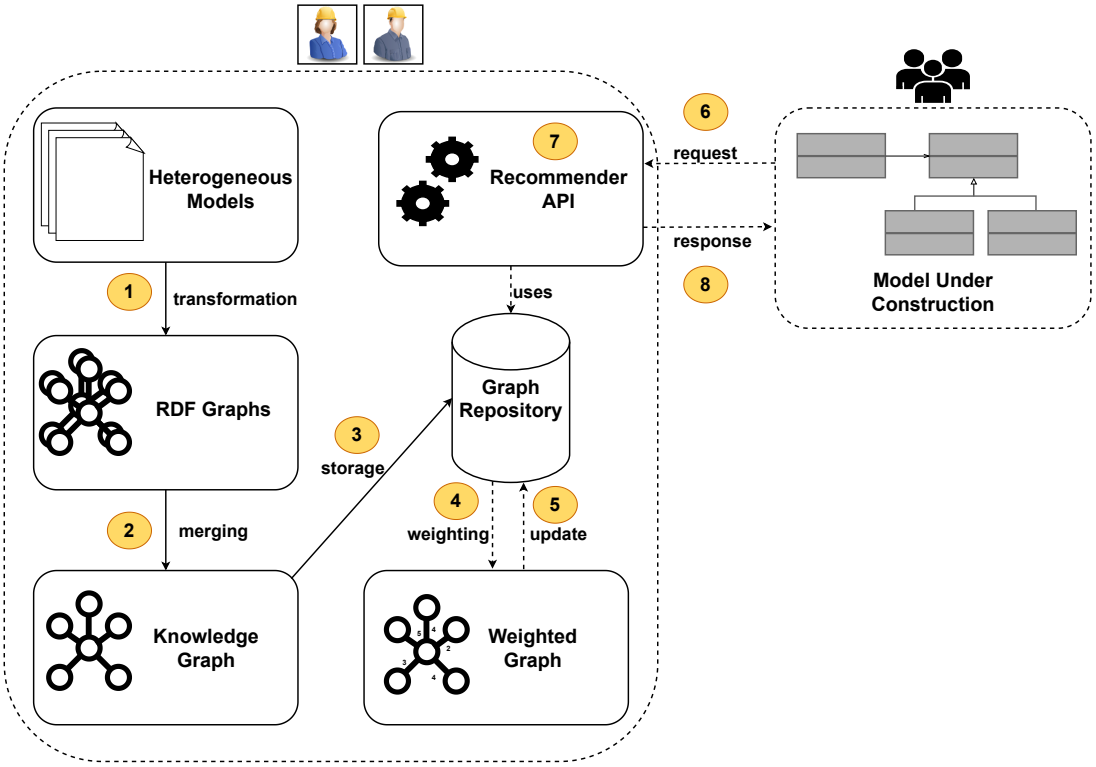

Figure 9: Overview of the model reuse approach

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. 9. 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. 9, this step is represented as step 5.

To have a better understanding of the format of abstraction, we have outlined the smallest draft of the knowledge base in Fig 10 which also emphasizes the most relevant parts for model reuse.

In conclusion, RDF serves as the abstraction format for our approach. So far, zAppDev models, Ecore models, and MoDisco reverse-engineered models [18] 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. 9 step 6, the "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 here 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. [19] 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

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.w3.org/TR/html4/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:j.0="http://acandonorway.github.com/XmlToRdf/ontology.ttl#">
  <BusinessObject>
    <Version></Version>
    <Model_Name>Notebook</Model_Name>
    <Model_Description></Model_Description>
        ......
         <j.0:hasChild>
      <Classes>
        <j.0:hasChild>
          <Class>
                <Name> CLASS NAME </Name>
                ......
            <j.0:hasChild>
              <Attributes>
                <j.0:hasChild>
                  <Attribute>
                    <Name> ATTRIBUTE NAME </Name>
                          .....
                  </Attribute>
                  </j.0:hasChild>
              </Attributes>
          </j.0:hasChild>
          </Class>
          .....
        </j.0:hasChild>
    <Associations>
        </j.0:hasChild>
          <Associatiion>
            <j.0:hasChild>
                ......

                <Class1> NAME OF CLASS 1 </Class1>
                <Class2> NAME OF CLASS 2 </Class2>
          </j.0:hasChild>
          </Association>
          .....
        </j.0:hasChild>
    </Associations>

    </BusinessObject>
</rdf:RDF>
```

Figure 10: An excerpt of the knowledge graph

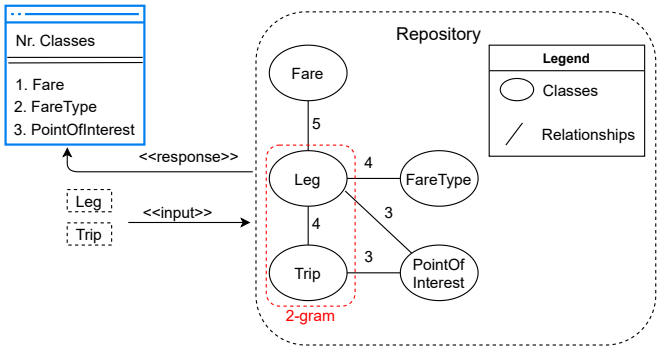

Figure 11: Extracting N-grams from the repository

from a given class ”Class1” (step 6 in Fig. 9), 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. 11. 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, the same if the approach is 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 is also the model attributes (also shown in Fig. 1). The approach is also capable of finding relevant model attributes

for a given class. First, it queries the repository for the class in which 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. ,9 the specialization process happens in step 7.

Integration: After the approach has determined possible model elements that can be recommended and reused based 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. ,9 the integration part is presented in step 8.

In section ,2.6 we will explain more about the integration process and tool support.

2.5.3 Class Recommendations from attributes

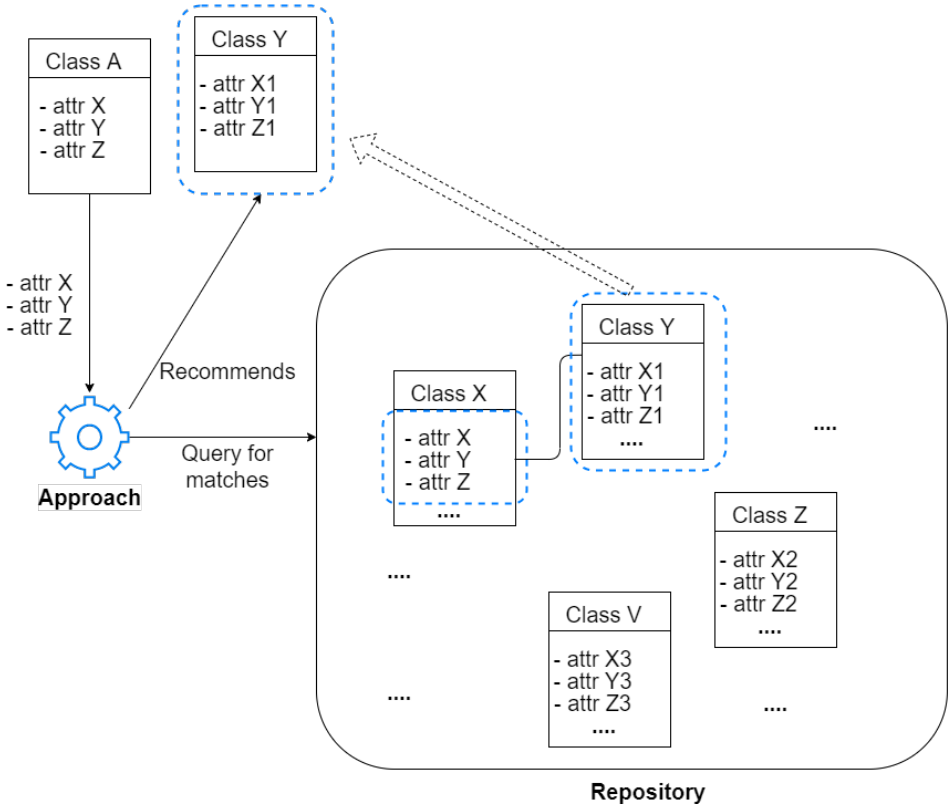

Figure 12: Class recommendations from attributes

The approach uses the attributes of the classes of the model under construction to provide recommendations about the next modeling step. As briefly explained in Fig. 8, the approach also uses class attributes for providing recommendations. If the approach cannot provide any recommendation based on N-grams, i.e., on the repository, no related classes could be found to the classes of the model under construction, then the approach starts using the class attributes for providing recommendations for the modeling step. The idea is that if there is nothing in the repository related to the class of the model under construction, maybe there is something related to any class that is similar to that class of the model under construction. The similarity is measured by the number of same attributes different classes have, i.e., the more attribute in common, the more similar the two classes are.

As outlined in more detail in Fig. 12, initially, the approach gets as input the list of attributes of the class from which the recommendation service has been triggered, e.g., attr X, attr Y, and attr Z. Then, it queries the entire repository to get classes that contain these attributes. Since we have constructed the query so that it matches any class that contains attr X OR attr Y OR attr Z, etc., it returns the matched classes ranked based on the number of attributes they contain, i.e., classes from the repository that has more attributes in common with the class from the model under construction will be ranked first. Afterward, the approach checks for each of the matched classes from the repository if they have connected classes. For instance, as depicted in Fig.8, Class X contains the attributes X, Y, and Z. Now, the approach checks for related classes to Class X, which in our case would be Class Y. Then, all the connected classes will be listed and ranked

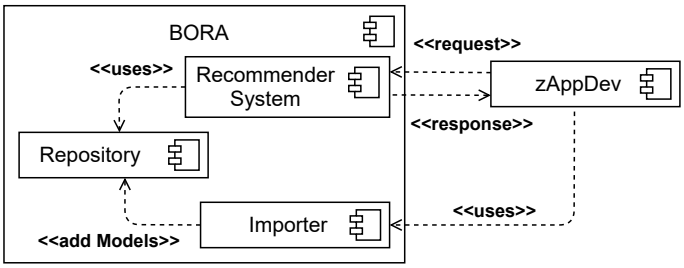

Figure 13: The architecture of BORA's integration into zAppDev.

based on their term frequencies. Finally, this list of classes will be recommended to the user as a suggestion for the next modeling step. In the case of Fig. 12, Class Y would be recommended after Class A.

2.5.3.1 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[8] synonyms.

2.6 Tool Support

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

As depicted in Fig. 13, zAppDev access BORA's functionalities by triggering any of its endpoints, e.g., one gram (String entity), onegramdomainrelevant(String cl, String domain), two gram(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 then be 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 afterward, 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. 14 and in Fig. 15, they have just 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 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 will the classes be integrated, but also the respective class attributes, connection type, connection name, and 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. 14. 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 pressed the "OK" button, the selected classes will be integrated into the modeling canvas with all the relevant 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, a 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.

2.7 Evaluation

In this section, we will explain 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

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

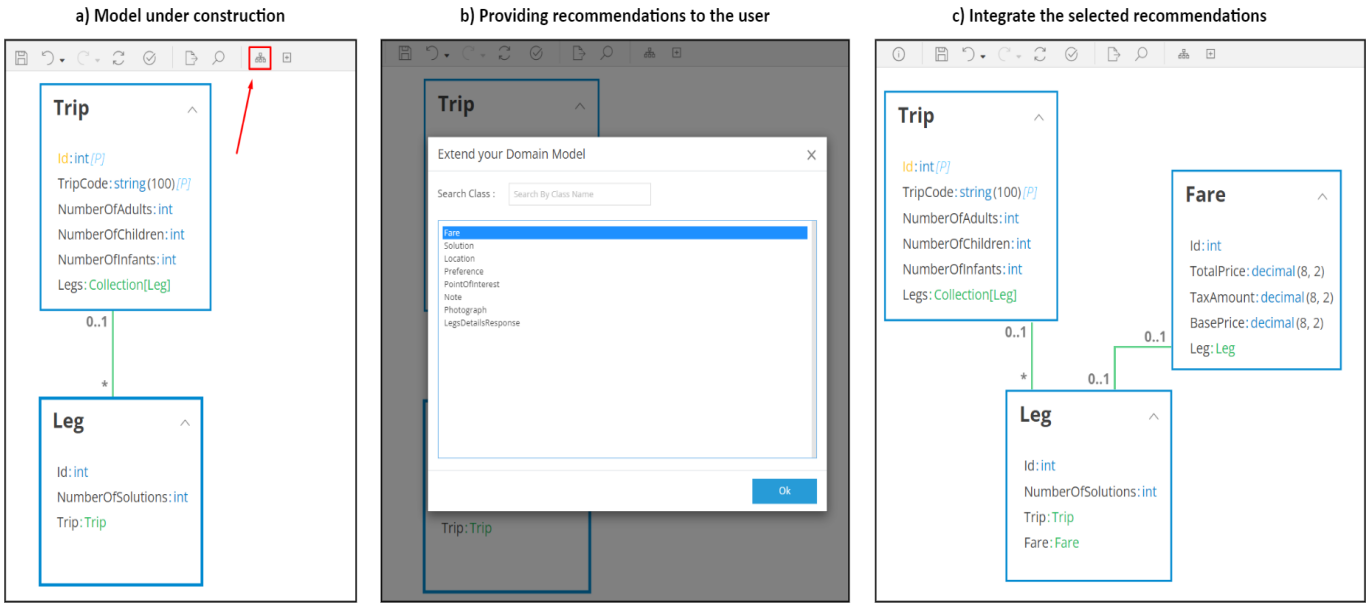

Figure 14: A screenshot of BORA's class recommendation performance on zAppDev

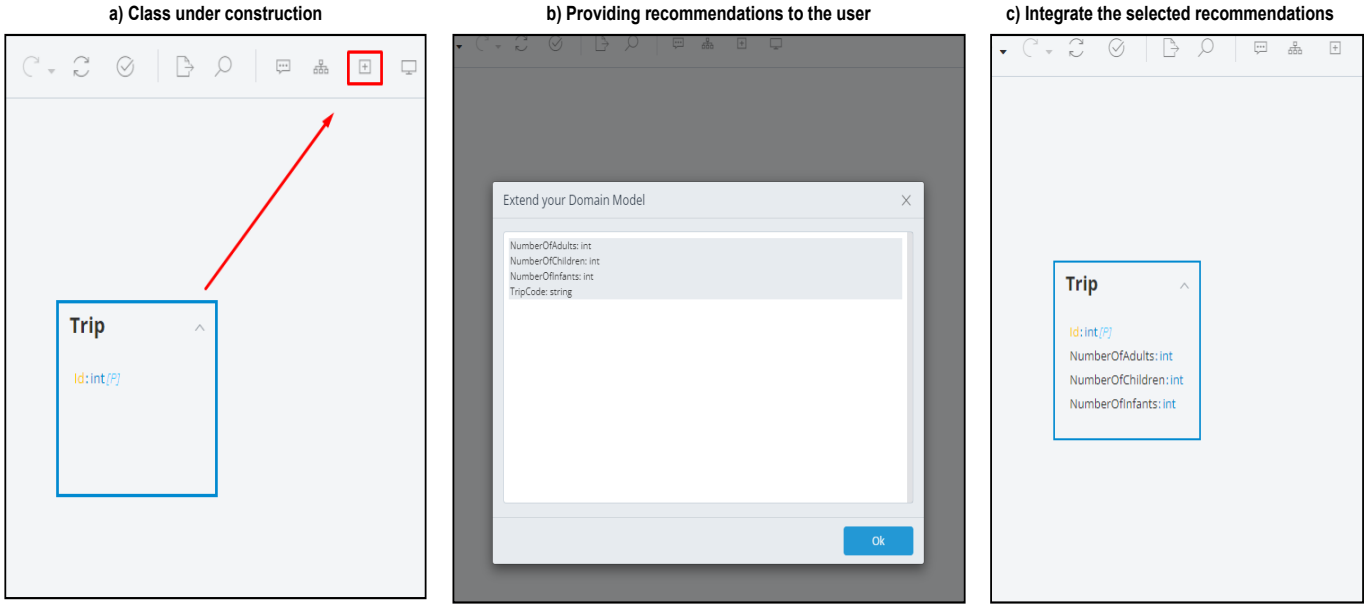

Figure 15: A screenshot of BORA's attribute performance on zAppDev

evaluation results, and finally, we will explain the results retrieved by our evaluation process.

2.7.1 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 in recommending model classes during the model process of new models?
   - How effective is 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 in covering domain-related model classes from the repository in order to provide recommendations?

2.7.2 Evaluation Methodology

2.7.2.1 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 [21] (since precision and recall have the same value of 0.5, we will be using the F1-score). For evaluation of the ranking of the approach, we used MAP (Mean Average Precision) [22]. 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 [23].

- **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:

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

- **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.

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

In order to build and test BORA during the developing phase, we used 668 zAppDev models[9], 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 reconstruction process of the new models. Since BORA is a reuse-based recommendation approach, we calculated how many domain-relevant classes BORA could utilize from the repository in order to reconstruct 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 have on the repository, or if BORA doesn't provide any relevant recommendations, 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 4

[9]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

Table 2: 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

2.7.3 Evaluations of the Recommendations From Attributes

Table 3: Evaluation process of class recommendations based on 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
										0.89450

Same as for the class names evaluation process, also for the class attribute evaluation, we simulated the construction of four different models with the help of our approach. Except, during the iteration from one class to another of the model under construction, this time, we considered the class attributes for getting recommendations. We manually selected the class's two or three most relevant attributes as input for our approach. Then, based on the recommendations, we calculated the metrics the same as we did for the class names evaluation process.

We present the evaluation results in Table 3. Based on the evaluation results, we explain the RQs in the discussion part:

2.7.4 Iteration to Get Completely a Model Reused

In addition, we have evaluated the approach by testing it on how many steps we can get to recommend/reuse an entire model within the repository. We started the reuse process from a given class within the model and proceeded to design the whole model by selecting the recommendations given by the approach. 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

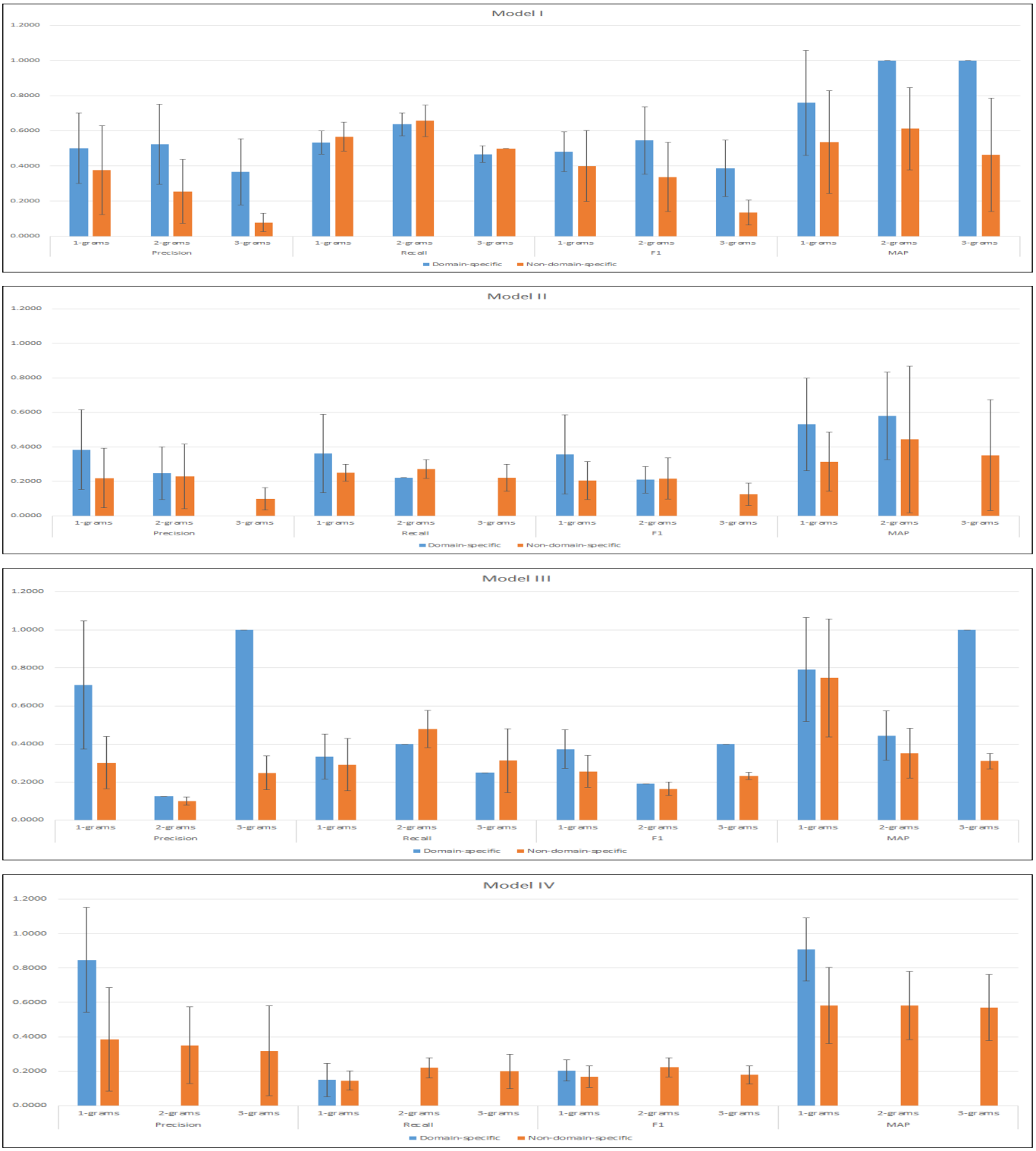

Figure 16: N-grams metrics comparison

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 Fig. 17, 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 a model named Actor with 16 classes. The results show that domain-specific n-grams almost always have higher precision than non-domain-specific ones. However, the recall is usually higher by non-domain-specific n-grams. One crucial point outlined in Fig. 17 is the confirmation of our assumption that starting from the most connected class required less iteration to completely get

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										

Figure 17: Recall the sum-up evaluation of BORA

recommended a model from the repository except on the Actor model. The reason why the Actor model is faster wholly recommended when starting from the least connected class, i.e., ActorInboxMessage, rather than starting from the highest connected class, is that in the observed Actor model, the highest connected class is ActorFunctionRestriction - which defacto is not the most connected class among all Trip-related models. It means there are other Actor-related models within our repository, and we concluded that the class Actor - not in our observed Actor model - is much more connected than ActorFunctionRestriction. Thus, since the least-connected class in our observed Actor model (ActorInboxMessage) is connected with the Actor class from another Actor-related model, cause that all Trip-related classes will be recommendations directly on the next iteration.

2.7.5 Discussion

Based on the results depicted in Table 4, we will explain the posed RQs:

- **Answer to RQ1:** As shown in Table 4 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, and 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 the models we aimed 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 4, the ranking of BORA measured by Mean Average Precision (MAP) is more accurate in N-gram 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. In conclusion, for the ranking of BORA, we can say that BORA has a promising accuracy for its recommendations.
- **Answer to RQ2:** As depicted in Table 4, we can see that BORA was capable of discovering 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, wherein 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. ,16 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 4 (except coverage@N because it is always 1 except for Model III in domain-specific 3-gram).

Converning the evaluation of the recommendations from class attribute similaritiy, we can conclude that:

- **Answer to RQ1:** As shown in Table 3, the precision is very low (with a maximum of 0.02543 - in Model III). The reason is that many classes have the exact attributes, which causes the approach to provide a lot of not relevant recommendations. And another reason for the low precision value is that the class attributes were not very domain descriptive, i.e., the attributes of the models under construction were general primarily, e.g., name, date, etc., and also, the classes of the model under construction didn't contain many attributes at all.

  The lack of class attributes is also the reason for the low recall value. Having no class-domain representative attributes on both sides - the model under construction and the repository - causes not finding relevant classes for recommendations.

  Consequently, since F1 is the harmonic function of Precision and Recall, its values are also relatively low.

  Finally, since we got many domain-irrelevant recommendations, the ranking of BORA is also relatively low.

- **Answer to RQ2:** 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 process, 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 the most 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 didn't match any class of the model under construction; thus, we got a low recall value.

In conclusion, in the overall evaluation process, we can say that BORA performs very well in reusing the knowledge of previous models in order to construct new ones. BORA also performs relatively well in supporting the citizen developers during the modeling process with domain-relevant recommendations. Finally, we have seen that considering class names instead of class attributes for recommendations is much more effective.

2.7.6 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, structure, and 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 could be considered synonyms and treated them as true positives.

2.8 Related Work

Compared to other approaches, this is the only graph-based search approach that is capable of specifying 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 of reusing 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. [24] 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 [**?**] 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, use these pieces of information to train a deep neural language model, and provides modeling recommendations provided by this trained deep neural language model.

Also, in [25], 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.

Angel et al. [26, 27] 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 [28] 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 [29] Capuano et al. present a UML class recommender that learns from code repositories.

Henning Agt-Rickauer et al. [19] present an approach for domain modeling recommendation namely DoMoRe. more uses a large-scale network of semantic-related terms extracted from different knowledge bases [30] to provide modeling recommendations during the modeling process.

Also, Lissete et al. [31] provide 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 recommended method, format, etc.

2.9 Summary

In this Section, 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 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.

3 Model Reuse for LCDPs based on Natural Language Processing

Contemporary technologies tend to use models as their core artifact since models, compared to code, are easier to understand, develop and maintain from experts in different domains. One of these technologies is the low-code development platform (LCDP). The main activity during the whole software production line when using LCDP is the domain modeling of the software under development, where on top of these models, a lot of upcoming technical steps are fully automated. Although even modeling can have some drawbacks, e.g., it can be a difficult task for novice LCDP users, a non-well-structured model can be prone to errors for upcoming automated steps, and finally, even for experienced modelers re-inventing a model from scratch can be a time-consuming task. Therefore, facilities that enable modeling support are highly demanded. In this work, we present an approach that supports the LCDP users during the modeling process by enabling them to reuse the information from previous models. The approach persists heterogeneous models in a knowledge graph and queries this graph by using a natural language (NL) description. The approach gets as input the user NL description, lemmatizes and tokenizes this text based on the extracted tokens, determines the user's request, queries the knowledge graph based on that request, and returns the relevant information to the user. Currently, this approach separates users' requests into five different cases, although conceptually, it can be extended to more cases. We have evaluated this approach by simulating different users' requests in NL and providing these as input to the approach. The results are promising for future studies.

3.1 Introduction

Modeling is one of the key concepts of software engineering and one of the most important steps during the software development life-cycle. Contemporary technologies tend to use models as their core artifact since models, compared to code, are easier to understand, develop and maintain from experts in different domains. One of those technologies that rely on models as their core artifacts is Low-Code Development Platforms (LCDP). The main activity during the whole software production line when using LCDP is the domain modeling of the software under development [2, 7], where on top of these models, a lot of upcoming technical steps are fully automated [5, 6]. Although even modeling can have some drawbacks. First, it can be a difficult task for novice LCDP users, inventing a model from scratch requires modeling experience on the one hand and domain-relevant knowledge on the other hand. Secondly, a non-well-structured model can be prone to errors for upcoming automated steps, for instance, if the domain model contains errors, then also the UI generated from the domain model, and also the DSL code will be prone to errors. And finally, even though modeling is easier than coding, re-inventing models from scratch can be a time-consuming task even for experienced modelers. Therefore, facilities that enable modeling support by reusing the existing, well-constructed models are highly demanded. Thus, in order to facilitate the modeling process, in this publication, we will present an approach that enables the reuse of the existing models by abstracting them into a homogeneous graph-based model and querying this graph from users' natural language (NL) requests. The approach is capable of reusing complete models or specific classes. At the current stage, the approach determines 5 different query cases to catch different users' intents, although the query cases can be extended based on different model types and user requirements. The users can ask to get a completed model from the repository by writing its name or some classes the model contains down to specific classes by writing their names, some of their attributes, or related classes. Whether the models or relevant classes will be suggested relies on the users' requirements.

The approach initially gets as input the user NL description, lemmatizes and tokenizes this text, and performs part-of-speech filtering. Then based on the extracted tokens, it determines the users' request, queries the knowledge graph based on that request, and returns the relevant information to the user.

The approach is currently integrated into the zAppDev LCDP in its professional and enterprise edition.

This Section is structured as follows: In Subsection ,3.2 we will provide a running example in order to clarify the aim of the Section and the goal of the presented approach. Afterward, in Subsection ,3.3 we will explain in more detail the proposed approach. This will be followed by tool support of the approach in Subsection 3.4. Then, we will present the evaluation of the approach in Subsection 3.5. In Section 3.6, we will present the related work to our approach, followed by some conclusion words in Subsection 3.7.

3.2 Running Example

To clarify the aim of this work, in this section, we present a running example of how to reuse models or model fragments by requesting them in natural language.

As depicted in Fig. 18, the user will be able to explain in her own language what kind of model or model artifact she is looking for. Based on the information extracted from the models in our repository, we distinguished five different query types users can run against our repository. We assume that users will ask to get a specific class from the repository as presented in Fig. 18 case a). If the user asks for a specific class like "Give me a Trip class", he will get as a response all Trip-relevant classes from the model

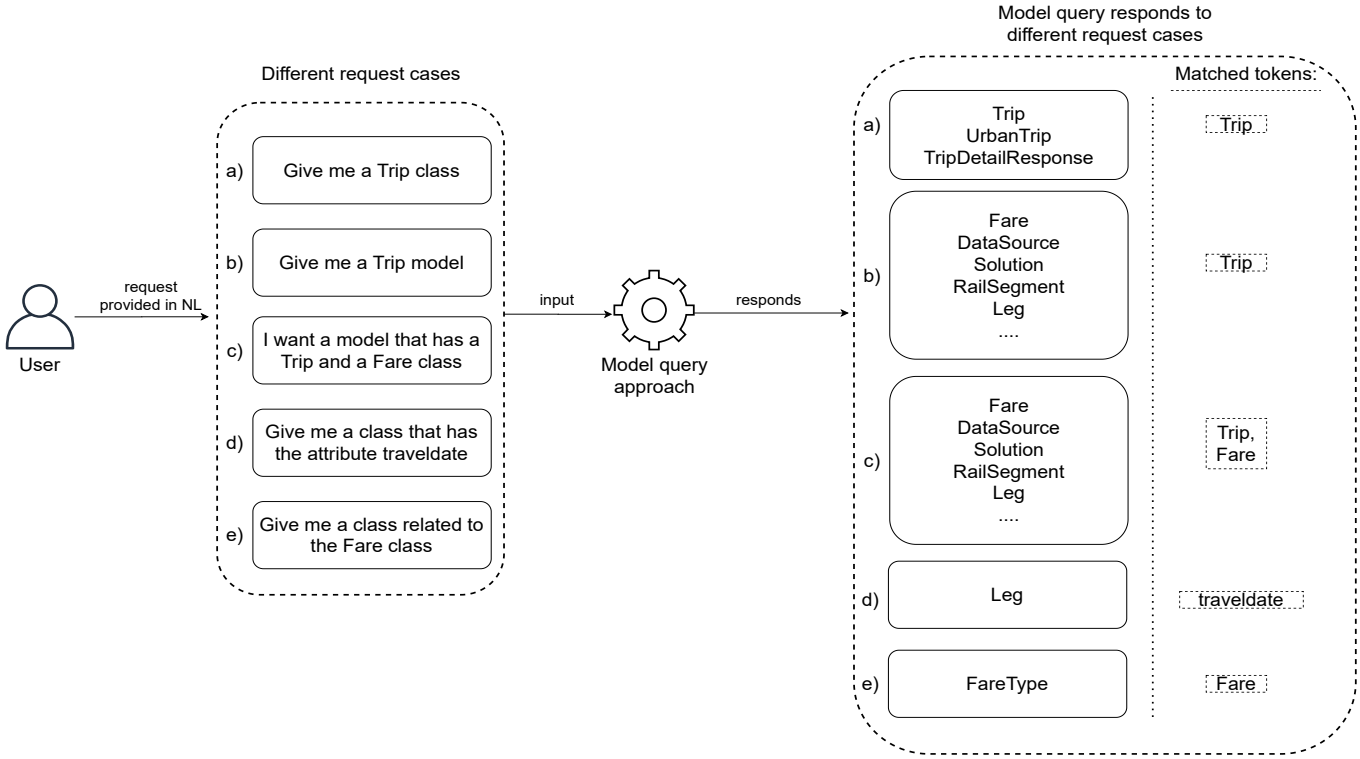

Figure 18: Running example of model reuse by providing natural language requests

query approach like Trip, UrbanTrip, or TripDetailResponse. To reason about the response, the approach provides also the matched tokens to the repository extracted from the users' request. As shown in case a), the relevant token is Trip which will also be provided next to the recommended classes. Fig. 18 at b) presents another case of user request. In this case, we assume that users will ask to get a complete model for reuse. If she types "Give me a Trip model", now will she get as a response all classes belonging to a Trip model like Fare, DataSource, Solution, etc. Now the user can select one or multiple classes, which will automatically be integrated into the modeling canvas.

We assume that users may ask also for getting models that contain specific classes, as shown in c) in Fig. 18. Or the users may ask for classes with specific attributes as described in d). Or finally, we also believe that the users' interest is to get related classes to any specific class, as shown in e).

We suppose that these five different user request cases can significantly support the users during different steps of the domain modeling process, e.g., instead of starting from scratch, the users can immediately ask for a completed and well-structured model persisted in the repository (case b) and c)). During the modeling process, or in the end, the user can ask for a specific class (case a)), a class related to another class (case e)), or even for a class that has specific attributes (d)).

Based on this running example, we have proposed an approach for model reuse which will be explained in the coming section.

3.3 Approach

This section explains in detail the proposed approach for model reuse based on querying a knowledge graph by a given NL request. Before we jump to our proposed approach, we believe it is important to initially explain how we built our knowledge graph.

3.3.1 Knowledge graph

The initial step of reusing heterogeneous models is model abstraction [17]. In this phase, the relevant information of the existing models will be extracted and persisted in a way that can be accessed afterward. In our proposed approach, we have abstracted heterogeneous models to an RDF (Resource Description Framework) graph. We selected RDF as it is the W3C standard[10] and also graph-based repositories are more query efficient than relational databases [12].

Even though any graph-based model can be integrated into this knowledge graph, Fig. 20 depicts only the current range of models that are persisted in our knowledge graph:

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

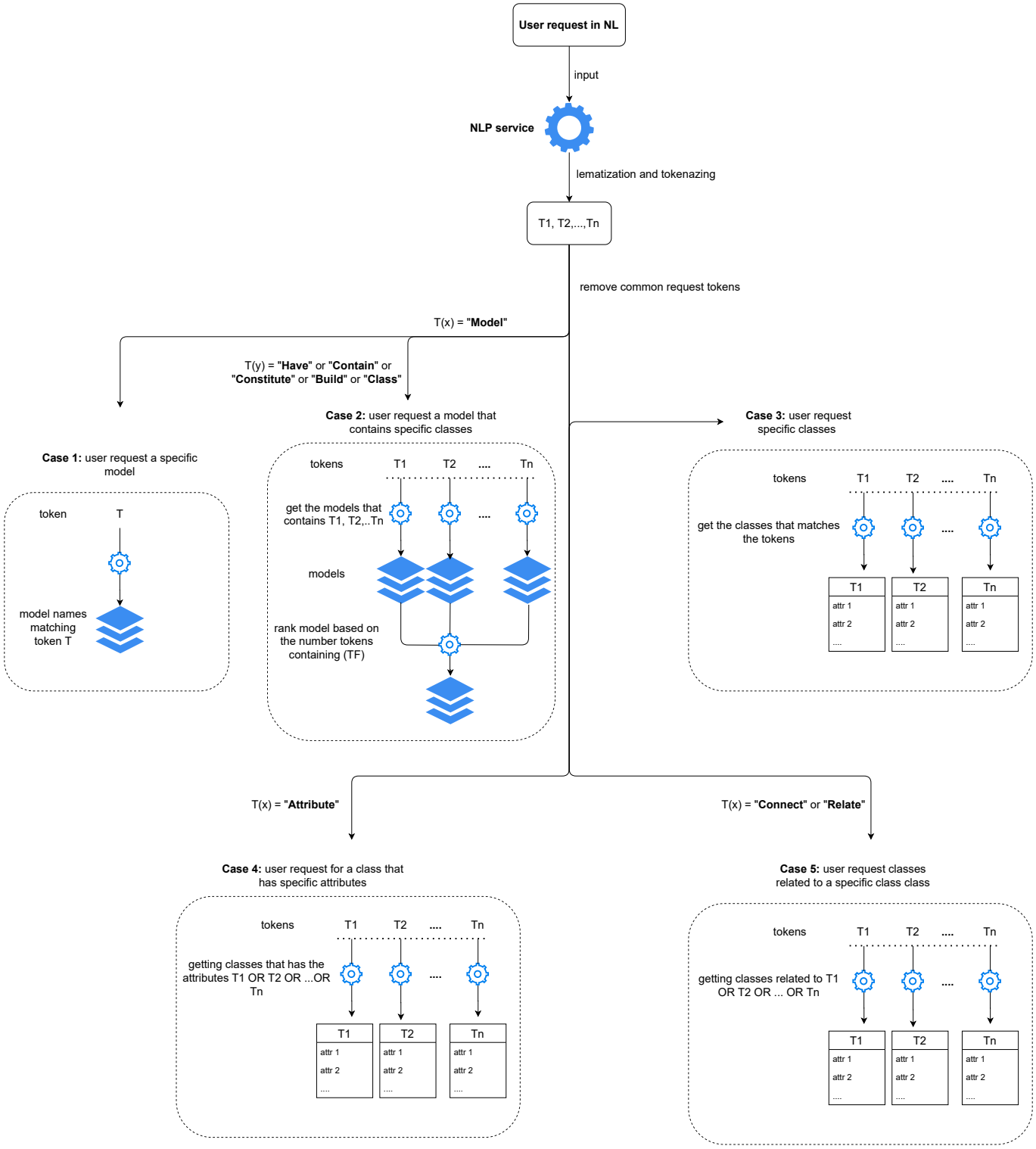

Figure 19: A model reuse approach based on NL and Knowledge graph

- **zAppDev** models: zAppDev[11] is a low-code platform that enables users with no programming background to build full-stack complex applications only by developing domain models and writing DSL code. The zAppDev domain models are called Business Object models (BO).
- **Microsoft CDM**: Also, the Microsoft Common Data Model has been integrated into our knowledge graph.
- **Ecore and MoDisco** models: The MoDisco[12] reverse-engineered models, and also the Ecore models are used here to enrich our knowledge graph.

As also shown in Fig. ,20 in order to integrate heterogeneous types of models into a homogeneous knowledge graph, we had to extract the relevant information from each model, like model name, class

[11]https://zappdev.io/
[12]https://www.eclipse.org/MoDisco/

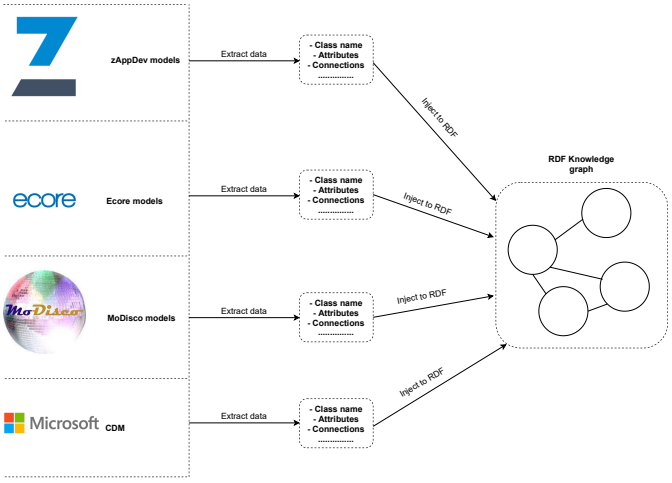

Figure 20: Models integrated into the knowledge graph

names, class attributes with their related datatypes, the relationship between the classes, relationship names and their cardinalities. All this extracted information has been persisted then into a single RDF graph which serves as the knowledge base of the approach. Although, in order to realize this integration, we had initially to build different model parsers so we could read and extract the relevant information from the respective model. Then we created a model encoder that enables injecting all the extracted information into a homogenous RDF file. We practically converted different types of models to RDF graphs. Finally, all these RDF graphs are merged into a single RDF graph so we can refer to it as a knowledge graph. The integration process of heterogeneous models into the knowledge graph is depicted in Fig. 21.

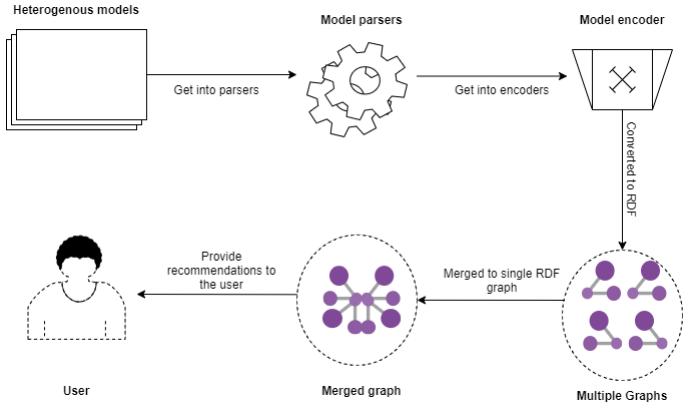

Figure 21: The integration process of heterogeneous models to the knowledge graph

3.3.2 Pre-processing of the users' request

As depicted in Fig. 19, the initial step of our approach toward proving modeling recommendations to the user is getting the user request given in natural language (NL) and passing it through an NLP service. In order to extract the relevant information from the NL request, we have used the Stanford CoreNLP[13] library as a service for processing natural language. With the help of this library, we derive all tokens that exists in the users' request. The text will initially go through a lemmatization process to remove inflectional endings of words and return to their base form. There will also all the stop words be removed automatically. Then the lemmatized text will go through a tokenization process in order to extract all the tokens in the text. The part-of-speech tags that will remain in our approach are NN (Noun, singular, or mass), NNP (Proper noun, singular), VB (Verb, base-form), VBD (Verb, past tense), VBG (Verb, gerund, or present participle), VBN (Verb, past participle), VBP (Verb, non-3rd person singular present), VBZ (Verb, 3rd person singular present). These tokens will constitute our pre-processed token list and will be used afterward for the querying process.

3.3.3 Querying process

After having extracted all the tokens from the users' requests, the next process is querying the knowledge graph. Currently, we distinguish five different query cases based on the users' requests. The approach

[13]https://nlp.stanford.edu/software/

can query for reusing completed models or reusing specific classes. Which query type will be triggered depends on what the user is asking for. Currently, the approach determines the user intent for getting a complete model reused if the user has typed ”model” in any form within his NL request. In case the token ”model” does not exist among the tokens set, the approach queries for specific classes to be reused. The query process for each case explained in detail is as follows:

- Case 1: Querying for specific models: If any of the tokens T1, T2,..., Tn is ”Model” e.g., from the user request ”Give me a Trip model”, then the approach will query the knowledge graph for a specific model. The model name to be queried will be specified by the remaining token after removing the ”Model” token from the pre-processed token list in the first step. In this case, the remaining token is ”Trip” hence the approach will now query for a ”Trip” model within the knowledge graph. If there is any Trip relevant model, the approach will query all its belonging classes and return them to the user as a response to his request for a specific model. And since the token ”Trip” was the matching token, the approach queried the knowledge graph, thus will, ”Trip” also be returned as a matching token in order for the approach to reason about the response.

- Case 2: Querying for models having specific classes: Similar to case 1, if any of the tokens T1, T2,..., Tn is ”Model”, the approach will check for a specific model. Except, in this case, there is more than one token left, and any of these remaining tokens is equal to any of these strings: ”Have”, ”Contain”, ”Constitute”, ”Build”, or ”Class”. The reason behind this is that we assume that in any case (or at least most cases), the user will use the combination of the tokens ”Model” and any of the above ones when she wants to query for a model that has specific classes, e.g., ”Give me a model that has a Trip and Fare”, the relevant tokens in this request are ”Model”, ”Have” - which derives from ”has”, ”Trip” and ”Fare”. Having clarified that the user intent is to query for a model that has specific classes, the approach will now query for a model within the knowledge graph that has the specific class names. The remaining tokens after removing ”Model” and ”Have” (or Contain”, ”Constitute”, ”Build”, or ”Class”) from the pre-processed list in the first step will be considered as the class names the user is looking for e.g., from the above user request the remaining tokens would be ”Trip” and ”Fare”. Afterward, since there might be more models that have any of the requested classes, all the models will be ranked based on the number of tokens they contain, i.e., the model that contains the most tokens will be ranked first in the recommendation list, the model that contains next most tokens will be ranked as the second, etc. Finally, since there may be a lot of models with ambiguous names (DTO, X, etc.) when the approach provides model recommendations, it also gives the list of tokens the recommended models have in order to reasons about the recommended models, e.g., Trip (Trip, Leg), DTO (Trip, PointOfInterest) which means: we recommend you the Trip model because it contains the ”Trip” and ”Leg” class as you requested.

- Case 3: Querying for a specific class: In any of the extracted tokens, T1, T2,..., Tn is equal to ”Class”, or non of them is equal to ”Model”, ”Attribute”, ”Connection”, and ”Relation”, then the approach determines that the users intent is to get specific classes from the knowledge graph, e.g., ”Give me a Trip class”. The names of the requested classes will be determined after removing the ”Class” token from the pre-processed list, the remaining tokens will be considered as the requested classes, e.g., from the above example, the remaining token would be ”Trip”. Thus, the approach will query the knowledge graph for each of the remaining tokens in the pre-processed list (without the ”Class” token now), and if a class corresponds to any token, this will be recommended to the user. If the user selects any of the recommended classes, this will be directly integrated into the modeling canvas together with all its belonging attributes.

- Case 4: Querying for a class that has specific attributes: If any of the tokens T1, T2,..., Tn is ”Attribute”, e.g., ”Give me a class that has attribute traveldate”, then the approach will query the knowledge graph for a class that has specific attributes. The attribute names to be queried will be specified by the remaining token after removing the ”Attribute” and ”Class” tokens from the pre-processed token list in the first step. In this case, the remaining token is ”traveldate” hence the approach will now query for a class that has an attribute ”traveldate” within the knowledge graph. If there is any matching class, the name of that class will be returned to the user as a query response, e.g., the class ”Leg” will be returned in that particular case. If the user selects the recommended class, then it will be integrated into the modeling canvas together with all its belonging attributes.

- Case 5: Querying for a class related to a specific class: If any of the tokens T1, T2,..., Tn is ”Connect” or ”Relate”, e.g., ”Give me a class is connected to Fare”, then the approach will query the knowledge graph for a class that is connected to a specific class. The class name the user is looking for will be specified by the remaining token after removing the tokens ”Connect”, ”Relate”, and”Class” from the pre-processed token list in the first step. In the above example, the remaining token will be ”Fare” hence the approach will now query for a class that is connected to ”Fare” within the knowledge graph. If there is any matching class, the name of that class will be returned to the user as a query response, e.g., the class ”FareType” will be returned as a response to the above user request. If the user selects the recommended class, then it will be integrated into the modeling canvas together with all its belonging attributes.

3.4 Tool Support

The approach is developed using Java Spring Boot and has been exposed as a REST API in order to enable its integration of it into any LCDP. The approach is open-source and can be downloaded on GitHub[14].

As a proof of concept for the proposed approach, we have integrated it into zAppDev. zAppDev[15] is an LCDP that enables users with no background in programming to create full-stack web applications just by developing models and simple Mamba code - the domain-specific language for zAppDev.

In Fig. ,22 we have depicted the screenshot of the performance of the approach in two cases, when the user requests a specific model (case 1), and when the user requests a specific class (case 3). When the user triggers the approach by clicking the respective button on the modeling canvas shown in step 1 of Fig. 22, a pop-up window will open which allows the user to write in NL its request for a specific class or model (A). After the user has given his request, he can press the "Search" button (B) to get recommendations based on his request. Based on his request, as depicted in Fig. ,19 the approach will provide recommendations to him (2. a and 2. b). The recommended classes will be given below the Search button as a list of classes (C). When the user selects any of the recommended classes, this class will be automatically integrated into the modeling canvas, as shown in step 3. The integrated class will be integrated with all its related attributes/datatypes it has in the Knowledge base.

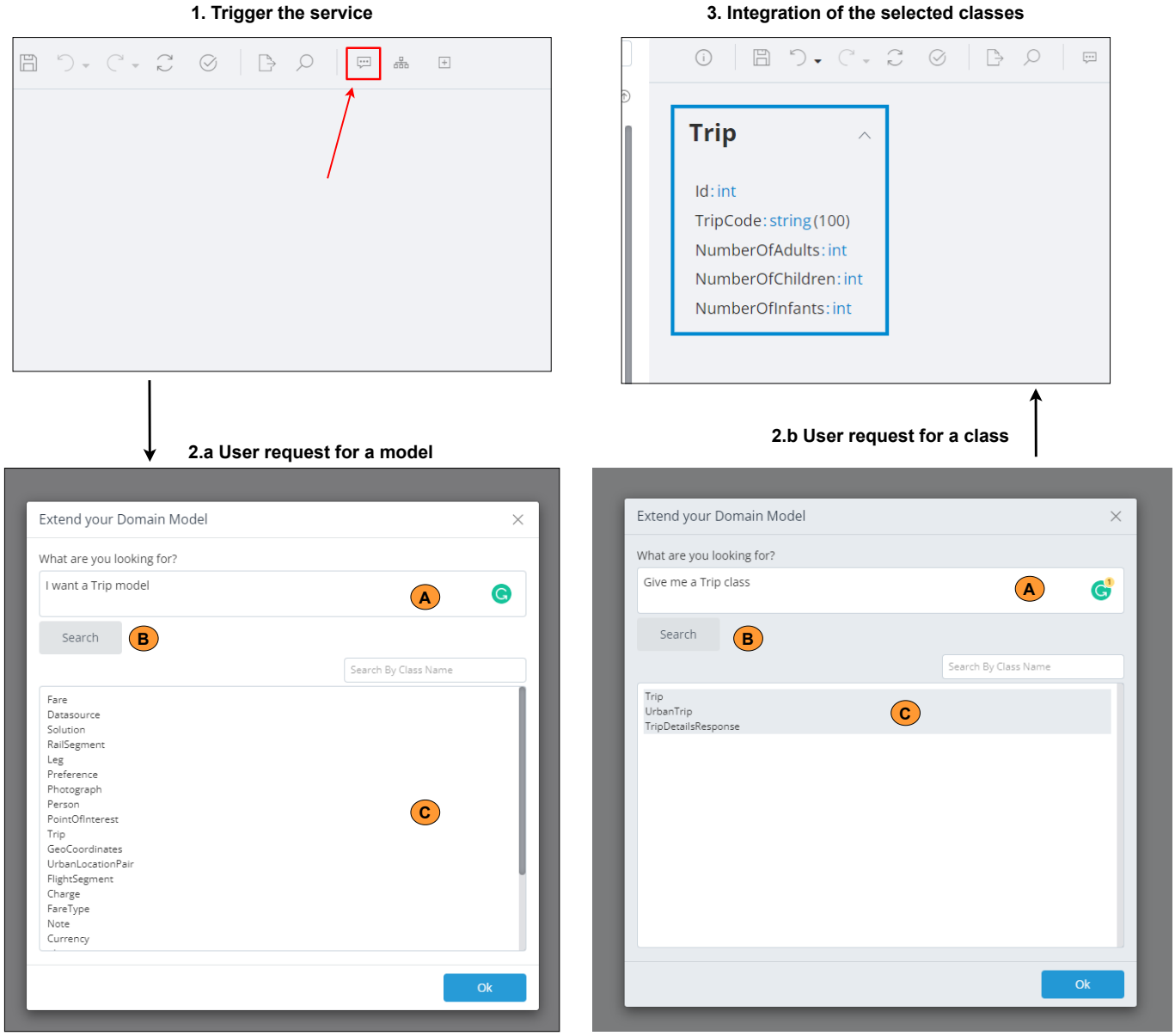

Figure 22: Integration of the approach into the zAppDev LCDP

[14]https://github.com/iliriani/Modeling_Bot, Due to industrial IP constraints, the zAppDev models cannot be found/queried by the open-source approach

[15]https://zappdev.com

3.5 Evaluation

3.5.1 Research Questions

Table 4: Evaluation of the model reuse approach

		NL request for a				
		Case 1: Specific model	**Case 2: Specific model that has classes**	**Case 3: Specific class**	**Case 4: Class that has specific attributes**	**Case 5: Related classes**
MRR	1	1.000000	1.000000	1.000000	1.000000	1.000000
	2	0.000000	1.000000	0.500000	0.000000	1.000000
	3	0.000000	0.500000	1.000000	0.000000	1.000000
	4	0.000000	1.000000	0.001880	1.000000	1.000000
	5	0.333333	1.000000	1.000000	1.000000	1.000000
	6	0.250000	0.000000	0.100000	0.000000	0.000000
	7	0.000000	1.000000	0.000000	0.333333	1.000000
	8	0.333333	0.333333	1.000000	0.000000	1.000000
	9	1.000000	0.125000	1.000000	0.142857	0.000000
	10	0.000000	0.333333	0.500000	1.000000	1.000000
	Average	**0.291667**	**0.629167**	**0.610188**	**0.447619**	**0.800000**

In order to evaluate our proposed approach, we initially set these research questions:

- **RQ1:** What is the efficiency of the approach to determine the user's intent?
- **RQ2:** What is the efficiency of the approach to return relevant models?
- **RQ3:** What is the efficiency of the approach to return relevant classes?

3.5.2 Evaluation Methodology

A proper evaluation of our approach would be to let different users query our approach by using different requests in their natural language and get their feedback. But due to limited free personnel at our company (CLMS) and since the approach is not available on the free edition of zAppDev, we had to simulate different users' requests in natural language in order to evaluate the approach. Thus, we initially created for each case described in Section 3.3.3 different sentences by trying to cover as much as possible the users' requests in natural language, e.g., some of the sentences for simulating case 3 where users ask for specific classes are: "Give me a class_name class", "I would like to have a class_name class", "Can I have a class_name class", "Give me a class_name", etc. where class_name is a placeholder for a randomly selected class from the repository. Then, for each case in Section 3.3.3, we randomly selected a sentence from the relevant list that contains all sentences for that specific case. In that randomly selected sentence, we replaced its placeholders with the randomly selected model artifacts from the repository. Now, these randomly selected sentences containing the randomly selected model artifacts will serve as the user input for our approach.

After providing the input sentences to the approach, in order to calculate the ranking of the requested model artifact responded to by the approach, we calculated the MRR (Mean Reciprocal Ranking) [32] for the requested model artifact. We calculated the MRR only when the requested model artifact was among the top ten recommendations provided by the approach. Otherwise, the MRR is considered 0. We repeated this process ten times for ten different sentences and calculated the average MRR at the end. The entire evaluation process proceeded the same for each case presented in Section 3.3.3. The evaluation results are depicted in Table 4.

3.5.3 Discussion

Based on the evaluation results provided in Table 4, the first thing we can recognize is that there are some 0 values among different cases and iterations. The first reason why we have some MRR 0 values is that some model artifact names are considered adjectives from the Stanford NLP tool and thus will be removed directly in the first part-of-speech filtering process. This happened mostly when requesting specific model names or classes that have specific attributes (case 1 and case 4). This can be resolved when adding also adjectives for further NL processing, i.e., adding the part-of-speech tags "JJ", "JJR", and "JJS", but this will consequently cause much more processing time. Whereas the reason why we have some MRR values less than 1 when requesting specific classes is twofold: first, some classes are in many models. Thus the requested classes might occur first in other models except for the one we are searching for, e.g., classes like User, Account, Manager, etc. And if the requested class is not among the top 10 recommendations, then the respective MRR will be set to 0. And secondly, there are some class/attribute names that are substrings of other class/attribute names thus, there may occur to get more results for a given class request, e.g., retailAttributesGlobalLookup is a substring of retailAttributesGlobalLookupEntity. Hence we will have more classes/models recommended for a given attribute/class, and that is why we have some MRR results lower than 1.

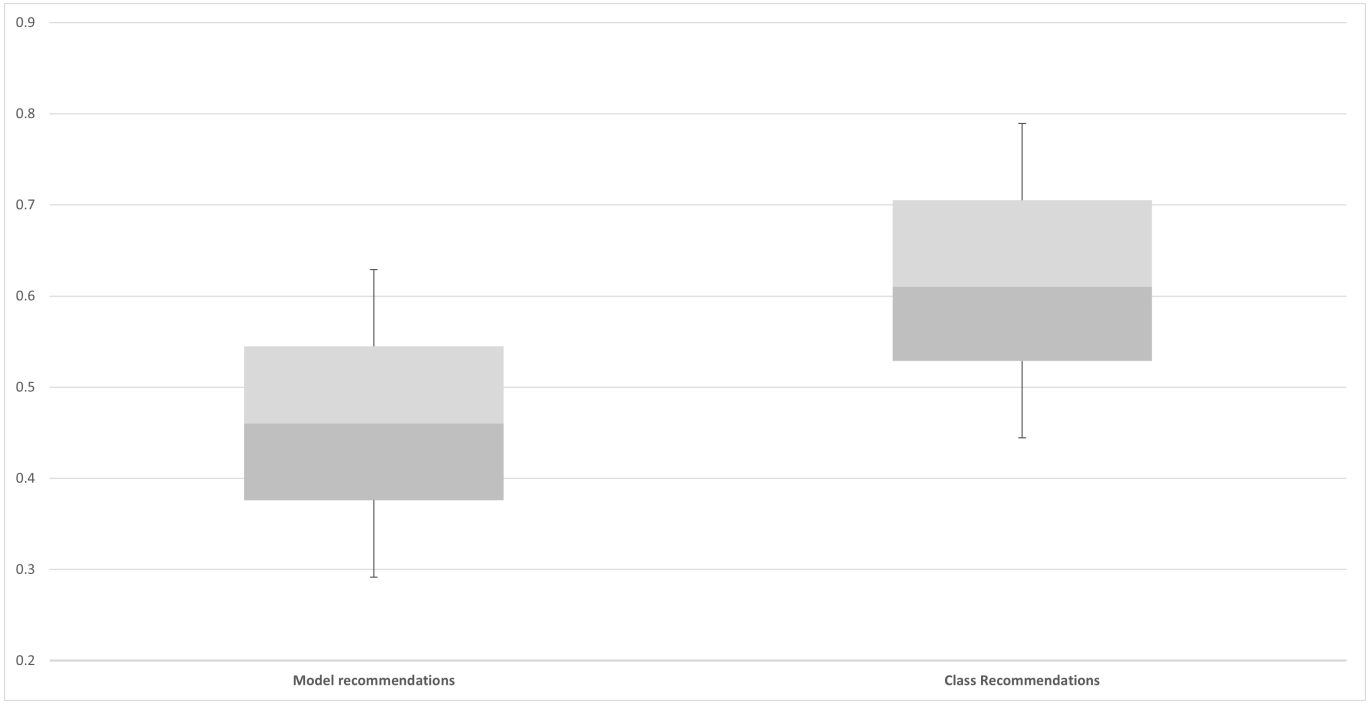

Figure 23: Average of the MRRs throughout different cases for model and class recommendations.

In general, the results are promising for future studies. And from the evaluation process, we can conclude concerning the submitted RQs and provide these respective answers:

- **Answer to RQ1:** We see that the approach was in each case able to determine the intent of the user, i.e., to correctly guess what the user is looking for. Even though the approach currently determines 5 different query cases provided by the user, the response is either a model or a class, and in each case, the approach correctly determines whether the intent of the user is to get a model or a class.
- **Answer to RQ2:** In Fig. 23, we have outlined the average of the respective MRRs for the model and class recommendations cases provided in Table 4, i.e., an average of the MRRs for case 1 and case 2 for model recommendations, and the average of the MRRs for case 3, case 4, and case 5 for class recommendations.

  Hence in Fig. 23, we can see that the average MRR for model recommendation is 0.46, which means that the exact model is mostly the second one and that the recommended model is always among the top three recommended ones. Nevertheless, when asking for a specific model name can often cause the approach to not respond since some model names can be considered as pronouns and will not be processed by the approach, as the evaluation results in Table 4 for that particular case show that the average MRR is 0.29.
- **Answer to RQ3:** Concerning the efficiency of returning relevant classes based on the user request, the evaluation results provided in Fig. 23 show that the average MRR for class recommendation is around 0.62, which means that the requested class is almost among the top 3 recommendations and is mostly the first one. As shown in Table 4, the MRR is lower only when asking for a specific class based on some attributes it might have (Case 4), which, as explained before, depends highly on the attribute names and whether they will be considered as names or adjectives by the Stanford NLP tool. Although, in general, based on the evaluation results, the approach seems to be quite effective in class recommendations.

As a wrap-up note, we can conclude that based on the evaluation results, the approach seems to be quite effective for determining the users' query intent and providing them with reasonable modeling recommendations based on their requests. The approach seems to be interesting for further studies.

3.5.4 Threats to validity

One possible threat to the validity of our evaluation could be the NL request we gave as input to the approach. The user requests in NL might not present all possible requests for querying the knowledge graph. Thus, to mitigate this, we used 10 different types of simulated user requests for each case and selected these randomly for each iteration.

Also, the selected model artifacts, i.e., model names, class names, and class attributes, can significantly change the evaluation results since some types of model artifacts are directly filtered out from the NLP service and thus, no relevant response will follow for them. Thus for each case, we randomly selected any relevant model artifact from the knowledge graph.

3.6 Related Work

Model recommendations based on NLP:

In [33] presents a bot, namely DoMoBot, which enables the modeler to create domain models just by adding NL. DoMoBot provides a description that reasons the provided recommendations, allows modelers to interact with the recommended domain model, and keeps track of the modelers' decisions. The difference in our approach to this is that ours is reuse-based, i.e., we get the users' intent and query a repository of previously created models in order to reuse their information for creating new ones.

In [34] is presented an approach that enables the querying of domain models based on NL. The approach relies on the manually created meta-model by the chatbot designer, which contains the structure of the models that have to be queried and the additional NL configuration information. In our approach, there is no need to provide any meta-model and any configuration in order to enable the querying process from NL, that's because we use a homogenous knowledge graph for persisting the models, which also allows our approach to be integrated and used on different LCDP and even other MDE tools.

[28] Burgueño et al. present 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. As a difference to it, our approach uses the knowledge of different models in order to provide modeling recommendations based on users' queries.

Similar work has been done recently in [29] where Capuano et al. present an approach for recommending UML classes. This approach extracts the knowledge from code repositories and uses NLP techniques and class relations afterward to predict the next modeling class. Here above all, our approach differs in that the source of the recommendations are models and not code.

Graph-based model recommendations:

In [35] presents an approach based on a pre-trained language model for providing metamodel concept recommendations. This approach gets lexical and structural information from metamodels, use these pieces of information to train a deep neural language model like BERT, and provides modeling recommendations provided by this trained deep neural language model.

Di Rocco et al. [24] present a metamodel recommendation approach that predicts based on collaborative filtering. The approach is named MemoRec. it uses graphs to encode various information extracted from metamodels.

Also, In [25] Di Rocco et al. present MORGAN, another graph-based model recommender that uses graph-kernel techniques to train and predict model artifacts.

Compared to the above-mentioned graph-based recommendation approaches, our approach uses NL for querying the knowledge graph, which makes it more user-friendly and easier to access, especially on LCDP.

Model query engines:

Lucrédio et al. [36] presents MOOGLE - a model search engine. Moogle is a query-based model search engine. It uses the information of metamodels for indexing and providing advanced queries on the models. The difference between our approach and Moogle is that Moogle keeps the models in their original format, it uses indexed model descriptors generated by different model extractors to find relevant query information, and the users have to specify the model format on their textual description. Whereas our approach uses a homogenous knowledge graph to abstract heterogeneous models. Thus, the query facility performs independently of any metamodel, and the information from heterogeneous models can be reused for constructing new models.

In [37], López et al. present MAR - a model search engine. It uses query-by-example to get matched models from the repository. MAR generates indexed paths from different models, which will be used to find similarities and for scoring the query-by-example input given by the user. The difference between MAR with our approach is that we use NL for querying the knowledge graph while MAR is a query-by-example approach.

Angel et al. [26, 27] present a model query approach named Extremo. Extremo abstracts different level and language-agnostic models in a homogenous format and provides a list of different queries to get information from those models. Our approach uses a graph-based repository instead and also uses NL to query that repository.

A significant difference in our approach to others is that it provides the reuse of heterogeneous models by using different only the users' requests provided in NL. The users can perform different query cases, starting from asking for completed models down to related classes and attributes level.

3.7 Summary

In this Section, we have presented an approach that enables the LCDP users to query a knowledge graph constituted of different level and language-agnostic models by using their natural language for their query request.

The approach enables the reuse of complete models or specific classes. Based on how the users might

request a specific model or class, the approach determines currently 5 different request cases, which conceptually can be extended for more specific cases.

We evaluated the approach by simulating different user requests for different model artifacts and the results revealed that the approach is relatively efficient to determine the users' intent and provide relevant feedback.

The current approach is integrated into the zAppDev LCDP.

4 Towards Process-Mining-as-a-Service for LCDPs

Businesses increasingly use Software as a Service (SaaS) to procure business application software. SaaS can facilitate cost reduction, quality improvement, and quick, low-cost innovation of current operations [38]. Similarly, the LCDPs can utilize process mining as a service component for various purposes. For instance, a mined process model can be used to recommend the users with the best next action to undertake in a given situation based on the past observed behavior of users.

In D4.2, we presented the operation-based mining approach and a collection of concepts for mining and mapping operational data. We discussed automation possibilities for the capture, serialization, and analysis of operational data in LCDPs by presenting a proof-of-concept Modeling Process Mining Tool (MPMT[16]) accessible via this GitHub repository [17]. In this deliverable, we present two new use cases. The CAEX Workbench case study is based on the Sirius MPMT plugin, and the Theia Ecore Tools case study is based on the newly implemented plugin for EMF Cloud.

The rest of this section is organized as follows: Section 4.1 recaps the MPMT capabilities while Section 4.2 and Section 4.3 present its use in two case studies.

4.1 Modeling Process Mining Tool in a Nutshell

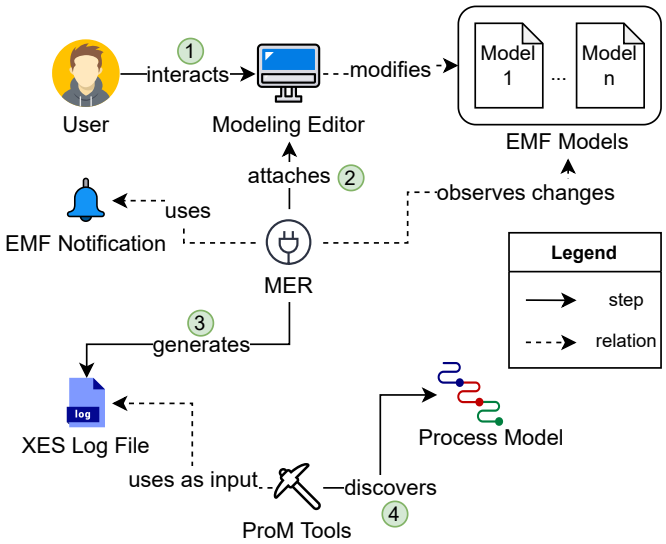

Figure 24: Modeling Process Mining Tool workflow

Figure 24 shows the workflow in MPMT on a high level. In this tool, first, the user starts interacting with the modeling editor (step 1). Then the Modeling Events Recorder (MER) plugin attaches itself to the created editing session (step 2). The plugin listens to changes in the EMF models being modified within the graphical editors. It can also observe the modifications of the Aird model, which contains the data about the graphical representation of the corresponding Ecore model. This change observation is done via the EMF Notification API, a built-in API in EMF. In step 3, as the editing session is closed, the plugin generates an XES (eXtensible Event Stream) log file. XES[18] is an IEEE standard format for software event logs. Step 4 involves leveraging a process mining tool called ProM[19] to create a graphical representation of the extracted process model depicting the user behavior as a well-known model such as Petri Net or BPMN. The MPMT is then the result of integrating two tools: MER, developed by JKU, and ProM [39], a well-known general-purpose process mining tool.

4.2 Process Mining for Sirius-based Graphical Editors: The CAEX case study

The first step in developing a Process Mining as a Service component for LCDPs was the development of an Eclipse Plugin, which could be attached to any graphical modeling editor based on Sirius.

Sirius is an Eclipse project that allows the creation of graphical editors for the Eclipse Modeling Framework (EMF) models [40]. Thanks to it, users have a reusable way of defining editors, for the IDE Eclipse or the web, by specifying a Viewpoint Specification Project without needing to be an expert in programming

[16] We introduce the acronym in this deliverable for the first time.
[17] https://github.com/lowcomote/sirius.process.mining
[18] https://xes-standard.org/
[19] https://www.promtools.org/

languages, the Eclipse environment, or its plugin system [41]. Sirius provides quick development of an editor which lets users graphically define models of different types, either using an existing metamodel or after specifying one for a certain domain [42]. It would be possible to specify representations of viewpoints of various natures with Sirius, such as generic diagrams, edition tables, crosstables, trees, and sequence diagrams [43]. These characteristics make Sirius a good candidate for a domain-specific graphical modeling editor to be used in LCDPs.

In this regard, we tested our plugin on a Sirius-based graphical modeling editor developed at JKU, the CAEX Modeling workbench [44]. CAEX[20] is one of the most promising standards for data interchange between engineering tools in production system automation. CAEX is used by the AutomationML standard[21], a neutral XML-based data format representing engineering knowledge in process automation and control. It is widely accepted in Industry 4.0 by its development within an academic and industrial consortium. AutomationML is an integration format for the following standards: CAEX for system topology, COLLADA for geometry and kinematics, and PLCopen XML for logic. Figure 25 provides a descriptive overview of AutomationML modeling concepts. By using CAEX, the entire system model is represented as an instance hierarchy in AutomationML. Devices of the system are represented as instances, which are called internal elements of the instance hierarchy mentioned above. The devices' types are described as system unit classes collected in system unit class libraries. Interconnections between interfaces of devices are represented as internal links. Role class libraries define the meaning of captured information, can be shared among various projects, and thus, are subject to many standardization efforts by the AutomationML consortium.

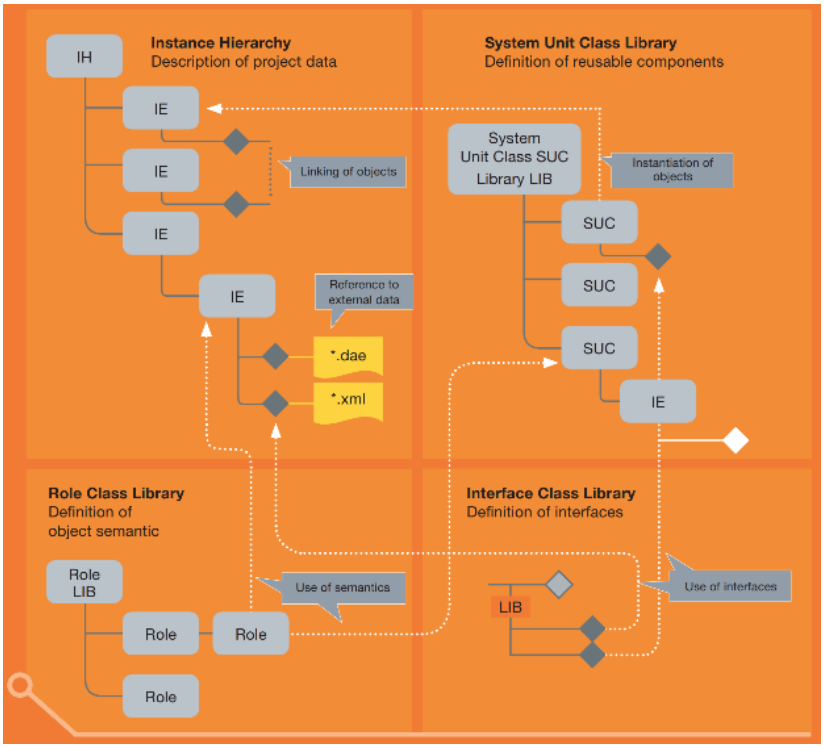

Figure 25: CAEX modelling elements exemplified.[22]

The CAEX modeling workbench has been developed on top of Eclipse EMF and used in research activities on AutomationML [45, 44]. Figure 26 depicts a screenshot of the environment of the CAEX modeling workbench. It allows editing the Instance Hierarchy of CAEX models.

As the user interacts with the editor and creates, modifies, and deletes graphical elements from the modeling canvas (IEs, interfaces, and links), the MPMT plugin (specifically the MER part) captures all of the corresponding events. After a modeling session[23], a log file is generated that includes everything that has happened during that session. Figure 27 represents a piece of such a log file containing all the corresponding events.

Afterward, using an off-the-shelf process mining tool such as ProM [39], we can extract a process model from the log file and generate a visualization from the extracted process model. Figure **??** shows a sample visualization of a process model extracted from a CAEX modeling session.

The CAEX case study is under development. In particular, we are working on offering the CAEX

[20] https://en.wikipedia.org/wiki/CAEX

[21] www.automationml.org

[22] source: https://www.automationml.org/wp-content/uploads/2021/06/AutomationML-Brochure.pdf

[23] A modeling session starts when the editor is opened and ends when the containing Eclipse project is closed.

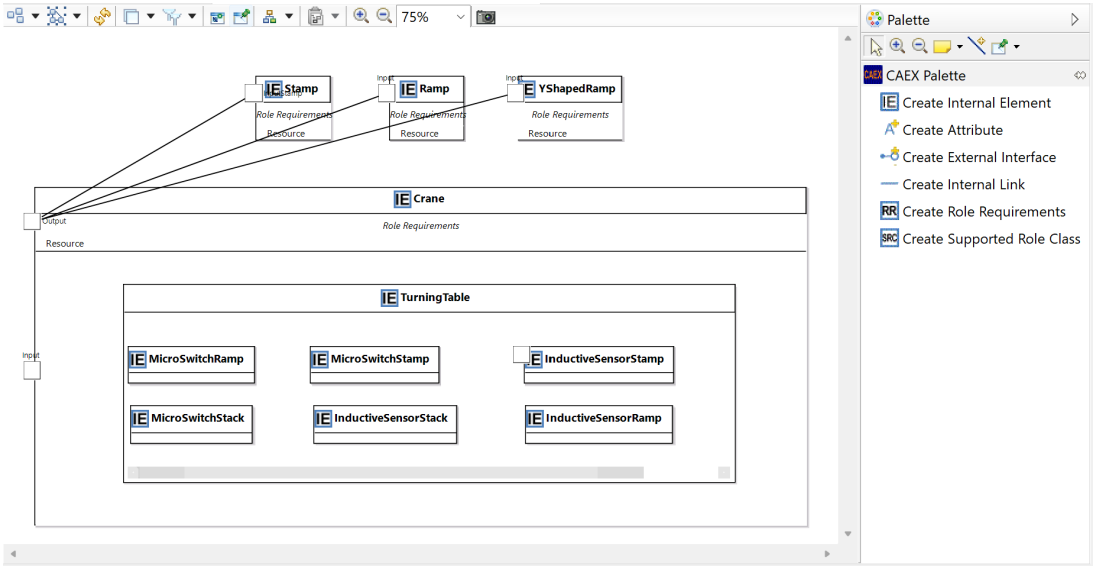

Figure 26: CAEX MDE Workbench: a graphical modeling editor based on Sirius.

```
<xml version="1.0" encoding="UTF-8">
<log xes.version="2.0" xes.features="-" openxes.version="2.27">
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <trace>
        <event>
            <boolean key="set" value="true"/>
            <string key="featureName" value="name"/>
            <string key="resource" value="caex"/>
            <string key="javaClass" value="caex.caex30.caex.InternalElement"/>
            <string key="dataValue" value="Ramp"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-09-07T15:20:39.349+02:00"/>
            <string key="proxy" value="platform:/resource/caex.caex30.ppu.model/PPU.caex#//@instanceHierarchy.0/@internalElement.8"/>
            <string key="concept:name" value="InternalElement:name"/>
            <string key="id" value="//@instanceHierarchy.0/@internalElement.8"/>
            <string key="class" value="InternalElement"/>
        </event>
        <event>
            <boolean key="set" value="true"/>
            <string key="featureName" value="name"/>
            <string key="resource" value="caex"/>
            <string key="javaClass" value="caex.caex30.caex.Attribute"/>
            <string key="dataValue" value="Requirements"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-09-07T15:20:43.795+02:00"/>
            <string key="proxy" value="platform:/resource/caex.caex30.ppu.model/PPU.caex#//@instanceHierarchy.0/@internalElement.8/@attribute.0"/>
            <string key="concept:name" value="Attribute:name"/>
            <string key="id" value="//@instanceHierarchy.0/@internalElement.8/@attribute.0"/>
            <string key="class" value="Attribute"/>
        </event>
        <event>
            <boolean key="set" value="true"/>
            <string key="featureName" value="internalElement"/>
            <string key="resource" value="caex"/>
            <string key="javaClass" value="caex.caex30.caex.InstanceHierarchy"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-09-07T15:20:51.124+02:00"/>
            <string key="proxy" value="platform:/resource/caex.caex30.ppu.model/PPU.caex#//@instanceHierarchy.0"/>
            <string key="concept:name" value="InstanceHierarchy:internalElement"/>
            <string key="id" value="//@instanceHierarchy.0"/>
            <string key="class" value="InstanceHierarchy"/>
        </event>
```

Figure 27: A piece of a sample log file from a session of the CAEX editor.

Modeling workbench augmented with the MER plugin as a solution for CPS modeling. The main goal is to support modeling recommendations for AutomationML modelers by integrating AI-augmented recommender tools to generate recommendations applicable to AutomationML models.

4.3 Process Mining for Cloud-based Graphical Editors: The Theia Ecore case study

With the growth of cloud computing and the progress in web application development, the software industry's future relies on cloud environments and web user interfaces. LCDPs are not exempt from this trend; even now, many of them have web interfaces that can be connected to their cloud-based, back-end servers. Considering that, providing the process mining service to these cloud-based web environments is very important. This is why we have taken some steps in this direction and developed a process mining plugin specifically for such platforms.

Eclipse Theia[24] is a an extensible platform that enables the creation of fully functional, multilingual,

[24]https://theia-ide.org/

cloud-based, and desktop IDE-like programs using cutting-edge web technologies [46]. It is implemented in HTML, CSS, and TypeScript. Theia is divided into two parts: a frontend that runs in a browser or a local desktop program and a backend that can be located on any host or locally on the desktop application. Through JSON RPC over websockets, the frontend and backend

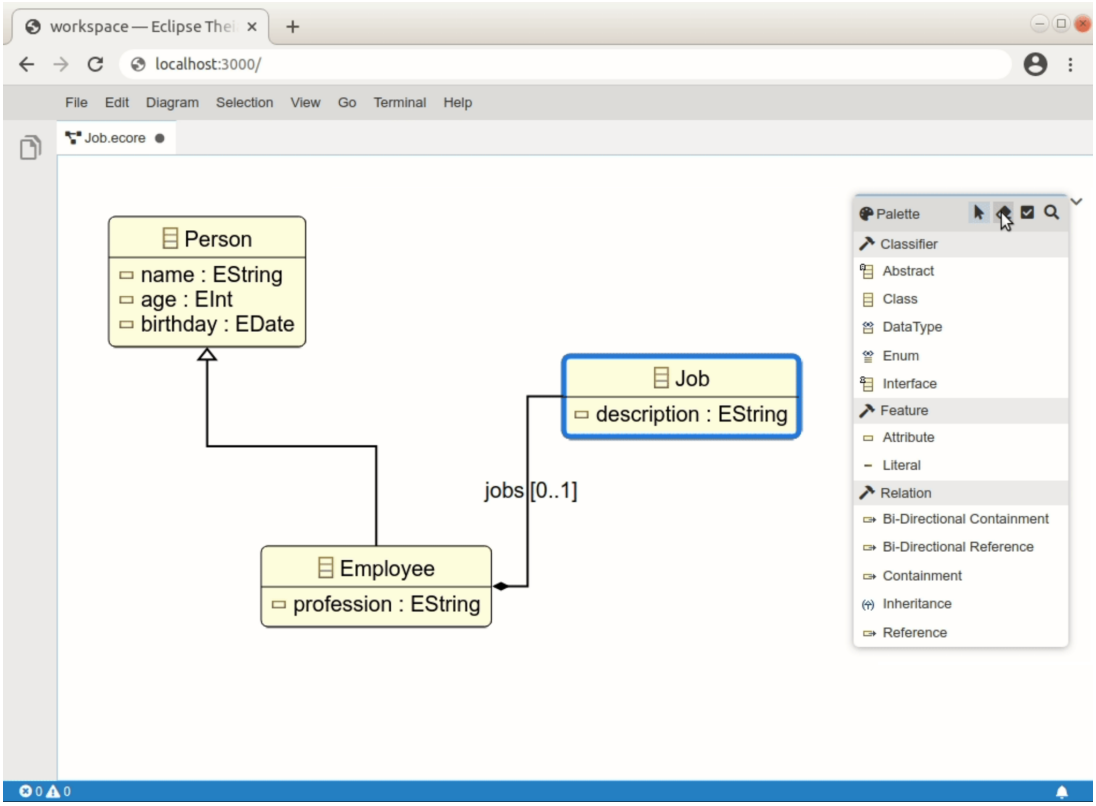

Figure 28: Theia Ecore Tools: a cloud-based graphical modeling editor[25].

communicate [46].

Figure 28 depicts the frontend user interface of Theia while Figure 29 and Figure 30 show a piece of the generated log and the extracted process model from that log, respectively. A Theia application consists of several extensions that can contribute to the front and backend parts. Everything in Theia is built using this extension system [46].

An extensible open-source framework called the Graphical Language Server Platform (GLSP)[26] is available for creating unique diagram editors based on web technologies. The language server protocol (LSP)[27] for diagrams is a feature of GLSP, which also offers extensible client and server frameworks. As a result, GLSP makes it possible to create cutting-edge, web-based diagram editors, with the server handling the labor-intensive tasks of loading, interpreting, and editing following the rules of the modeling language. To utilize GLSP editors easily in web pages, Eclipse Theia, VSCode, and even Eclipse desktop, GLSP offers integration layers [47].

The GLSP-based Ecore editor, integrated in the Eclipse Theia IDE, provides a web-based version of the popular Ecore tools. It allows its users to graphically create and modify Ecore Models via a diagram editor integrated into the web-based IDE Eclipse Theia [48].

The GitHub repository of the Eclipse Theia Ecore Tools has been forked by us, and we have integrated the process mining plugin into the Theia Ecore Tools so that whenever a graphical model is saved, the corresponding event log would be saved. The code is accessible via this repository [28].

4.4 Ongoing Work

The two use cases have shown the applicability of the MPMT tool in different tools and contexts. This paves the path to providing a universal modeling process mining component as a service to low-code development platforms. Evaluation and Empirical studies will be considered in the future.

[25] source: https://github.com/eclipse-emfcloud/ecore-glsp
[26] https://www.eclipse.org/glsp/
[27] https://microsoft.github.io/language-server-protocol/
[28] https://github.com/lowcomote/EMFCloud-process-mining

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="2.0" xes.features="-" openxes.version="2.27">
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <trace>
        <event>
            <boolean key="set" value="false"/>
            <string key="featureName" value="eClassifiers"/>
            <string key="resource" value="ecore"/>
            <string key="javaClass" value="org.eclipse.emf.ecore.EPackage"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-12-12T09:59:17.486Z"/>
            <string key="proxy" value="file:/usr/src/client/workspace/D45/model/D45.ecore#/"/>
            <string key="concept:name" value="EPackage:eClassifiers"/>
            <string key="id" value="/"/>
            <string key="class" value="EPackage"/>
        </event>
        <event>
            <boolean key="set" value="false"/>
            <string key="featureName" value="elements"/>
            <string key="resource" value="enotation"/>
            <string key="javaClass" value="org.eclipse.emfcloud.ecore.enotation.Diagram"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-12-12T09:59:17.486Z"/>
            <string key="proxy" value="file:/usr/src/client/workspace/D45/model/D45.enotation#/"/>
            <string key="concept:name" value="Diagram:elements"/>
            <string key="id" value="/"/>
            <string key="class" value="Diagram"/>
        </event>
        <event>
            <boolean key="set" value="true"/>
            <string key="featureName" value="name"/>
            <string key="resource" value="ecore"/>
            <string key="javaClass" value="org.eclipse.emf.ecore.EClass"/>
            <string key="dataValue" value="NewEClass0"/>
            <string key="type" value="featureChange"/>
            <date key="time:timestamp" value="2022-12-12T09:59:26.065Z"/>
            <string key="proxy" value="file:/usr/src/client/workspace/D45/model/D45.ecore#//Person"/>
            <string key="concept:name" value="EClass:name"/>
            <string key="id" value="//Person"/>
            <string key="class" value="EClass"/>
        </event>
```

Figure 29: A piece of a sample log file from a session of the Theia Ecore Tools.

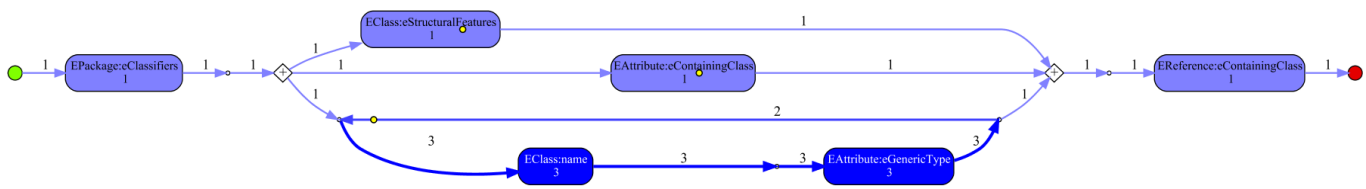

Figure 30: A sample extracted process model from a session of Theia Ecore Tools.

5 Towards Modularization-as-a-Service for LCDPs

In computer science, a conceptual model is an artifact that aims to express the meaning of terms and concepts used by domain experts to discuss the problem and find the correct relationships between different concepts. Conceptual modeling is, therefore, the activity that introduces abstractions that are applied to reduce the complexity of a specific domain, normally to address a certain purpose and derive a common understanding [49]. Independently of the conceptual model's ultimate purpose, we expect a human being to be involved in the creation and analysis process. For example, Entity-Relationship (ER) models have been applied for several decades to create and analyze an abstract, conceptual representation of a data model for human beings. Legacy systems and their associated models often evolve into large monolithic artifacts. This jeopardizes human understandability and maintainability. Such discomfort can be more evident for LCDP users like citizen developers with potentially limited IT backgrounds.

In this deliverable, we aim to extend the approach presented in [4] by introducing the initial research efforts spent to apply the Reinforcement Learning approach, currently under development, to address the modularization problem and present a running example of ER models.

The rest of this section is organized as follows. Section 5.1 provides a quick background on modularization and and section 5.2 describes the common terms used in reinforcement learning (RL). Then Section 5.3 maps the modularization problem into an RL problem. Section 5.4 presents a preliminary example applying the modularization approach to an ER model. Finally, Section 5.5 reports on the ongoing development activities toward tool support for the proposed modularization approach.

5.1 Modularization at a Glance

"The most common way of reducing complexity of large systems is to divide them into smaller parts or subsystems: This is called modularization [50]." Decomposing monoliths into modular structures is an established practice in software engineering. Recently, modularization has been applied in conceptual modeling and ontology development aiming to improve comprehension [51, 52, 53]. Some previous work aimed at adapting modularization to *conceptual data models* as well. Moreover, research gaps exist originating from the absence of openly available tool support. In [4], Bork et al. presented an approach for the modularization of ER models supported by the ModulER tool. This open-source Sirius-based graphical editor applies genetic algorithms (GA) to modularise ER models. Modularization is a very complex task that requires automation as the number of possible ways to modularize a system grows exponentially with the increase of the number of elements of that system. For a model with 10 modulable elements, there are already 115,975 alternative modularizations [4]. The number of all possible modularizations can be calculated mathematically as follows [54]:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

5.2 Reinforcement Learning at a Glance

Reinforcement Learning (RL) is an area of Machine Learning used for teaching a computer program (an agent) how to achieve a certain goal inside environments that require sequential decision-making [55, 56, 57]. RL can solve problems in environments with certain or uncertain rules. Recently, there have been many advances in the field of RL in various domains, such as robotics and game-playing. There are many policies to use inside the RL program [55, 56, 57]. Q-Learning is a commonly-used method in RL that maintains a table containing the reward for each of the agent actions in each possible state, and it is hence suitable for a small number of states and actions. Deep Q-Learning can address larger state spaces by replacing the reward table with a neural network (NNET) that learns the reward for each action in each possible state. The state space for model decomposition is generally very large because of the number of possible mappings of elements to modules. To handle such a large state space, we apply Deep Q-learning.

We briefly list the common terms in Deep Q-Learning [56]:

- *Agent*: It is the part of the RL program that takes actions in each state of the problem based on some policy.
- *Environment*: Any external process the agent interacts with is called the environment.
- *Game*: The problem being solved by Q-Learning. This problem can have different states, and the agent can take some actions throughout the game.
- *Action*: By taking an action, the game will enter another state. The action taken in each step of the game is determined by the output of the NNET.
- *Episode*: A game played from the beginning to the end.
- *State*: Each different game configuration occurring throughout a game. The state of the game's current step is the input of the NNET.

- *Reward Function*: Function that computes the immediate value given by a certain action in a certain state.
- *Q-value*: It estimates the cumulative expected reward from a certain state to the end of the game. The reward function only considers the next action, while Q-Value considers the whole path from the current state to the game's final state. This ideal function is what the NNET tries to learn internally during training.

The agent is the learner and decision-maker. Any external process the agent interacts with is called the *environment*. The agent and the environment interact continuously. The agent chooses actions, and the environment responds to these actions and calculates the next state based on the current state and the action taken. The environment also gives rewards to the agent, and the agent tries to maximize the overall cumulative reward through its choice of actions. Figure 31 shows the interaction between the agent and the environment in Deep Q-Learning problems [56]. The agent receives the current State (S_t) and the current reward (R_t) from the environment, then it chooses an action (A_t) and sends it to the environment. Finally, the environment starts the same process for the next state (S_{t+1}) and the next (updated) reward (R_{t+1}).

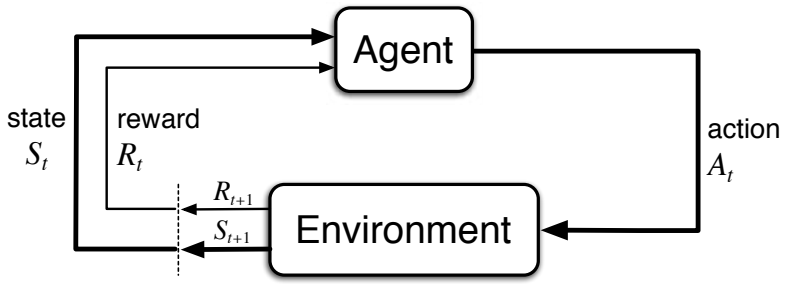

Figure 31: The agent–environment interaction in a Deep Q-Learning problem [56]

5.3 Mapping the modularization problem into an RL problem

To use RL to tackle the modularization problem, we need to map it to a standard RL problem, i.e., we have to redefine the RL terms specifically for our problem. In this scenario, we define the **game** as modularizing the monolithic ER model into smaller modules. The **game** needs an initial **state**, some middle **states**, and a final **state**. We define the **states** as below:

- *Initial State*: Each model element is put separately inside a module. So we have the same number of modules as the number of model elements.
- *Middle States*: The **agent** tries merging two of the modules to increase the quality of the model. The merging is called an **action**, and the quality metric is called the **reward function**.
- *Final State*: When we have merged all of the modules, we have a monolithic model again. This is called the final **state**.

In games as such, the model with the best quality is usually a middle **state**, not the final one. The **reward function**, which measures the quality of the model, is defined as the *Modularization Quality* (MQ-Index) [58] of the model. The MQ evaluates the balance between coupling and cohesion by combining them into a single measurement. Because MQ is a well-studied objective function, we choose it as the **reward function**.

To define MQ, first, we need to define the modularization factor (MF_k) of each module k. MF_k can be defined as:

$$MF_k = \begin{cases} 0, & \text{if } i = 0, \\ \frac{i}{i+\frac{1}{2}j}, & \text{if } i > 0, \end{cases}$$

In the formula above, i represents the number of intra-edges and j represents the number of inter-edges. The intra-edges are those for which the source and target of the edge lie inside the same module. The inter-edges are those for which the two ends of the edge lie in distinct modules. The reason for the occurrence of the term $\frac{1}{2}j$ in the above equation (rather than merely j) is to split the penalty of the inter-edge across the two modules that are connected by that edge [58].

The MQ can be calculated in terms of MF as:

$$MQ = \sum_{k=1}^{n} MF_k$$

where n is the number of clusters.

The goal of MQ is to limit, but not eliminate, excessive coupling. So, naively thinking that coupling is bad, a "perfect" solution would contain a single module containing all elements. Such a solution has no coupling. However, this is not an ideal solution. This is because modules do not have the best possible cohesion. The MQ measurement attempts to balance coupling and cohesion by combining them into one measurement. The values produced by MQ can be arbitrarily large because the value is the total number of modules present in the solution and the MQ function is not a metric one. The aim is to reward increased cohesion with higher MQ scores and penalize increased cohesion with lower MQ scores.

5.4 Example

The RL agent knows nothing about what actions to take in the first episode, so it randomly takes actions. After each episode, the agent updates itself based on the reward or punishment it got during that episode. After many episodes, the agent knows what action would eventually result in better rewards and takes that action in each step. Here we show how a properly trained RL agent takes actions in each step during a complete episode of modularizing a simple ER model. Figure 32 shows this simple ER model that we want to modularize.

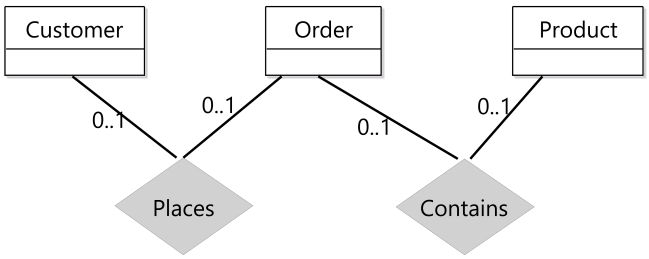

Figure 32: A simple ER model

The first step would be to put each element in a separate module. So we would get Figure 33, which contains 5 modules.

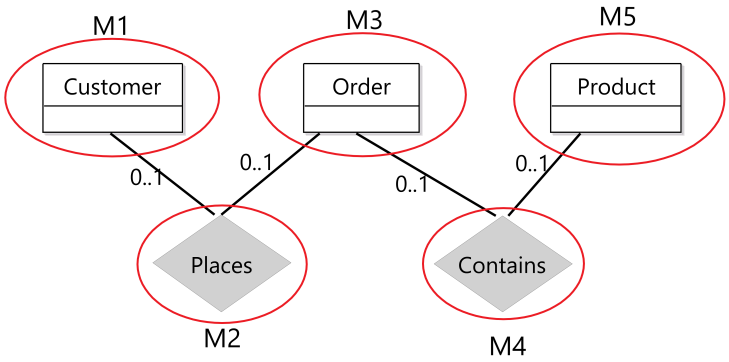

Figure 33: Step 1 of episode

$$MF_1 = 0, MF_2 = 0, MF_3 = 0, MF_4 = 0, MF_5 = 0$$
$$MQ = 0$$

In the next step, the agent decides to merge M1 and M2. This action would bring us to Figure 34. The Modularization Factor of each module is as below:

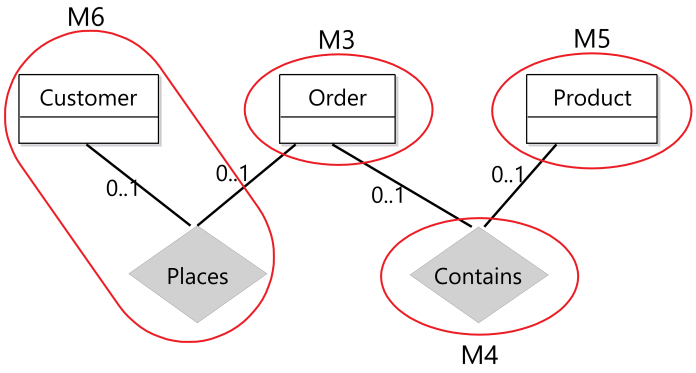

Figure 34: Step 2 of episode

$$MF_6 = \frac{1}{1+\frac{1}{2}} = 0.67, MF_3 = 0, MF_4 = 0, MF_5 = 0$$
$$MQ = 0.67$$

Step 3 would merge M4 and M5, resulting in Figure 35.

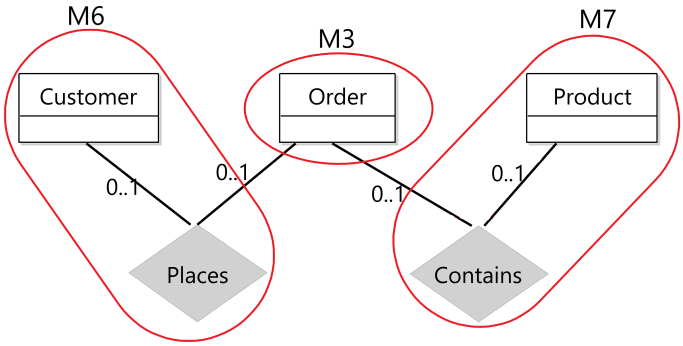

Figure 35: Step 3 of episode

$$MF_6 = \frac{1}{1+\frac{1}{2}} = 0.67, MF_3 = 0, MF_7 = \frac{1}{1+\frac{1}{2}} = 0.67$$
$$MQ = 1.34$$

In step 4, M6 and M3 will be merged to form M8. Figure 36 depicts the result.

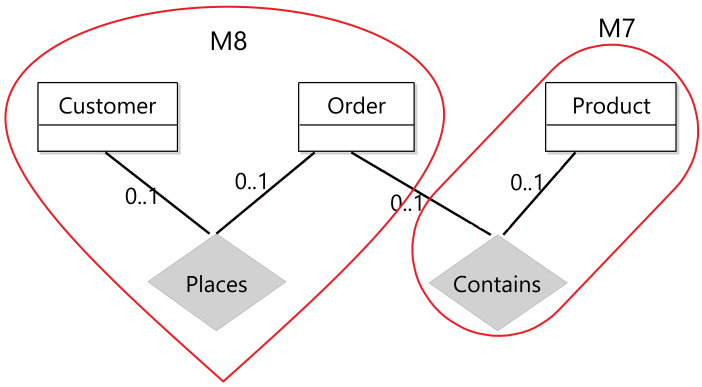

Figure 36: Step 4 of episode

$$MF_8 = \frac{2}{2+\frac{1}{2}} = 0.8, MF_7 = \frac{1}{1+\frac{1}{2}} = 0.67$$
$$MQ = 1.47$$

In the final step, the remaining modules, M8 and M7, are merged so that we would once again have the monolithic model we started from. Figure 37 shows this last step.

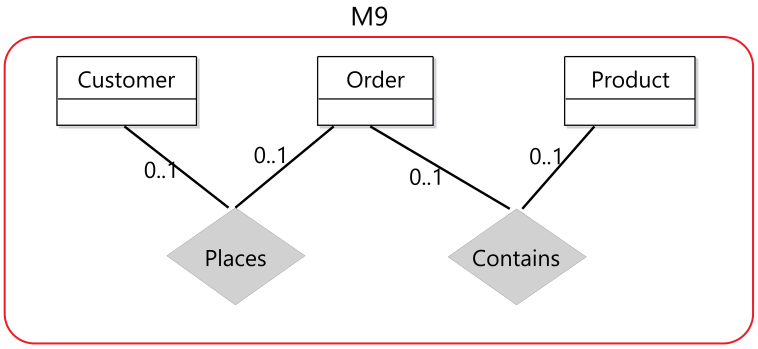

Figure 37: Step 5 of episode

$$MF_9 = \frac{4}{4+\frac{0}{2}} = 1$$
$$MQ = 1$$

Notice that the model with the highest MQ (Modularization Quality) is at step 4. In the training phase, the agent repeats similar episodes as the one shown above to learn how to reach a higher MQ.

It is worth noting that at each step, the agent is presented with an estimation of how good taking each possible action is. This estimation comes from the neural network, which produces more accurate estimations over time after many training episodes. In Deep Learning environments, problem space exploration occurs in the training phase rather than the execution phase. What was presented in this subsection is the execution phase.

5.5 Ongoing Work

The RL program needs a set of ER models to train on to modularize any given ER model after the training. For this purpose, we have not found any good ER model data set. To overcome this limitation, we reuse an existing SQL schema data set[29] and developed a text-to-model transformation from DBML to ER to transform relational database schemas presented in DBML into ER models. Then, we performed the training on the generated ER models. The text-to-model transformation from DBML to ER for this has been developed using Xtext[30]. The transformation and the ER models dataset, which are the result of the execution of the transformation, are available on a GitHub repository[31]. The Reinforcement Learning program is under development, and the prototype can be accessed via the GitHub repository here[32].

[29] https://drawsql.app/templates
[30] https://www.eclipse.org/Xtext/
[31] https://github.com/lowcomote/DBML-to-ER
[32] https://github.com/hadiDHD/ER-RL

6 Conclusion and Future Work

This deliverable represented different cloud-based services for Low-code development platforms. The first model reuse service enables model recommendations by persisting heterogeneous models on a knowledge graph and using N-grams for predicting the next modeling step by recommending further elements of interest. The second service is a REST API that enables the reuse of different models or model elements by processing natural language requests. The third is a process mining service that captures the user interaction events in a log file that can be later used to mine a process model for different purposes (e.g., model recommendations). The deliverable concludes by presenting the ongoing efforts to define and implement an RL-based approach for the modularization of large monolithic models. Such an approach can contribute to model reuse by having meaningful chunks of model elements to reuse instead of having to reason on single elements in isolation.

In future work, we aim to combine any of the above-mentioned state-based services with the operation-based mining one to provide tailored-made suggestions to the user during the modeling process by considering the users' logs and the state of the models. We aim to generate probabilistic graph models that abstract the structural changes done on a model as operational data for each user.

Moreover, we aim to investigate the potential contributions and challenges of *process mining-as-s-service* via the MPMT tool. It may represent a possible horizontal lightweight integration mechanism across knowledge bases collected from heterogeneous LCDPs in terms of targeted application domains and (modeling) technologies. Nevertheless, process mining-as-a-service, by collecting and analyzing user-related data, is expected to raise privacy and intellectual property challenges.

Concerning the modularization of large monolithic models, we are considering the adoption of NLP techniques besides the MQ reward to improve the quality of the proposed modularization by considering entities', attributes', and relations' names. Finally, as already proposed for model reuse and process mining, we are considering offering the modularization approach as a service and combine this service with other services such as for model versioning [59] which would allow to use additional information for the modularization process.

References

[1] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-Driven Software Engineering in Practice: Second Edition. *Synthesis Lectures on Software Engineering*, 2017.

[2] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 2022.

[3] Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis. Licensed for Distribution Magic Quadrant for Enterprise Low-Code Application Platforms. pages 1–34, 2019.

[4] Dominik Bork, Antonio Garmendia, and Manuel Wimmer. Towards a multi-objective modularization approach for entity-relationship models. In *ER Forum/Posters/Demos*, pages 45–58, 2020.

[5] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso 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.

[6] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019.

[7] Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, (Lcd), 2022.

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

[9] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan De Lara, Esther M Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel 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 co-located with Software Technologies: Applications and Foundations (STAF 2019)*, CEUR Workshop Proceedings (CEUR-WS.org), Eindhoven, Netherlands, July 2019.

[10] 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. Opportunities in intelligent modeling assistance. *Software and Systems Modeling*, (June), 2020.

[11] Jimmy Lin. N-Gram Language Models N-Gram Language Models. 2009.

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

[13] Adam Grzech, Leszek Borzemski, Jerzy Świątek, and Zofia Wilimowska. Preface. *Advances in Intelligent Systems and Computing*, 430(October):V–vi, 2016.

[14] Amitabha Bhattacharyya and Durgapada Chakravarty. (Graph Database: A Survey). *2020 International Conference on Computer, Electrical and Communication Engineering, ICCECE 2020*, 2020.

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

[16] Bob DuCharme. *Learning SPARQL (free chapters)*. 2013.

[17] 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.

[18] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. MoDisco: a Model Driven Reverse Engineering Framework. *Information and Software Technology*, 56(8):1012–1032, August 2014.

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

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

[21] Cyril Goutte and Eric Gaussier. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. *Lecture Notes in Computer Science*, 3408(April):345–359, 2005.

[22] Steven M. Beitzel, Eric C. Jensen, and Ophir Frieder. *MAP*, pages 1691–1692. Springer US, Boston, MA, 2009.

[23] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Riccardo Rubei. Development of recommendation systems for software engineering: the CROSSMINER experience. *Empirical Software Engineering*, 26(4), 2021.

[24] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Alfonso Pierantonio. MemoRec: a recommender system for assisting modelers in specifying metamodels. *Software and Systems Modeling*, 2022.

[25] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Phuong 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.

[26] Mora Segura Ángel, Juan de Lara, Patrick Neubauer, and Manuel Wimmer. Automated modelling assistance by integrating heterogeneous information sources. *Computer Languages, Systems and Structures*, 2018.

[27] Ángel Mora Segura and Juan de Lara. EXTREMO: An Eclipse plugin for modelling and meta-modelling assistance. *Science of Computer Programming*, 2019.

[28] Loli Burgueño, Robert Clarisó, Shuai Li, Sébastien Gérard, Jordi Cabot, Loli Burgueño, Robert Clarisó, Shuai Li, Sébastien Gérard, and Jordi Cabot An 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.

[29] Thibaut Capuano, Houari Sahraoui, Benoit Frenay, and Benoit 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).

[30] Henning Agt and Ralf-Detlef Kutsche. Automated construction of a large semantic network of related terms for domain-specific modeling. In *CAiSE*, 2013.

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

[32] Nick Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, Boston, MA, 2009.

[33] Rijul Saini, Gunter Mussbacher, Jin L.C. Guo, and Jörg Kienzle. Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling*, 2022.

[34] Sara Pérez-Soler, Gwendal Daniel, Jordi Cabot, Esther Guerra, and Juan de Lara. Towards automating the synthesis of chatbots for conversational model query. *Lecture Notes in Business Information Processing*, 387 LNBIP:257–265, 2020.

[35] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. Recommending metamodel concepts during modeling activities with pretrained language models. *Software and Systems Modeling*, 2022.

[36] Daniel Lucrédio, Renata Fortes, and Jon Whittle. Moogle: a metamodel-based model search engine. *Software & Systems Modeling*, 11:183–208, 2010.

[37] José Antonio Hernández López and Jesús Sánchez Cuadrado. Mar: A structure-based search engine for models. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, page 57–67, New York, NY, USA, 2020. Association for Computing Machinery.

[38] Euripidis Loukis, Marijn Janssen, and Ianislav Mintchev. Determinants of software-as-a-service benefits and impact on firm performance. *Decision Support Systems*, 117:38–47, 2 2019.

[39] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The prom framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005*, pages 444–454, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[40] Eclipse sirius documentation. https://www.eclipse.org/sirius/doc/. Accessed: 2022-03-23.

[41] Francesco Bedini, Ralph Maschotta, and Armin Zimmermann. A generative approach for creating eclipse sirius editors for generic systems. In *2021 IEEE International Systems Conference (SysCon)*, pages 1–8, 2021.

[42] Sven Jäger, Ralph Maschotta, Tino Jungebloud, Alexander Wichmann, and Armin Zimmermann. Creation of domain-specific languages for executable system models with the eclipse modeling project. pages 1–8, 04 2016.

[43] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of dsm graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238, 2014.

[44] Tanja Mayerhofer, Manuel Wimmer, Luca Berardinelli, and Rainer Drath. A model-driven engineering workbench for caex supporting language customization and evolution. *IEEE Transactions on Industrial Informatics*, 14:2770–2779, 6 2018.

[45] Antonio Garmendia, Manuel Wimmer, Alexandra Mazak-Huemer, Esther Guerra, and Juan de Lara. Modelling production system families with automationml. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1057–1060, 2020.

[46] Eclipse theia. https://projects.eclipse.org/proposals/eclipse-theia. Accessed: 2022-07-11.

[47] Eclipse glsp. https://www.eclipse.org/glsp/. Accessed: 2022-07-11.

[48] Emf cloud. https://www.eclipse.org/emfcloud/. Accessed: 2022-07-11.

[49] John Mylopoulos. Conceptual modelling and telos. conceptual modelling, databases, and case: An integrated view of information system development. *New York: John Wiley & Sons*, 49:68, 1992.

[50] Daniel Moody. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on software engineering*, 35(6):756–779, 2009.

[51] Antonio Garmendia, Esther Guerra, Juan de Lara, Antonio García-Domínguez, and Dimitris Kolovos. Scaling-up domain-specific modelling languages through modularity services. *Information and*

Software Technology, 115:97–118, 2019.

[52] Tuğba Özacar, Övünç Öztürk, and Murat Osman Ünalır. Anemone: An environment for modular ontology development. *Data & Knowledge Engineering*, 70(6):504–526, 2011.

[53] Michaël Verdonck, Maria Hedblom, Giancarlo Guizzardi, Guylerme Figueiredo, Frederik Gailly, and Geert Poels. An empirical study concerning ontology-driven modularization and ontology-neutral modularization techniques. In *Proc. of the 13th Int. Workshop on Value Modelling and Business Ontologies*, 2019.

[54] Martin Fleck, Javier Troya Castilla, and Manuel Wimmer. The class responsibility assignment case. *TTC 2016: 9th Transformation Tool Contest, co-located with the 2016 Software Technologies: Applications and Foundations (STAF 2016)(2016), p 1-8*, 2016.

[55] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey, 2020.

[56] Richard S Sutton and Andrew Barto. *Reinforcement learning: an introduction*. The MIT Press, 2018.

[57] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model, 2020.

[58] Kata Praditwong, Mark Harman, and Xin Yao. Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2):264–282, 2011.

[59] Gabriele Taentzer, Claudia Ermel, Philip Langer, and Manuel Wimmer. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Softw. Syst. Model.*, 13(1):239–272, 2014.