Democratizing the Development of Recommender Systems by means of Low-code Platforms

Conference Paper · September 2020

DOI: 10.1145/3417990.3420202

3 authors:

Claudio Di Sipio
Università degli Studi dell'Aquila
21 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Davide Di Ruscio
Università degli Studi dell'Aquila
200 PUBLICATIONS **2,697** CITATIONS
SEE PROFILE

Phuong Nguyen
Università degli Studi dell'Aquila
54 PUBLICATIONS **359** CITATIONS
SEE PROFILE

Democratizing the Development of Recommender Systems by means of Low-code Platforms

Claudio Di Sipio, Davide Di Ruscio, Phuong T. Nguyen
Università degli studi dell'Aquila, Via Vetoio 2, 67100 – L'Aquila, Italy
claudio.disipio@graduate.univaq.it,davide.diruscio@univaq.it,phuong.nguyen@univaq.it

ABSTRACT

In recent years, recommender systems have gained an increasingly crucial role in software engineering. Such systems allow developers to exploit a plethora of reusable artifacts, including source code and documentation, which can support the development activities. However, recommender systems are complex tools that are difficult to personalize or fine-tune if developers want to improve them for increasing the relevance of the retrievable recommendations.

In this paper, we propose a low-code development approach to engineering recommender systems. Low-code platforms enable the creation and deployment of fully functional applications by mainly using visual abstractions and interfaces and requiring little or no procedural code. Thus, we aim to foster a low-code way of building recommender systems by means of a metamodel to represent the peculiar components. Then, dedicated supporting tools are also proposed to help developers easily model and build their custom recommender systems. Preliminary evaluations of the approach have been conducted by reimplementing real recommender systems, confirming the feasibility of developing them in a low-code manner.

KEYWORDS

Recommendation systems, Low-code platforms, Modeling

ACM Reference Format:
Claudio Di Sipio, Davide Di Ruscio, Phuong T. Nguyen. 2020. Democratizing the Development of Recommender Systems by means of Low-code Platforms. In *Proceedings of The First LowCode Workshop (LowCode 2020).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, software development activity has reached a high degree of complexity, led by the heterogeneity of the components, data sources, and tasks. To address such challenges, suitable machinery is needed to transform raw data into practical knowledge that can really help developers with their programming tasks. The introduction of recommender systems in software engineering [24] grants to assist developers in navigating large information spaces and getting instant recommendations that might be helpful to solve the particular development problem at hand. Thus, a recommender system in software engineering (RSSE) aims at giving developers recommendations, which can consist of different items including code examples, issue reports, reusable source code, possible third-party components, documentation, to name a few [24].

Several RSSEs are nowadays available to support software development activities by using well-defined strategies. Although the usage of such systems have attracted a lot of attention from both academia and industry, significant effort and specific know-how is needed to design, develop, and maintain RSSEs. In particular, conceiving or even adopting recommender systems tailored for the particular problem at hand can be a very hard task for developers, especially for those who do not have enough knowledge about the main building components of recommender systems and how they work [20].

In this paper, we promote the adoption of a low-code development paradigm to simplify and democratize the development of recommender systems. Low-code development [22] is a new emerging concept based on a combination of various approaches, i.e., model-driven engineering, and rapid application development. A low-code development platform (LCDP) is a system that offers user-friendly graphical interfaces, drag-and-drop utilities, allowing users to build software applications from scratch. The concept has a heavy impact on the industrial domain as enterprises are continuously looking for new development strategies that reduce costs and increase efficiency. On the other hand, the topic still remains challenging as no previous work has been dedicated to democratizing RSSEs development. Thus, our work aims at addressing the following challenges:

- *How is it possible to simplify the development of RSSEs?* Developing recommender systems can be a daunting task. We believe that an LCDP can simplify and speed up the overall process.
- *What are the modeling constructs needed to support the specification of a new RSSE?* It is necessary to investigate in detail a possible set of features and create a model capable of representing them. Taking into account existing work, we define a feature model representing all the relevant RSSE features.
- *Can such kind of systems support modeling activities?* Supporting modeling activities (e.g., metamodeling, and development of model-to-model or model-to-code transformations) could be a daunting task, as it is an activity extremely bounded with the personal developer's experience. Investigating possible applications of the LCPD to develop modeling assistants can disclose interesting and research directions.

Considering the inner complexity of the considered problem, this work provides initial results of the envisioned approach, by reimplementing well-founded recommender systems in the literature using a meta-model tailored for this domain. Though it is an initial work and thus, it needs a more comprehensive analysis,

LowCode 2020, October, 2020, Montreal, Canada

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

we believe that our work approach is capable of trigger additional future work to investigate in detail.

The paper is organized as follows: Section 2 gives an overview of low-code development in recent years. Additionally, we present successful examples of general recommender systems as well as modeling assistant tools. Section 3 introduces the proposed low-code way of developing recommender systems. A preliminary evaluation of the approach is presented in Section 4 by outlining the reimplementation of existing recommender systems by using our proposal. Section 6 concludes the paper by summarizing the contributions and the main steps that have been performed and that still need to be undertaken to realize the proposed idea.

2 MOTIVATION AND BACKGROUND

This section presents an overview of existing recommender systems tailored for supporting the development of software systems (see Section 2.1) and the ones specifically conceived for supporting modeling activities (Section 2.2). Section 2.3 presents the low-code applications landscape to identify peculiar aspects that can be exploited to support the development of recommender systems.

2.1 Recommender systems to support software development

Holmes *et al.* [7] propose Strathcona, a recommender system that analyzes the developer's context from the structural point of view and suggests a possible implementation related to the current development task. The recommendation algorithm is based on six heuristics that consider inheritance hierarchy, field types method calls, and object usage to build the query. Strathcona executes the recommendation algorithm on a local repository and retrieves code examples, which can be navigated by the developer both in a graphical and in a textual way.

Sourcerer [12] performs code search in large-scale repositories by exploiting different components. The first component is the crawler, which automatically downloads repositories to build a knowledge base. Then, it parses the source code to represent it as a database entity. Additionally, Apache Lucene and fingerprint are used to support respectively the keyword-base search and structural representation of the repository. Finally, the ranker retrieves the most relevant results according to these strategies

Recently, StackOverflow has been exploited to enrich code queries, with the ultimate aim of getting relevant source code. In particular, FaCoY [9] is a code-to-code search engine that recommends relevant GitHub code snippets to a project being developed. It is based on an alternate query technique to augment the possible retrieved results. The initial query is built from StackOverflow posts and the additional query is performed directly on GitHub local repositories to deliver final recommendation items. PROMPTER [21] is an automatic tool to search and retrieve recommendations coming from StackOverflow. The system crawls data from Google, considering the developer's context, i.e., the source code being developed. Starting from this data, PROMPTER builds a query by parsing raw data using an island parser developed by the authors. Then, the system sends the created query to the Google and Bing search engines. The retrieved posts are arranged and shown to the developer according to a custom ranking model.

Differently from the previously reviewed work that proposes tools being able to provide developers with specific recommendations, Korotaev *et al.* [10] introduce a GRU-based recurrent neural network (RNN) aiming to build a universal recommender system. To this end, the approach supports the recommendation phase by means of a client-server architecture equipped with different components. The data collection and processing modules are conducted, taking into consideration the user's behavior. The data mining module is used to feed a GRU-based RNN. To support user profiling, the approach uses ontologies to build an external knowledge representation module. The proposed network outperforms the long short-term memory (LSTM) standard technique with respect to accuracy.

2.2 Modeling assistants

With the same spirit of recommender systems in software engineering, over the last few years, also in the modeling domain some efforts have been spent to develop modeling assistant tools with the aim of supporting developers during their modeling activities as overviewed in the following.

The Extremo tool [15] has been proposed to assist modelers in an agnostic fashion. It is able to handle heterogeneous resources by exploiting a common data model created by analyzing the modeling context. Then, a flexible query mechanism is used to explore and find useful entities that help the modeler to complete its model under construction. The tool has been successfully integrated as an Ecplise plugin and it is able to interact with the whole modeling environment. To assess the quality of the work, Extremo has been used to implement a DSL for the financial domain.

Dupont *et al.* [6] propose a Papyrus plugin to support domain-specific modeling by relying on UML profiles. The tool makes use of the EMF generator model to map each meta-class of the profile to a real Java class in order to manipulate them. Additionally, the definition of a palette and a context menu is required to manipulate the elicited meta-classes using the plugin. Even though the proposed tool works in practice, there is possible room for improvement i.e., proactive triggering of recommendations or fine-grained customizations using EMF utilities.

A modeling assistant based on a multi-objective optimization problem (MOOP) has been proposed in [1]. Starting from a set of metamodels together with relevant information, the tool is able to discover a set of representative input models using a well-founded evolutionary algorithm, called NSGA-II. Such an algorithm is employed to solve the MOOP by using the definition of Pareto optimality concept to find relevant candidates. Then, the retrieved partial models need to be completed by expert-domain users by selecting different coverage degree or pre-defined minimality criteria. The quality of the tool has been evaluated by comparing the proposed NSGA-II adaptation with random and mono-objective functions. The results show that the proposed methodologies overcome the other function in terms of performances.

In [14], the authors show an example-driven tool that is able to recommend a complete meta-model starting from model fragments sketched by means of graphical tools i.e., Visio, PowerPoint, or Dia. From these initial examples, the tool is able to elicit untyped model fragments. Then, a neutral meta-model is automatically induced

from such pieces and the modeler can enrich it by adding manually new elements or by relying on the provided virtual assistant. The constructed model is validated by using pre-defined test cases and the modeler can select a target platform to compile it.

2.3 Low-code application landscape

Low-code development platforms (LCDPs) are easy to use environments that are being increasingly introduced and promoted by major IT players to speed up the development of software applications to face the typical time constraints in a company domain. The key benefits of this kind of approach concern the increase of digital innovation, easy development, and cost reduction during the entire process. The overall objective of the kind of platform is to assist the so-called *citizen developers*, who have little or no development skills at all.

A report provided by the Forrest Industry [23] highlights the main LCPD market segments and vendors as well as ossible future evolution in this field. Besides the aforementioned benefits and advantages, low-code development requires at least medium level programming skills to properly develop the requested application as well as design process experiences to integrate the different tasks. Concerning the market segments, they are divided by considering the type of application and expected goals i.e., *general-purpose*, *process apps*, *database*, *request-handling*, and *mobile apps*.

The majority of LCDPs cover the *general-purpose* applications and supports several types of software systems as well as their lifecycle management [26]. Platforms for *process apps* aim to handle collaboration and coordination among different people in very large companies. LCDPs for the *database* segment allow us to handle, manipulate, and retrieve data from relational databases in a user-friendly manner without handling the entire application workflow. *Request-handling* low-code platforms are used to handle custom processes for different departments within a company and to manage requests for services. Similarly to process app development, these platforms must address the integration among various entities and provide self-service functionalities. LCDPs for developing *mobile apps* include strongly dedicated functionalities i.e., mobile middleware and different widget. Concerning future trends, the segment of the mobile app development will disappear [23]. In contrast, a promising field for low-code development is the IoT domain.

3 PROPOSED APPROACH

To support the development of RSSE for citizen developers, in this section we present initial ideas about the possibility of conceiving a supporting low-code platform. The users will be provided with a graphical environment to declaratively specify the main behavioral components, which need to be employed and composed to realize the wanted recommendations. To this end, we distill the peculiar design features implemented by existing recommender systems in software engineering (see Section 3.1). To perform such feature elicitation phase we rely on the knowledge we gained in the context of the EU CROSSMINER project[1] to develop different kinds of recommendation systems as discussed in Section 3.2.

The performed domain analysis phase has been done by following an iterative process consisting of the following main steps: *i)* first of all, we identify existing paradigms and features from well-founded RSSE already discussed in Section 2.1; *ii)* subsequently, a corresponding metamodel is defined to represent all the elicited features (see Section 3.3). Such a metamodel is iteratively refined with the aim of formalizing all the modeling constructs that are needed to properly model the recommender systems systems under analysis.

The performed domain analysis is preparatory to identify the reusable building blocks that users can drag and drop in the graphical environment provided by the envisioned low-code platform (see Section 3.4).

[1] https://www.crossminer.org/

3.1 Design features of recommender systems

According to Robillard *et al.* [24], any recommender system in software engineering consists of four main activities:

Data Preprocessing In this phase techniques and tools are applied to extract valuable information from different data sources, e.g., AST parsing, text mining, and natural language processing;

Capturing Context Subsequently, the developer context is excerpted from the programming environment to create the recommendation query. Depending on the RSSE, it can be triggered pro-actively or reactively;

Producing Recommendations The actual recommendation algorithms are chosen and executed to produce hints that are relevant for the current development task. Possible recommendation approaches that can be employed at this stage can be for example model-based, memory-based, make use of filtering techniques, or based on some domain-specific heuristics;

Presenting Recommendations As the last phase, the produced recommendation items are to be presented to the developer for instance directly in the IDE being used or in an external Web application.

By adhering to the principles and the methods underpinning the development of recommender systems [3, 8, 11], we have further investigated such high-level phases with the aim of defining interrelated *features* [2] as shown in Fig. 1. For each activity (represented as boxes), corresponding features (and sub-features) are depicted as ovals to represent possible techniques that can be employed. Due to space limit, the model does not represent the relationships among features as well as possible constraints governing their usage. For the same reason, we describe only the main techniques' categories belonging to the fundamental components of a recommender system.

Concerning the data preprocessing phase, we reported the most common ones used in software development i.e., techniques that mine source code, documentation, and software projects. Such strategies have been excerpted both from existing RSSE as well as recommender systems that we have actually implemented [5, 16–19, 25] in the context of the CROSSMINER project. According to the peculiar nature of data sources, one technique is more suitable rather than another. For instance, *AST parsing* and *Fingerprints* works properly only with structured data that are compliant with well-defined rules. In contrast, unstructured data such as text plain

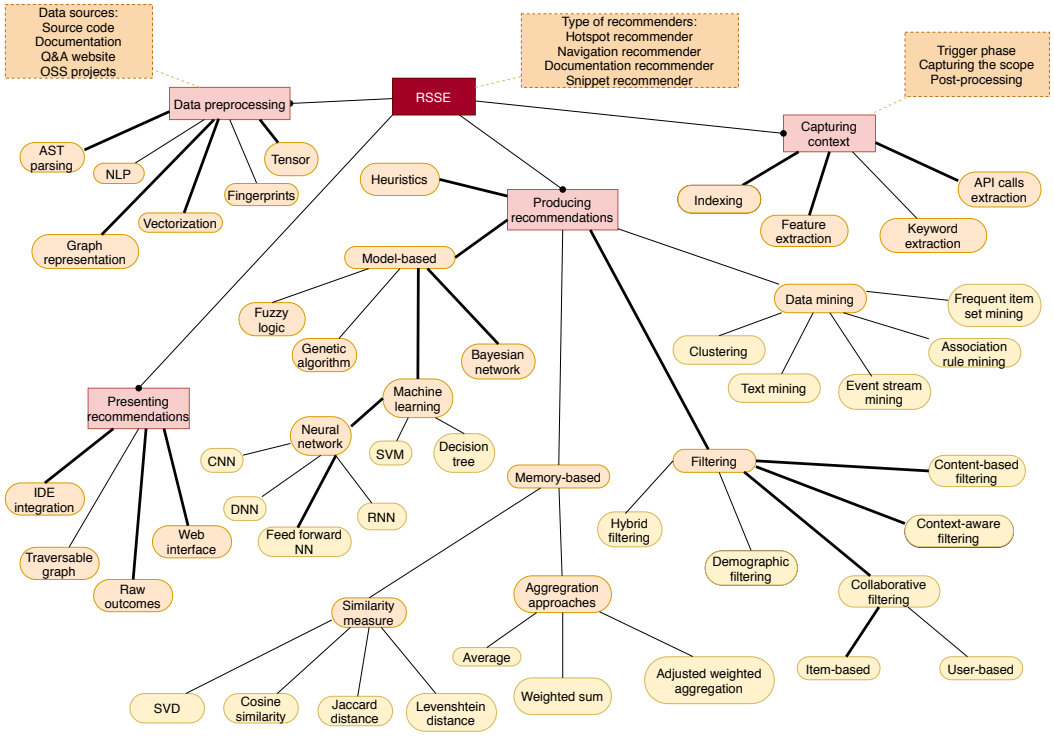

Figure 1: Feature model excerpted from recommender systems in software engineering.

Recommendation system	Data preprocessing	Capturing context	Producing recommendation	Presenting recommendation
Strathcona [7]	AST parsing	Keyword extraction	Six heuristic functions	IDE integration
Sourcerer [12]	AST parsing	Indexing	Custom ranking scheme (Heuristic)	Web interface
FaCoY [9]	AST parsing	Indexing	Alternate query (Heuristic)	Web interface
PROMPTER [21]	NLP	Indexing	Custom ranking model (Heuristic)	IDE integration
CrossSim [18]	Graph representation	Feature extraction	Content-based filtering	Web interface
CrossRec [19]	Graph representation	Feature extraction	Collaborative filtering	IDE integration
FOCUS [16]	Tensor	API calls extraction	Context-aware collaborative filtering	IDE integration
AURORA [17]	NLP	Feature extraction	Feed forward neural network	Web interface
MNB [5]	Vectorization	Feature extraction	Bayesian Network	Raw outcomes
PostFinder [25]	AST Parsing	Indexing	Heuristics	IDE integration

Table 1: Characterizing recommender systems with respect to features.

files should be analyzed using different techniques i.e., *NLP* or *Vectorization*. Data sources that exhibit mutual relationships can be analyzed using *Graph representation* and *Tensor* data representation components.

Immediately after the data preprocessing, the context of recommendation is excerpted from the developing environment to foster the next phases. Capturing context often involves the search over big software projects. In this specific case, indexing is a technique mainly used by the code search engines to retrieve relevant elements in a short time. When the context is limited to a single software project, the API calls extractor is recommended i.e., it can excerpt valuable information in a shorter time.

Producing recommendations phase can be supported by a plethora of strategies, that we are able to group in five main classes as depicted in the feature model. *Data mining* techniques are usually employed in the presence of a huge amount of raw data. The overall objective is to find mutual relationships and use them to retrieve the actual recommendations. If the users' preferences are available, *Filtering* strategies can play the role of the recommendation algorithm by means of items and corresponding ratings. Depending on the type of filtering technique, historical data or users' behavior could be taken into consideration. By relying on matrixes, *Memory-Based* strategies compute the similarity using several well-founded algorithms to minimize the distance among the considered elements i.e., Jaccard distance, Cosine similarity to name a few. A possible limit of such strategies is that they require direct usage of the input data, unavailable user certain conditions. To cope with this issue, *Model-based* approaches have been successfully employed in the literature. The underpinning strategy of these strategies makes use of a model that is directly built from the data itself. Although all the abovementioned techniques work in practice, they suffer from some limitations led by their peculiar nature. Thus, *Heuristics* techniques were born to overcome such constraints by encoding the knowhow of domain experts in the recommendation algorithm. The main advantage is that they reduce the development effort

by avoiding complex data structures. Despite this, they may reach sub-optimal results compared to more sophisticated technique and they should be carefully selected considering the context of the recommendations.

Finally, the recommendation items have to be properly presented to the final user according to the nature of the outcomes. To this end, *IDE integration* offers several advantages, especially for support the modeling activity. As shown in [6], a plugin is the most common methodology to realize such an integration even though the deployed artifacts have to be carefully integrated to avoid possible incompatibilities with components that are already in place. Using the recommendation system in a stand-alone fashion is a more maintainable solution as the developers don't have to cope with the mentioned compatibility issues. Relying on *Web interface*s represents a spreadly technique that falls in this category. Besides the strengths, presenting recommendations through this strategy must consider several issues that commonly arise i.e., failure in server connections, and granting suitable response times to name a few. Interactive data structures might be employed in navigating the recommended items in a simpler way i.e., no integration effort is required at all. *Traversable graph* is just one successful example of this category that allows the visualization of additional details about the retrieved outcomes. Due to certain constraints, all the mentioned presenting techniques could be not applicable or available. Thus, presenting *Raw outcomes* should be necessary even though it is not the best possible solution. Nevertheless, the informative value of the items retrieved in such a way can be improved by supporting them with a detailed description as well as possible use cases.

The elicited feature model exhibits also dashed boxes that represent attributes of the mandatory features to include more information about them, e.g., data preprocessing can be employed on source code, Q&A items like StackOverflow posts, etc.

3.2 Preparatory work

We have partially realized the paradigm presented in Fig. 1 in a series of recent work [16–19]. In particular, various components, which are listed in the gray rows of Table 1, have been properly implemented and evaluated using real-world datasets (marked as the bold edges in Fig. 1). Due to space limit, the components are briefly recalled in the succeeding paragraphs. Interested readers are kindly referred to the corresponding cited work for a greater detail.

We analyzed the recommendation systems described in Section 2 and characterized them using the elicited features properly. For instance, CrossSim [18] retrieves data from source code repositories and encodes the retrieved project metadata in a graph based representation. A feature extraction mechanism is employed to capture the developer's context and create the query for the subsequent phase, i.e., the recommendation production. To this end, a *content-based filtering* approach is applied to retrieve projects that are similar to that being developed. The recommendation items are shown in a dedicated *Web interface*.

We succeeded in developing CrossRec [19], a recommender system for finding relevant third-party libraries by exploiting the graph presentation. Using the developer's context as a set of libraries being included, the system utilizes a collaborative-filtering technique to select further relevant libraries from highly similar projects.

FOCUS [16] supports the development activities by recommending API function calls. After the extraction of relevant elements from the developer's source code, the tool encodes these elements in a tensor and searches for possible matching by using a context-aware collaborative-filtering technique. Eventually, it provides the developer with the list of the most relevant API function calls, which are shown directly in the Eclipse IDE.

With the aim of overcoming the limitations and easing the burden of manual categorization of metamodels, we propose AURORA [17], an automated classifier for metamodel repositories using a feed forward neural network. AURORA learns from labeled metamodels and effectively classifies incoming unlabeled ones. An empirical evaluation of dataset of 555 metamodels shows that the tool predicts a label for an arbitrary metamodel with a considerably high accuracy.

We investigate the application of Multinomial Naïve Bayesian (MNB) networks to automatically classify GitHub repositories [5]. By analyzing the README file(s) of the repository to be classified and vectorizing their content using the TF-IDF weighting scheme, the conceived approach is able to recommend GitHub topics. To the best of our knowledge, this is the first supervised approach addressing the considered problem. Consequently, since there exists no suitable baseline for the comparison, we validated the approach by considering different metrics, aiming to study various quality aspects.

In a recent work [25], we developed PostFinder – a StackOverflow posts recommender system, which is based on a two-phase approach to retrieve posts from StackOverflow by taking various measures on both the data collection and query phases. To improve efficiency, we make use of Apache Lucene to index the textual content and code coming from StackOverflow. During the first phase, posts are retrieved and augmented with additional data to make them more exposed to queries. Afterwards, we boost the context code with different factors to construct a query that contains information needed for matching against the stored indexes. In a nutshell, we make use of heuristics based on *multiple facets* of the data available at hand to optimize the search process, with the ultimate aim of recommending highly relevant SO posts.

3.3 The proposed RS metamodel

The previous phase was preparatory to identify the essential components building up a recommender system. The envisioned LCDP should provide developers that have the need for new recommender systems with the supporting tools for selecting the wanted features from those elicited in Fig. 1. Then, the selected features have to be properly configured to eventually obtain software tools implementing the wanted recommender systems. To this end, we formalized the features shown in Fig. 1 in a corresponding metamodel as shown in the fragment depicted in Fig. 2. The metamodel permits to specify input data in terms of `Datasource` and `DataRepresentation` elements. The former represents the input data that feed the system at the beginning of the recommendation process. The latter is

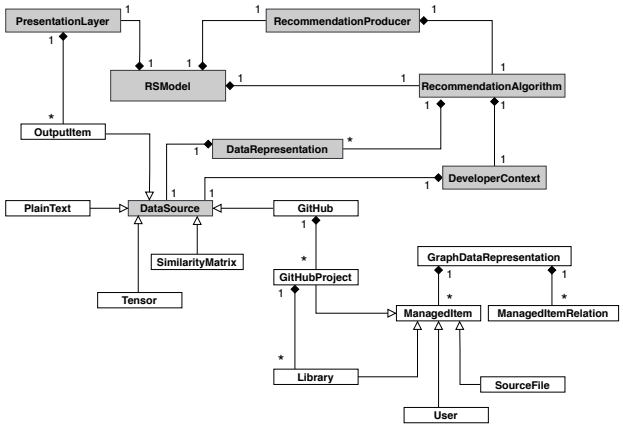

Figure 2: The proposed RS metamodel.

needed to generalize and manage different types of data sources, e.g., GitHub projects, plain text files, matrices, to name but a few.

The `RecommendationProducer` metaclass aggregates different elements that are necessary to produce the actual recommendations, i.e., `RecommendationAlgorithm` and `DeveloperContext` metaclasses. The former is required to specify the business logic of the approach, by following well-founded archetypes. The latter is used to represent the context of the developer, which can be considered as input for the wanted recommendation. The metaclass `PresentationLayer` is used to represent and deliver the retrieved recommendation items to the developer.

Such main metaclasses are represented in light-gray rectangles in Fig. 2. They are supposed to be always needed to specify any recommender system. Such metaclasses can be subsequently refined in order to cover specific technologies (see the metaclasses represented in white rectangles in Fig. 2). By focusing on the metaclass `DataSource`, the specializations that have been defined so far are the following ones:

- The `GraphDataRepresentation` metaclass is suitable to represent input data in a graph format, particularly useful for several recommendation algorithms. This entity includes a set of `ManagedItem` that encodes data sources features in a node of the graph. Relationships between two items are modeled by the `ManagedItemRelation` metaclass;
- `GitHubProject` encodes relevant information about the GitHub projects including the URL, name, and used third-party libraries. The metaclass `GitHub` is used to aggregate a set of different GitHub projects;
- To model repositories metadata, the `ManagedItem` metaclass has been specialized by means of the metaclasses `Library`, `User`, and `SourceFile`;
- Many techniques operate directly on documents. Thus, we introduce the metaclass `PlainTextRepresentation` to represent them in our model;
- A plethora of approaches make heavy usage of matrices to recommend relevant items as in the case of user-item matrices in collaborative filtering technique. Thus, two dedicated metaclasses have been defined, i.e., `Tensor` to represent n-dimensional matrices and `SimilarityMatrix` to add the concept of similarity to among different elements.

3.4 The envisioned LCDP

By relying on the modeling constructs provided by the metamodel overviewed in the previous section, the envisioned LCDP will permit users to declaratively select the behavioral components that they want to use for developing the desired recommender systems. In particular, for each feature in the model described in Section 3.1, the envisioned graphical environment provides a corresponding component, which can be selected and connected with the others. In other words, a domain specific language will be provided to specify correct configurations consisting of proper selections of the available features.

The editing phase will be guided by the platform so as to properly cover all the phases that need to be implemented. Moreover, the available behavioural components will be specified to restrict their composition with other ones according to a precise definition of input and output interfaces. For instance, if the user wants to set in place a modeling assistant tool similar to the ones presented in Section 2.2 to obtain recommendations during the modeling activity, the system will provide several ways to represent the input model (e.g., graph data structure) that are allowed to be taken asinput.

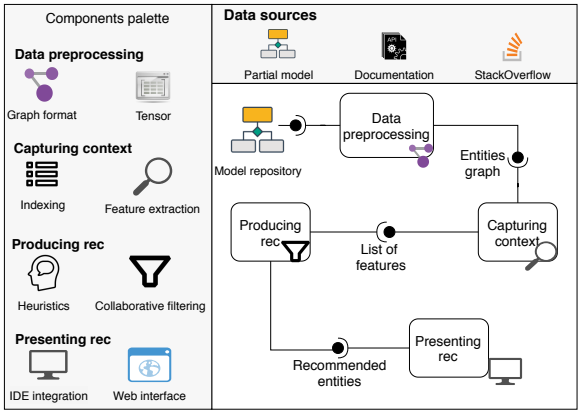

Figure 3: Mock-up of the envisioned LCD environment.

To give a flavour of the proposed platform, Fig. 3 shows a mock-up of the prospective graphical environment. All the behavioural components are organized with respect to the corresponding phase they can support (see the palette of tools on the left-hand side[2]). Different data sources supported by the platform are also made available to the user (see the upper side of Fig. 3). According to the specification given in the workspace, the user is defining a recommender system able to assist modelers that are defining (meta-) models. To this end a collaborative filtering component is employed and it is trained by means of (meta-)models that are retrieved from an available model repository. Such models are used to fed to the data preprocessing component, which encodes in a graph-based representation the valuable entities extracted from the input models. The obtained graphs are given as input to the feature extraction

[2]Only some of the potentially available features are shown

component, which in turn produces outputs that are consumed by the available collaborative filtering component. The recommendation items that will be produced at run-time depending on the user query, will be integrated into a well-founded IDE that supports modeling, i.e., Eclipse. This is just a possible scenario in which the envisioned LCDP could be used to conceive modeling assistant tools as well as general-purpose recommender systems. Each component used in the specification might require specific configurations e.g., the authentication details for the connection with the considered model repository, or the features that have to be considered for the encoding during the data preprocessing phase.

4 PRELIMNARY EVALUATION

The previously discussed metamodel is undergoing a validation process, which consists of specifying already available recommender systems in terms of the concepts defined in the RS metamodel. At this stage, the graphical environment envisioned in Fig. 3 is not available yet, as we first want to finalize the underpinning metamodel. In particular, we are reimplementing existing recommender systems in terms of models conforming to the metamodel presented in Section 3.3. To this end, we are making use of the Eclipse Modeling Framework to define the RS metamodel in Ecore and to programmatically create conforming models.

So far the CrossSim and CrossRec tools have been reimplemented and in this section we give some details about the CrossRec reimplementation. CrossRec can find relevant third-party libraries for the current developer's context, which is composed of a set of libraries to be included. Starting from this, the system utilizes a collaborative-filtering technique to select further relevant libraries from highly similar projects. By examining the current implementation of CrossRec, we refactor it by using the meta-model concepts. In particular, we model the necessary data-structures used to capture the context, i.e., the graph-based representation and the similarity matrices. Listing 1 and 2 show fragments of the original CrossRec implementation and the one based on proposed meta-model, respectively.

```
graph = new Graph();
String trainingDictFilename = "", trainingGraphFilename = ""
, filename = "";
allLibs = new HashSet<String>();
allTrainingLibs = new HashSet<String>();

for (Integer keyTraining : keyTrainingProjects) {
        trainingPro = trainingProjects.get(keyTraining);
        trainingFilename = trainingPro.replace("git://github.com/",
        "").replace("/", "__");
        trainingGraphFilename = Paths.get(this.srcDir,
         "graph_" + trainingFilename).toString();
        trainingDictFilename = Paths.get(this.srcDir,
         "dicth_" + trainingFilename)
        .toString();
        trainingLibs = reader.getLibraries(trainingDictFilename);
        allTrainingLibs.addAll(trainingLibs);
        trainingDictionary = reader.readDictionary
        (trainingDictFilename);
        trainingGraph = new Graph(trainingGraphFilename,
        trainingDictionary);
        if (graph == null) {
                graph = new Graph(trainingGraph);
        } else {
                graph.combine(trainingGraph, trainingDictionary);
                }
}
 }
```

Listing 1: Creation of the training graph in the original CrossRec code.

It is worth noting that once this type of restructuring is operated for different kinds of recommender systems, and thus the expressive power of the RS metamodel is assessed, a graphical environment will be developed. Thus, final users will be able to define RS models that eventually will be transformed to source code in a completely transparent manner. Users will interact with the development environment only at the abstraction level provided by the RS metamodel to compose reusable artifacts in a guided manner.

```
public GraphDataRepresentation createGraph (GitHub dataSource) {
   GraphDataRepresentation crossRecGraph = LowcodersFactory
   .eINSTANCE.createGraphDataRepresentation();
   List<Dependence> deps = new ArrayList<Dependence>();
   crossRecGraph.setDatasource(dataSource);
   for (GitHubProject proj : dataSource.getGitHubProjects()) {
      for (String s : proj.getDependecies()) {
            deps.add(createProjectDep(proj, s));
         }
   }
   for (Dependence dep : deps) {
         crossRecGraph.getEdgeType().add(dep);
         crossRecGraph.getNodeType().add(dep.getSource());
   }
return crossRecGraph;
}
```

Listing 2: Creation of the training graph in terms of RS metamodel elements.

According to the experience gained in the CROSSMINER project and to the experiments performed so far by using the proposed RS metamodel, we believe that the envisioned LCDP as shown in Section 3.4 can be used in practice to develop real recommender systems including modeling assistant tools. The contributions of the implemented tools we presented in Section 3.2 and Table 1 are twofold. First, they are proofs of concept for our approach, and second they serve as a base for further development. We marked the branches that we have addressed so far as bold edges in Fig. 1 to show the coverage of our current implementation. For future work, we plan to develop various sub-systems to realize the features depicted in Fig. 1 and put them together in a dedicated LCDP.

5 RELATED WORK

This section discusses existing approaches, aiming to realize a personalized recommender system. Tuning such a system requires a preliminary analysis of key functionalities and features.

Besides software development, recommender systems are spread in different domains. Cheng *et al.* [4] propose a recommender system to support an e-commerce business application. To foster its personalization and tuning, a system function module structure was proposed. This design phase includes the capture and analysis of users' behavior, commodities similarity investigation, and the customization of the recommendation algorithm. The approach embeds all the identified components in a well-defined web application in which the business logic realizes the proposed tuning.

Heterogeneity issues in customizable recommender systems have been analyzed by involving two different use cases [27]. The participants had to tune the system according to their preferences. In the first session, users configured a travel recommender by means of different facets of the trip, i.e., costs, food, and location. The second use case involved a personal exercise recommender system for the training activity. The results show that even homogeneous

groups of users select different system configurations. Thus, a tailored recommender system might consider the mental model of the target users, namely their preferences and custom algorithms.

Blended recommending [13] introduces a similar strategy embedded in a movie recommender. It implements several filtering techniques used in the domain, i.e., content-based filtering and collaborative filtering. Using the blended recommending strategy, users can specify a recommendation algorithm as well as refine its parameters in a hybrid filtering fashion. In this landscape, our work aims to cope with these challenges by promoting the adoption of a low-code platform. To our best knowledge, this is the first attempt to use such a technology in this domain.

6 CONCLUSIONS AND FUTURE WORK

We proposed a novel approach that leads up a new opportunity: allowing software developers to build personalized and well-defined recommender systems using an LCDP. The key points of this kind of platform are flexibility, easy development, and maintainability. The proposed idea is built on well-known concepts, including modeling, and domain specific languages. By exploiting the proposed feature model, we could represent not only the existing recommender systems but also new approaches belonging to this class of systems. The overall architecture will manage to face the main challenges arising from this first study, especially the integration of heterogeneous components and the user's experience.

To allow a concrete usage of this platform, a formal specification is needed to handle the complexity of the task, e.g., specifying constraints among the components and take care of the input and output interfaces. The performed domain analysis is reassuring, recommender systems can be developed in a disciplined manner as they consist of components implementing recurring functionalities (i.e., data preprocessing, capturing context, producing and presenting recommendations). The effort done so far leverages on such commonalities with goal of asking recommender systems developers to focus only on core functionalities, which cannot be abstracted and generalized, e.g., tailored similarity functions or specific encodings of metamodel entities. Thus, we expect hierarchies of DSLs each focusing on specific aspects or phase of the recommendation process.

ACKNOWLEDGEMENTS

The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223.

REFERENCES

[1] Edouard Batot and Houari Sahraoui. 2016. A generic framework for model-set selection for the unification of testing and learning MDE tasks. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16*. ACM Press, Saint-malo, France, 374–384. https://doi.org/10.1145/2976767.2976785

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (Sept. 2010), 615–636.

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. 2013. Recommender systems survey. *Knowledge-Based Systems* 46 (July 2013), 109–132.

[4] TianMin Cheng. 2019. Product Recommendation System Design. In *Proceedings of the 2019 2nd International Conference on Information Management and Management Sciences - IMMS 2019*. ACM Press, Chengdu, China, 71–74.

[5] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T. Nguyen. 2020. A Multinomial Naïve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories. In *Proceedings of the Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '20)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/3383219.3383227

[6] Guillaume Dupont, Concordia University, Sadaf Mustafiz, Concordia University, Ferhat Khendek, Concordia University, Maria Toeroe, and Ericsson Inc. 2018. Building Domain-Specific Modelling Environments with Papyrus: An Experience Report. (2018), 8.

[7] R. Holmes, R. J. Walker, and G. C. Murphy. 2005. Strathcona Example Recommendation Tool. (2005), 4.

[8] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal* 16, 3 (Nov. 2015), 261–273.

[9] K. Kim, D. Kim, T. F. Bissyandé, E. Choi, L. Li, J. Klein, and Y. L. Traon. 2018. FaCoY: a code-to-code search engine. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. ACM Press, Gothenburg, Sweden, 946–957.

[10] A. Korotaev and L. Lyadova. 2018. Method for the Development of Recommendation Systems, Customizable to Domains, with Deep GRU Network:. In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. SCITEPRESS - Science and Technology Publications, Seville, Spain, 231–236.

[11] LASER and LASER. 2015. *Software engineering: international summer schools, LASER 2013-2014, Elba, Italy: revised tutorial lectures*. Number 8987 in Lecture notes in computer science Programming and software engineering. Springer, [Cham] Heidelberg. OCLC: 944106474.

[12] E. Linstead, Sushil Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery* 18, 2 (April 2009), 300–336.

[13] Benedikt Loepp, Katja Herrmanny, and Jürgen Ziegler. 2015. Blended Recommending: Integrating Interactive Information Filtering and Algorithmic Recommender Techniques. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. ACM Press, Seoul, Republic of Korea, 975–984.

[14] Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. 2015. Example-driven meta-model development. *Software & Systems Modeling* 14, 4 (Oct. 2015), 1323–1347. https://doi.org/10.1007/s10270-013-0392-y

[15] Ángel Mora Segura and Juan de Lara. 2019. Extremo: An Eclipse plugin for modelling and meta-modelling assistance. *Science of Computer Programming* 180 (July 2019), 71–80. https://doi.org/10.1016/j.scico.2019.05.003

[16] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta. 2019. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 1050–1060.

[17] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, A. Pierantonio, and L. Iovino. 2019. Automated Classification of Metamodel Repositories: A Machine Learning Approach. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 272–282.

[18] P. T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio. 2018. CrossSim: Exploiting Mutual Relationships to Detect Similar OSS Projects. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 388–395.

[19] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta. 2020. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software* 161 (2020), 110460.

[20] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes. 2018. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *2018 IEEE Symposium on Security and Privacy (SP)*. 634–647.

[21] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. 2016. Prompter: Turning the IDE into a self-confident programming assistant. *Empirical Software Engineering* 21, 5 (Oct. 2016), 2190–2231.

[22] C. Richardson and J. R. Rymer. 2016. The forrester wave: Low-code development platforms, q2 2016. Technical report, Forrester Research.

[23] C. Richardson and J. R. Rymer. 2016. Vendor Landscape: The Fractured, Fertile Terrain Of Low-Code Application Platforms. (2016), 23.

[24] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann (Eds.). 2014. *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[25] Riccardo Rubei, Claudio Di Sipio, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow posts to support software developers. *Information and Software Technology* (2020), 106367. https://doi.org/10.1016/j.infsof.2020.106367

[26] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020 - to appear*.

[27] Jacob Solomon. 2016. Heterogeneity in Customization of Recommender Systems By Users with Homogenous Preferences. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 4166–4170.