2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) | 978-1-6654-2484-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/MODELS-C53483.2021.00090

DoMoBOT: An AI-Empowered Bot for Automated and Interactive Domain Modelling

Rijul Saini
Dept. of ECE
McGill University
Montréal, QC, Canada
rijul.saini@mail.mcgill.ca

Gunter Mussbacher
Dept. of ECE
McGill University
Montréal, QC, Canada
gunter.mussbacher@mcgill.ca

Jin L.C. Guo
School of Computer Science
McGill University
Montréal, QC, Canada
jguo@cs.mcgill.ca

Jörg Kienzle
School of Computer Science
McGill University
Montréal, QC, Canada
joerg.kienzle@mcgill.ca

***Abstract*—Domain modelling transforms informal requirements written in natural language in the form of problem descriptions into concise and analyzable domain models. As the manual construction of these domain models is often time-consuming, error-prone, and labor-intensive, several approaches already exist to automate domain modelling. However, the current approaches suffer from lower accuracy of extracted domain models and the lack of support for system-modeller interactions. To better assist modellers, we introduce DoMoBOT, a web-based Domain Modelling BOT. Our proposed bot combines artificial intelligence techniques such as natural language processing and machine learning to extract domain models with higher accuracy. More importantly, our bot incorporates a set of features to bring synergy between automated model extraction and bot-modeller interactions. During these interactions, the bot presents multiple possible solutions to a modeller for modelling scenarios present in a given problem description. The bot further enables modellers to switch to a particular solution and updates the other parts of the domain model proactively. In this tool demo paper, we demonstrate how the implementation and architecture of DoMoBOT support the paradigm of automated and interactive domain modelling for assisting modellers.**

***Index Terms*—Domain Models, Natural Language (NL), Machine Learning (ML), Bot, Model Extraction, Recommendation, Bot-Modeller Interactions, Traceablity Knowledge Graph**

I. Introduction

During requirement analysis or early stages of design, domain modelling transforms informal requirements (domain problem descriptions) expressed in natural language (NL) into concise and analyzable domain models. These models capture the key concepts of a domain in the form of classes, attributes, relationships, and cardinalities (class diagrams). Constructing these domain models manually is time-consuming, error-prone, and labor-intensive, particularly for large problem descriptions. Therefore, several approaches already exist to automate the construction of these domain models for the given problem descriptions. However, the extracted models from these approaches suffer from low accuracy. Furthermore, these approaches lack the support of system-modeller interactions.

Motivating Scenario: Domain modelling scenarios can be often modelled in multiple ways. For example, the scenario "Products have two categories - commodity and premium", can be modelled using the *Generalization* solution (the categories are modelled as sub-classes) or the *Enumeration* solution (the categories are modelled as enumeration items). The support for automated domain modelling is advantageous to quickly prototype a domain model for the given problem description. However, extracted domain models using current approaches represent only one solution for a given modelling scenario. This inhibits modellers to gain insights into other possible solutions which may further leave problem descriptions imprecise. For instance, the extracted model uses the *Enumeration* solution to model categories. The presentation of the other solution (*Generalization*) can indicate to a modeller that more information about the categories needs to be added in the problem description if categories are to be modelled only by the *Generalization* solution, i.e. when categories exhibit different characteristics (attributes or associations). Furthermore, manually updating the *Enumeration* solution to the *Generalization* in the extracted domain model defeats the whole purpose of automated domain modelling. Therefore, we argue that synergy is required between automated support and system-modeller interactions to better assist modellers.

In our previous work, we combine natural language processing (NLP) and machine learning (ML) techniques to extract domain models of high accuracy [1], [2], [3]. In addition, we propose a *Traceability Information Model* to trace domain modelling decisions taken by our extractor system [4]. In this tool demo paper, we present our web-based **do**main **mo**delling **bot**, **DoMoBOT**, with the main focus to demonstrate how the architecture and implementation of DoMoBOT facilitate bot-modeller interactions. The rest of the paper is organized as follows: Section II presents an overview of our proposed bot and its architecture. Sections III explains our tool's capabilities in the context of possible bot-modeller interactions. Section IV describes existing tools that are similar to ours and explains how our tool is different from them. Finally, Section V highlights ongoing and future work.

II. Overview of DoMoBOT

The architecture of our proposed bot is composed of various components, as shown in Figure 1. The bot has a Frontend that provides the *User Interface*. The Frontend is developed using the React JS and GOJS frameworks (see Figure 1). The Backend of our bot is developed using the Python programming language and composed of *Model Extraction*, *Recommendation*, and *Query-Answering Components* in addi-

978-1-6654-2484-4/21/$31.00 ©2021 IEEE
DOI 10.1109/MODELS-C53483.2021.00090

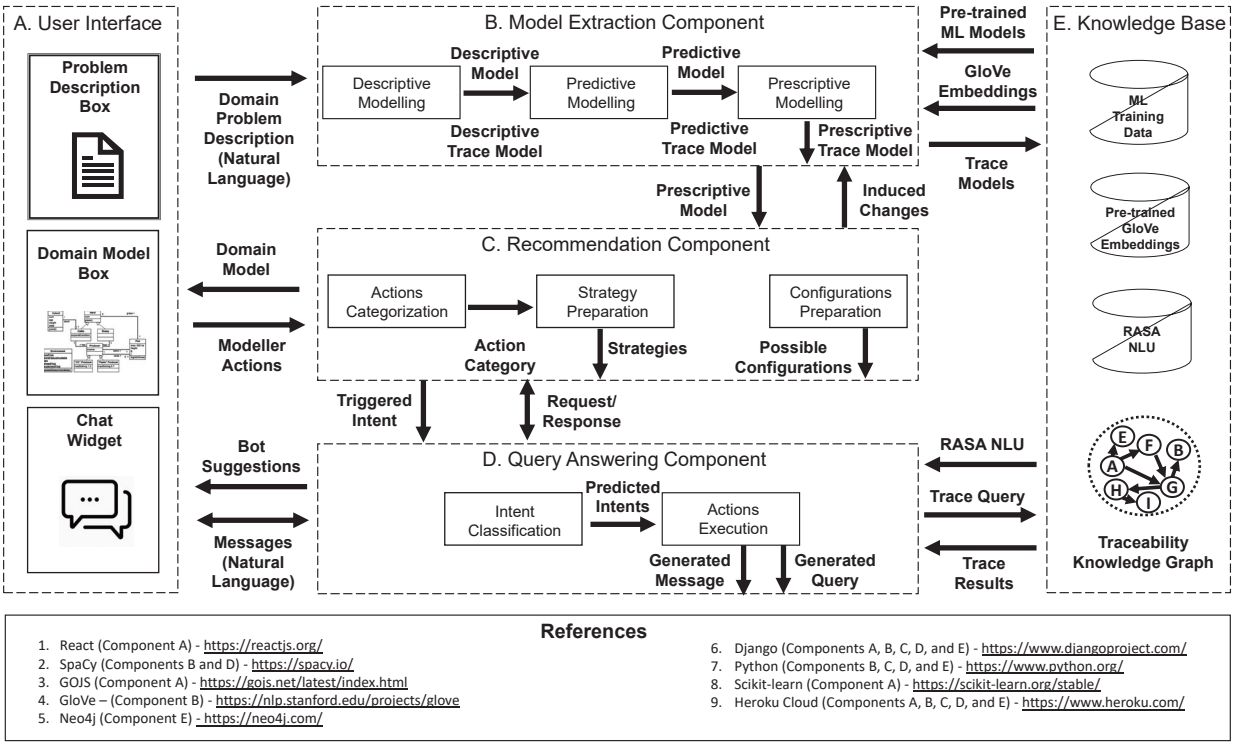


Fig. 1. Architecture of DoMoBOT.

tion to the *Knowledge Base*. The Django framework integrates our Frontend and Backend. We further plan to deploy our bot on the Heroku cloud as a web-based service. Excluding all external libraries, the bot is approx. $\approx$ 4500 lines of source code. In this section, we explain the architecture of our bot using a small problem description of an Employees Management System (PD_{EMS}): Employees are assigned to multiple projects. Each project has a project ID and title. Employees can be temporary employees or permanent employees.

A. User Interface

Our proposed bot provides an intuitive user interface to the modeller for performing various functions. First, a modeller can write a problem description in NL in the *Problem Description Box*. This problem description is then processed in the *Model Extraction Component*. Second, an extracted domain model is visualized in the *Domain Model Box*. The bot also allows modellers to update the extracted model in this box. All the changes are collected and processed in the *Recommendation Component*. Finally, bot-modeller communications take place inside the *Chat Widget*. These communications are further processed in the *Query Answering Component*.

B. Model Extraction Component

The bot extracts domain models from the given problem descriptions by combining NLP and ML techniques. Different models and their meta-data are generated across the various stages of the model extraction process. These models are then merged to generate a domain model and their meta-data is transformed into a *Traceability Knowledge Graph*.

Descriptive Modelling: We call this step *descriptive* because it abstracts a given *Problem Description* written in NL and captures only the relevant domain concepts in the form of classes, attributes, relationships, and cardinalities. In the *Descriptive Modelling* step, we use spaCy library (see Figure 1) to extract linguistic features, e.g., part-of-speech (POS) tags and lemmatized form of each word entity in the *Problem Description*. These linguistic features are then processed with our rule-based NLP to extract candidate domain concepts and their relationships [1]. In addition, for relationships and cardinalities, we compute semantic similarity scores between identified entities and a pre-defined list of sample word tokens for each relationship (e.g., association and attributes) and cardinality (e.g., zero-to-one and zero-to-many). For example, for the first sentence of PD_{EMS}, the "assigned" entity gets a maximum score in the *association* relationship list. The extracted concepts and their relationships are represented by the *Descriptive Model*, while their meta-data such as POS tags represent the *Descriptive Trace Model*.

Predictive Modelling: We call this step *predictive* as its objective is to improve the *Descriptive Model* based on the predictions of *Pre-trained ML Models* from the *Knowledge Base*. The information in the problem description is often not sufficient to model some necessary domain concepts. For example, the concepts "id" and "title" from the second sentence of PD_{EMS} are often modelled as attributes with types "Integer"

and "String", respectively. However, the information about their types is usually not present in a problem description. To address these scenarios, we use GloVe word embeddings [5] to capture the context information of participating concepts. For the classification of concepts, relationships, and cardinalities, we use the techniques proposed in our previous work [1], [2], [3]. The prediction results represent the *Predictive Model*, while their meta-data such as probability scores represent the *Predictive Trace Model*.

Prescriptive Modelling: We call this step *prescriptive* as it merges the *Descriptive* and *Predictive Models* and makes decisions for the domain model to be generated. In this step, we use our proposed algorithms [3] to resolve the conflicts while merging the *Descriptive* and *Predictive Models*. For example, for the third sentence of PD_{EMS}, the *Descriptive Model* generates *generalization* configuration with super class ("Employee") and sub-classes ("TemporaryEmployee" and "PermanentEmployee"). In contrast, the *Predictive Model* predicts *Enumeration* configuration for the same decision point. To resolve these conflicts, the proposed algorithms [3] consider the probability and semantic similarity scores of these results from the *Descriptive* and *Predictive Models*. In addition, we consider domain modelling policies which refer to general rules which we use while practicing domain modelling to resolve conflicts. For example, to resolve the current conflict (enumeration vs generalization), a policy is used which gives preference to the generalization configuration if the associated concepts ("TemporaryEmployee" and "PermanentEmployee") exhibit different characteristics, e.g., different *associations* or *attributes*. The selected configurations for all the decision points represent the *Prescriptive Model*. In contrast, the meta-data such as the source of selected configurations (*Descriptive* or *Predictive*) represent the *Prescriptive Trace Model*.

C. Recommendation Component

The *Prescriptive Model* is transformed into a suitable form in the *Recommendation Component* so that it can be visualized as a domain model. A modeller can update this extracted domain model where each modeller's change is processed in this component to generate proactive responses by the bot [6].

Configurations Preparation: During the model extraction process, decision points are identified for domain scenarios present in a given problem description. For each decision point, *Possible Configurations* are generated. Furthermore, while preparing these configurations, slices are created for each configuration. For example, the *enumeration* decision point is used to model the scenario which is present in the third sentence of PD_{EMS}. To prepare the corresponding configuration, four slices are created – (Employee, *attribute*, type), (type, *type*, EmployeeType), (EmployeeType, *enumeration_item*, TemporaryEmployee), and (EmployeeType, *enumeration_item*, PermanentEmployee).

Actions Categorization: A modeller can update the extracted domain model or problem description. Each action change made by a modeller is collected and categorized into discrete categories in the *Action Categorization* step. For example, a modeller's action to change the type of attribute "id" from "Integer" to "String" falls into the *Attributes - Update Type* category. Similarly, we create other categories for potential actions which could be performed by a modeller [6]. The output from this step is the *Action Category*.

Strategy Preparation: Based on each *Action Category*, one or more *Strategies* are identified to create the *Bot Suggestions* followed by the *Induced Changes* to update the *Prescriptive Model* and *Domain Model*. For example, a modeller's action of removing a class from the extracted model (*Class - Remove* action category), prepares a strategy for the bot response. In this strategy, the bot starts a search for alternative configurations for the associated decision point, which are closest to the new state of extracted model, i.e., after removing a class. If the bot discovers any alternative configurations then a new strategy is prepared for presenting the configuration to the modeller. We further explain bot-modeller interactions in Section III.

D. Query Answering Component

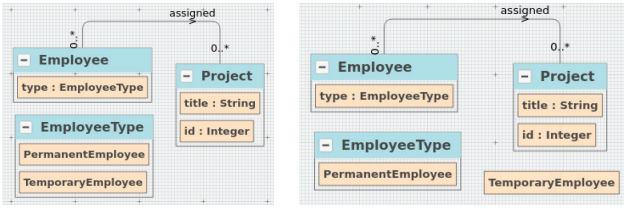
Bot-modeller communication is managed by the *Query Answering Component*. For example, a modeller can initiate communication with the bot if the modeller is interested to gain insights into the modelling decisions for extracted model elements. Similarly, a bot can initiate communication by presenting suggestions or taking confirmation from a modeller before updating the domain model with a new configuration.

Intent Classification: A modeller's message to the bot which is written in NL is first interpreted in the *Intent Classification* step. In this step, we use the *RASA NLU (Natural Language Unit)* pipeline which includes pre-trained models to classify the messages into one or multiple pre-defined intents. For example, an intent *Find Rationale* is predicted for the modeller's message "Why is the EmployeeType enumeration class created?". Similarly, some intents are forcefully triggered (*Triggered Intents*) by the bot, e.g., when presenting alternative configurations to a modeller (*Offer Suggestions* intent).

Actions Execution: For each *Predicted Intent*, the corresponding actions are executed, which further results in a *Generated Message* and/or *Generated Query*. A *Generated Message* is presented to the modeller in NL, while a *Generated Query* is transformed into a *Trace Query* to retrieve the trace results (meta-data) from the *Traceability Knowledge Graph*. For example, a trace query for an extracted model element returns the indices of sentences and words of the problem description which are responsible for modelling this element.

E. Knowledge Base

The *Knowledge Base* provides *Pre-trained ML Models* and *GloVe Embeddings* to the *Model Extraction Component* for extracting the domain model from a given problem description. In addition, the *Knowledge Base* provides *RASA NLU* to the *Query Answering Component* for interpreting the messages and predicting their associated intents. We manually prepare the training data for *Pre-trained ML Models* and for the classifiers used in the *RASA NLU* pipeline. Furthermore, the *Trace Models* constructed during the model extraction process are



(a) Initial State (b) New State

Fig. 2. Two States of the Extracted Domain Model.

transformed into the corresponding trace graphs. These graphs are then weaved together to create a *Traceability Knowledge Graph*. This graph conforms to the *Traceability Information Model* which maintains different artifacts generated across the model extraction process and trace relationships among these artifacts to enable traceability of modelling decisions [4].

III. Bot-Modeller Interactions

In this section, we demonstrate our bot's support for bot-modeller interactions. We use the running scenario of PD_{EMS} to describe some possible bot-modeller interactions.

Domain Model Generation: A modeller provides the problem description PD_{EMS} and the corresponding domain model is generated automatically as shown in Figure 2a. The extracted domain model is composed of "Employee", "Project", and "EmployeeType" classes. The "Employee" class has the "type" attribute (with type "EmployeeType"). The "EmployeeType" enumeration class has enumeration items – "TemporaryEmployee" and "PermanentEmployee". The "Project" class has "id" and "title" attributes with "Integer" and "String" types, respectively. Finally, there is an association relationship between Employee and Project classes with zero-to-many cardinality on both sides of the relationship.

Modeller Actions: After observing the extracted model, a modeller may want to switch to a different configuration for the decision points which exist in the domain model. For the running scenario, the modeller prefers the configuration with the *Player-Role* pattern over the *Enumeration* configuration to support modelling of features of temporary and permanent employees. Therefore, the modeller makes changes to the extracted domain model in multiple ways with the intention of bringing the state of the extracted domain model closer to the *Player-Role* configuration for the associated decision point. For example, a modeller brings the *TemporaryEmployee* enumeration item outside of the *EmployeeType* enumeration class, as shown in Figure 2b. This action leads to the change of the state of the extracted domain model, i.e., the "TemporaryEmployee" concept is no longer an item of a class.

Bot's Analysis: The bot analyzes the new state of the extracted domain model as shown in Figure 2b. During this analysis, the bot determines that a modeller has performed a *Concept - Ungroup* action, i.e., removing the *TemporaryEmployee* from the class (group). Next, the bot prepares a strategy to address the *Concept - Ungroup* action. In this strategy, the bot searches for equivalent configurations for the associated decision point where the *TemporaryEmployee* concept is not a part of the group, i.e., not an enumeration item. The bot further validates if these equivalent configurations are rejected by the modeller in the past. The bot also allows modellers to skip this validation by deselecting the flag "Consider Modeller Rejections" in the tool.

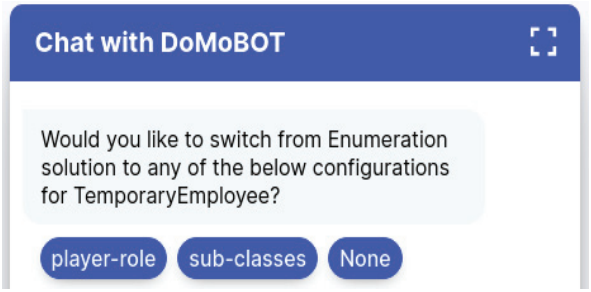


Fig. 3. Bot's Suggestions.

Bot's Recommendations: During analysis, if the bot discovers alternative configurations, then the bot prepares a new strategy to present these configurations to the modeller. For the example scenario, the bot finds two alternative configurations: *Player-Role* and *Sub-classes*. The bot then triggers *Offer Suggestions* intent to present alternative configurations as two options inside the chat widget, as shown in Figure 3.

Modeller Confirmation: A modeller can confirm "Yes" by selecting either the *Player-Role* option or the *Sub-classes* option, as shown in Figure 3. On the other side, a modeller can also decline the suggestions by selecting the *None* option. A modeller's "Yes" confirmation triggers the *Accept Suggestion* intent, while a modeller's "No" confirmation triggers the *Decline Suggestion* intent.

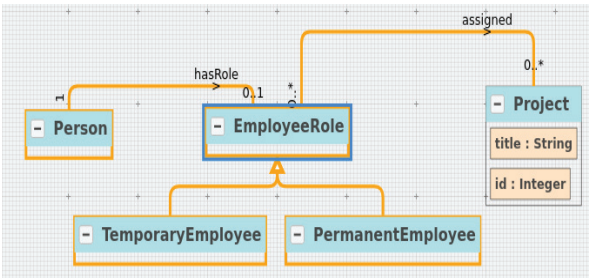


Fig. 4. Updated Domain Model.

Bot's Response: Based on the modeller's response to the bot's suggestions, different intents are triggered. These intents induce changes to the *Prescriptive Model* where the selection of configurations for the associated decision points changes. In addition, configurations for other decision points may be changed so that all the configurations are compatible together when presented to a modeller again in the form of an updated extracted domain model. A new version of the domain model (adapted) is presented to the modeller with the highlighted changes, as shown in Figure 4. The modeller can also undo or redo the changes by selecting the respective buttons.

IV. Related Work

In the literature, some recommendation systems already exist which aim to assist modellers by offering suggestions based on certain factors such as the similarity between the current content (models) and the existing models in a knowledge base [7], [8], [9], [10], [11], [12]. In addition, a research plan is proposed by Savary-Leblanc et al. [13] which aims to create modelling assistants using Artificial Intelligence (AI) techniques for providing suggestions. In contrast, our proposed bot aims to extract complete domain models with high accuracy from problem descriptions written in NL and to facilitate bot-modeller interactions. The existing recommendation systems in the literature aim to provide suggestions based on the syntactic or semantic match with the contextual knowledge or general knowledge of the domain. On the other side, the objective of our bot is to provide insights into the modelling decisions of extracted model elements based on the *Traceability Knowledge Graph*, which goes beyond providing suggestions. Also, Pérez-Soler et al. [14][15] develop a modelling bot using NLP techniques to interpret the users' inputs in their NL for building a metamodel incrementally. However, their bot neither generates a complete domain model for a given problem description nor provides insights into the modelling decisions taken by the extractor system. Furthermore, Mussbacher et al. present a conceptual framework to build intelligent modelling assistants more systematically [16]. In contrary, our work focuses on building a modelling assistant for one formalism – domain modelling, and provides a proof-of-concept in the form of a web-based modelling bot.

V. Discussion and Future Work

Synergy between automated model extraction and bot-modeller interactions is required to better assist modellers. Support for bot-modeller interactions enables modellers to quickly switch to their preferred configurations in the extracted domain model and to update the problem descriptions when required. In this paper, we present DoMoBOT, a web-based modelling bot to automate model extraction and facilitate bot-modeller interactions. In our previous work, we validate the accuracy of extracted domain models [1], [2], [3], [4] for an unseen set of problem descriptions. The average precision and recall scores of extracted domain models using our proposed approach are 90% and 77%, respectively. During bot-modeller interactions, our bot discovers alternative configurations, presents these configurations to a modeller in the form of suggestions, and updates the domain models. Therefore, we also perform the evaluation of our bot for these categories where we compare the bot responses with the ground truth domain models (manually constructed domain models) for the test problem descriptions. During this comparison, we obtain the total relevant concepts, total retrieved concepts, and the total retrieved relevant concepts for each category to calculate F1 scores. Our bot achieves the median F1 scores of 81%, 90%, and 88% in the Found Configurations, Offered Suggestions, and Updated Domain Models categories, respectively [6]. Finally, we conduct an exploratory user study with 11 modellers [6]. The preliminary results are promising.

In the future, first, we plan to augment the training data based on bot-modeller interactions. This augmented data can improve the accuracy of predicted intents while interpreting the messages or actions in the future. Second, we plan to improve the model extraction process to increase the accuracy of extracted domain models. Third, we plan to cover a wider range of domain modelling scenarios as well as potential solutions for these scenarios. Fourth, we aim to perform a more rigorous evaluation of our bot. Finally, we plan to conduct a large-scale user study to assess the benefits and limitations of our bot in facilitating bot-modeller interactions.

References

[1] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Towards queryable and traceable domain models," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020, pp. 334–339.

[2] ——, "A neural network based approach to domain modelling relationships and patterns recognition," in *2020 IEEE Tenth International Model-Driven Requirements Engineering (MoDRE)*, 2020, pp. 78–82.

[3] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "DoMoBOT: A bot for automated and interactive domain modelling," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3417990.3421385

[4] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Automated traceability for domain modelling decisions empowered by artificial intelligence," in *2021 IEEE 29th International Requirements Engineering Conference (RE) (to be published)*.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[6] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Automated, interactive, and traceable domain modelling," *Software & Systems Modeling (to be published)*, 2021.

[7] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An NLP-based architecture for the autocompletion of partial domain models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 91–106.

[8] H. Agt-Rickauer, R.-D. Kutsche, and H. Sack, "Domore-a recommender system for domain modeling." in *MODELSWARD*, 2018, pp. 71–82.

[9] A. Elkamel, M. Gzara, and H. Ben-Abdallah, "An UML class recommender system for software design," in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. IEEE, 2016, pp. 1–8.

[10] T. G. Cerqueira, F. Ramalho, and L. B. Marinho, "A content-based approach for recommending uml sequence diagrams." in *SEKE*, 2016, pp. 644–649.

[11] D. Lucrédio, R. P. d. M. Fortes, and J. Whittle, "Moogle: a metamodel-based model search engine," *Software & Systems Modeling*, vol. 11, no. 2, pp. 183–208, 2012.

[12] Á. M. Segura, A. Pescador, J. de Lara, and M. Wimmer, "An extensible meta-modelling assistant," in *EDOC 2016*. IEEE, 2016, pp. 1–10.

[13] M. Savary-Leblanc, "Improving MBSE tools UX with AI-empowered software assistants," in *MODELS 2019 Companion*. IEEE, 2019, pp. 648–652.

[14] S. Pérez-Soler, E. Guerra, J. de Lara, and F. Jurado, "The rise of the (modelling) bots: Towards assisted modelling via social networks," in *ASE 2017*. IEEE Press, 2017, pp. 723–728.

[15] S. Pérez-Soler, E. Guerra, and J. de Lara, "Collaborative modeling and group decision making using chatbots in social networks," *IEEE Software*, vol. 35, no. 6, pp. 48–54, November 2018.

[16] G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean *et al.*, "Opportunities in intelligent modeling assistance," *Software and Systems Modeling*, vol. 19, no. 5, pp. 1045–1053, 2020.