2021 IEEE 29th International Requirements Engineering Conference (RE) | 978-1-6654-2856-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/RE51729.2021.00023

Automated Traceability for Domain Modelling Decisions Empowered by Artificial Intelligence

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 abstracts real-world entities and their relationships in the form of class diagrams for a given domain problem space. Modellers often perform domain modelling to reduce the gap between understanding the problem description which expresses requirements in natural language and the concise interpretation of these requirements. However, the manual practice of domain modelling is both time-consuming and error-prone. These issues are further aggravated when problem descriptions are long, which makes it hard to trace modelling decisions from domain models to problem descriptions or vice-versa leading to completeness and conciseness issues. Automated support for tracing domain modelling decisions in both directions is thus advantageous. In this paper, we propose an automated approach that uses artificial intelligence techniques to extract domain models along with their trace links. We present a traceability information model to enable traceability of modelling decisions in both directions and provide its proof-of-concept in the form of a tool. The evaluation on a set of unseen problem descriptions shows that our approach is promising with an overall median F2 score of 82.04%. We conduct an exploratory user study to assess the benefits and limitations of our approach and present the lessons learned from this study.**

***Index Terms*—Domain Models, Traceability, Natural Language (NL), Machine Learning (ML), Traceability Knowledge Graph (TKG), Traceability Information Model (TIM)**

I. Introduction

Model-Driven Software Engineering prescribes the use of models and their transformations across different stages of software engineering to better understand and build software systems. These models can be defined by various modelling notations, e.g., use case models, state machines, and goal models to capture structural, behavioural, and intentional system aspects. One of the most widely used models is the domain model (class diagrams) [1] which captures the structural aspects of a system as a part of the domain modelling activity. Modellers practice domain modelling often during the requirement analysis or the early stages of design to transform a problem description which expresses requirements in natural language (NL) into domain concepts in the form of classes, attributes, relationships, and association cardinalities. The manual practice of domain modelling is both time-consuming and error-prone. These issues are further aggravated when problem descriptions are long which makes it hard to examine requirements manually [2] and to trace modelling decisions taken by modellers from domain models to problem descriptions or vice-versa. Furthermore, the inherently imprecise and ambiguous nature of NL leads to multiple interpretations of the underlying requirements in a problem description [3]. The use of models has been proposed among many other techniques to overcome the weaknesses of NL. However, these models can be incomplete and ambiguous too while abstracting the requirements. This inhibits the modellers to confirm confidently and promptly if the constructed domain models are complete and concise. Though several approaches exist to assist modellers by extracting domain models from problem descriptions in an automatic [4], [5], [6], [7], [8], [9] or semi-automatic manner [10], [11], several challenges exist due to an evident lack of support to enable traceability of domain modelling decisions taken by an extractor system.

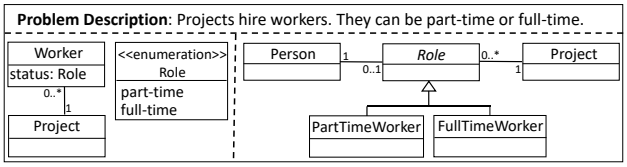

Fig. 1. Domain Modelling Example.

To illustrate these challenges, we use a domain modelling example where we show two possible domain model solutions (Figure 1) for a small problem description stated at the top. First, the word “They” can refer to either ”projects” or ”workers”, leading to distinct modelling decisions. Therefore, a traceability link is required between “They” in the problem description and “Worker” class in the domain model to highlight the current modelling decisions so that the problem description can be updated if not modelled correctly. Second, word entities for some concepts, e.g., “Person” and “Role” classes are not mentioned explicitly in the problem description. We call these concepts implicit concepts because they are required to be modelled in a domain model but are usually not present in a problem description. Traceability links are required for these concepts too to enable modellers to gain insights into the modelling decisions which abstract these implicit concepts. Third, both enumeration solution (left of Figure 1) and Player-Role pattern solution (right of Figure 1)

978-1-6654-2856-9/21/$31.00 ©2021 IEEE
DOI 10.1109/RE51729.2021.00023

correctly model the multiple roles a worker can play. However, it can be highlighted with traceability that no differentiating characteristics exist for these roles in the current problem description, and hence the Player-Role pattern does not need to be applied. Finally, modelling decisions traceability for the cardinalities is also required to indicate the assumption that the model extractor system made while modelling the cardinalities of a relationship, e.g., an employee can also be associated with multiple projects (zero-to-many). Therefore, an automated support for tracing domain modelling decisions in both directions is advantageous.

In this paper, we propose an automated approach to facilitate modelling decisions traceability using a Traceability Information Model (TIM) for the extracted domain model elements and their meta-data. The model extraction techniques in our proposed approach are built upon our previous work where we combine natural language processing (NLP) and Machine Learning (ML) techniques to extract domain models from problem descriptions in NL [5], [12], [13]. Since the TIM directly weaves the various artifacts generated across the model extraction process (model elements and their meta-data), we evaluate the model extraction results as well as the traceability results. In this paper, the main contributions are:

1) We propose an automated approach to establish the traceability relations among the artifacts which are generated during the model extraction process.
2) We present a *TIM* to facilitate the traceability of modelling decisions in forward and backward directions.
3) We provide a proof-of-concept of our approach in the form of a prototype tool that expresses the trace links and provides insights into the modelling decisions.
4) We evaluate and compare the model extraction results of our approach and that of our previous work [13]. Our proposed approach improves the precision and recall scores of our previous work by 4% and 9%, respectively.
5) Our traceability approach achieves an overall median F2 score of 82.04%. Also, we conduct an exploratory user study and present the lessons learned from this study.

The rest of the paper is organized as follows: Section II describes our proposed approach. Sections III presents our evaluation setup and results. Section IV provides a detailed discussion and presents the lessons learned. Section V describes threats to validity. Section VI surveys related work. Finally, Section VII concludes and highlights future work.

II. Proposed Approach

In this section, we present our proposed approach to address the challenges presented in Section I. Figure 2 shows an overview of our approach, which represents various layers to extract a *Domain Model* and the corresponding *Trace Graphs* from a given problem description. Though our approach builds upon our previous work for model extraction [5], [12], [13], we still explain the extraction process to better explain our approach for weaving the extraction results using the proposed TIM. First, we extract *Descriptive Model (DM)* and *Trace Graph* in the *Descriptive Layer* based on *Rule-based NLP* and *Descriptive Qualification* (see Section II-A). Second, we generate *Predictive Model (PM)* and *Trace Graph* in the *Predictive Layer* based on ML models and *Predictive Qualification* (see Section II-B). Third, we merge and evaluate *DMs* and *PMs* in the *Prescriptive Layer*. The results are then processed with *Prescriptive Qualification* to generate *Domain Model* and *Prescriptive Trace Graph* (see Section II-C). Finally, we weave *Descriptive*, *Predictive*, and *Prescriptive Trace Graphs* in the *Trace Layer* to obtain the *Traceability Knowledge Graph (TKG)* (see Section II-D).

A. Descriptive Layer

We call this layer descriptive as it abstracts a given *Problem Description* and captures only the relevant domain concepts in the form of classes, attributes, relationships, and cardinalities. These domain concepts are derived from the information that is already available in the form of textual problem descriptions. To explain our approach, we use a small *Problem Description* which represents *University Enrolment System* (PD_{UES}): *Departments manage enrolment of students in a university. They can be part-time students or full-time students. Each student is identified by a student ID and an email.*

Pre-processing: In this step, we use CoReference Resolution technique to identify all noun phrases that refer to the same entity or concept in a *Problem Description*. For example, the "They" in the second sentence of PD_{UES} refers to the entity "Students". Also, we split the *Problem Description* into individual sentences. The outputs from this step are *References* (a map of word entities and their corresponding noun phrases) and *Sentences* for a given *Problem Description*. Collecting these *References* separately distinguishes our model extraction process from our previous work [5], [12], [13].

Linguistic Features Assignment: The *References* and the individual *Sentences* from the previous layer are then processed in *Linguistic Features Assignment*. Before assigning linguistic features, we use *References* to replace the expressions by suitable concept entities. Next, we use spaCy [14] pre-trained models to assign linguistic features such as part-of-speech (POS) tags, dependency labels to the word tokens, and lemmas (lemmatized form of word entity). The output from this step is a *Document Object* which holds the individual word tokens and their corresponding meta-data such as linguistic features and *References*. At this stage, we do not pre-process the sentences based on features such as stop word. However, these features are considered in the next step with rule-based NLP while extracting domain concepts.

Descriptive Qualification: The *Document Object* is then processed in the *Descriptive Qualification* step which represents a set of *Qualifiers* for the *DM* (Q_{dm}):

$$\bigcup_{i=1}^{n} Q_{dm_i}(\bigwedge_{a=1}^{k} C_{dm_a}, \bigwedge_{b=1}^{l} R_{dm_b}, \bigwedge_{c=1}^{m} A_{dm_c})$$

Each *Qualifier* is further composed of sets of *Conditions* (C_a), *Rationale* (R_b), and *Actions* (A_c). A set of conditions represents heuristics based on rule-based NLP to extract noun

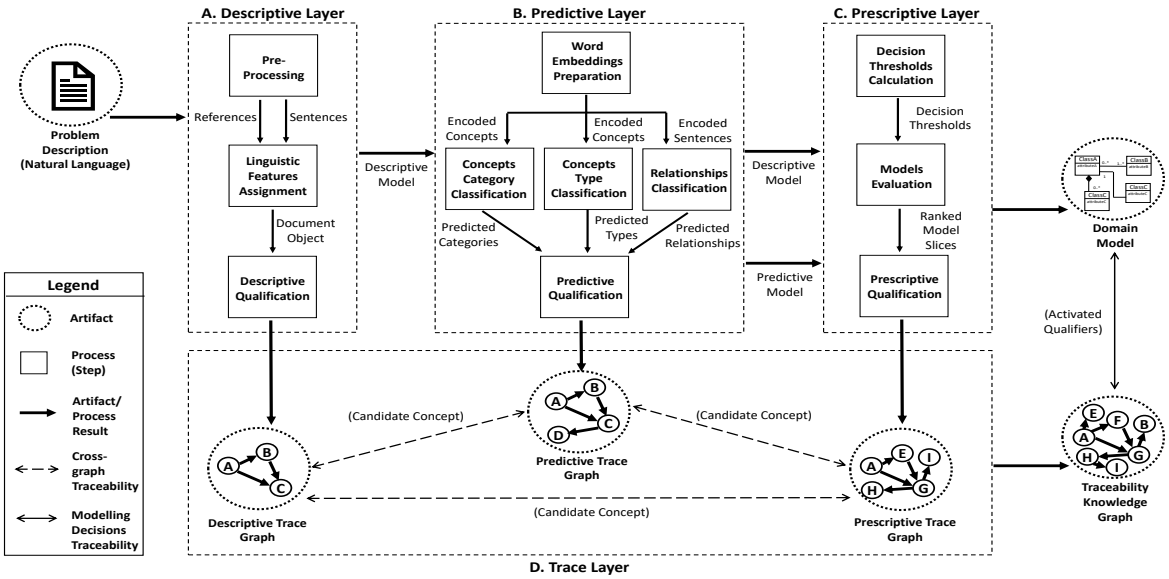

Fig. 2. Our Proposed Approach for Automated Traceability of Domain Modelling Decisions.

chunks and to determine if single or conjugated entities in the problem description can be qualified for a candidate domain concept. For example, conjugated entities "part-time students" in the second sentence of UES problem description form candidate concept "PartTimeStudent". Furthermore, these heuristics perform navigation over the *document object* to identify entities with verb-based dependencies. Next, we use spaCy pre-trained models to calculate semantic similarity scores between identified entities and a pre-defined list of sample word tokens for each relationship. The relationship's list with the maximum score determines the relationship of the entity. For example, "identified" entity in the third sentence of PD_{UES} obtains the maximum score for *attributes* relationship. A similar recipe is also used for cardinalities. A rationale indicates a logical basis or a reason in natural language which can be presented to a modeller during traceability of modelling decisions. Finally, an action represents a set of operations required under this *Qualifier* such as creating a class or an attribute (*DM*) and extracting the meta-data of these concepts such as sentence and word indices (*Descriptive Trace Graph*).

Each set of *Condition*, *Rationale*, and *Action* exists in conjunction, i.e., all its conjuncts should be true for an activated qualifier. A *Qualifier* is activated when all the *conditions* in the set are true. This will further perform all the operations in the *Actions* set and associate them with the set of *Rationale*. The *Activated Qualifiers* results in two artifacts – *DM* and *Descriptive Trace Graph*. The *DM* represents the extracted concepts, relationships and their cardinalities, based on the actions performed under the *Activated Qualifiers*. The *Descriptive Trace Graph* includes the meta-data associated with the extracted model elements and the problem description in addition to the similarity scores. We further explain the *Descriptive Trace Graph* in Section II-D.

B. Predictive Layer

We call this layer predictive because it aims to improve the *DM* based on the predictions of pre-trained ML models. Information in the problem description is often not sufficient to model the necessary domain concepts. For example, the concepts "student ID" and "email" from the third sentence of PD_{UES} 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 cases, it is imperative to capture the context information of participating concepts. However, it is either difficult or not possible to capture this context information using rule-based NLP alone. In this layer, we capture the context information of participating concepts using GloVe word embeddings [15]. For the classification of concept category, concepts type, and relationships, we use the technique proposed in our previous work [5].

Word Embeddings Preparation: The *DM* provides candidate domain concepts to the *Predictive Layer* which are first processed in the *Word Embeddings Preparation* step. We use GloVe word embeddings to obtain real-valued vector representations of candidate domain concepts which we call *Encoded Concepts*. The encoding of these vectors is done in accordance with their positions in the word embeddings space. For example, encoding of vectors for words "hospital" and "doctor" is similar as their vectors lie closer to each other in the semantic representation. Furthermore, we consider each concept as a primary concept one by one and pair it with other concepts (secondary concepts) at the sentence level to form *Encoded Sentences*. We place special tags to indicate the positions of primary and secondary concepts to an ML Model, e.g., "$< e1 >$ Departments $< /e1 >$ manage enrolment of $< e2 >$ students $< /e2 >$ in a university".

Concepts Category Classification: In this step, we use a binary classifier which predicts whether a given *Encoded Concept* should be modelled as a class or as an attribute. Some concepts such as "phone number" and "date of birth" are often modelled as attributes. Likewise, some concepts such as "Flight" and "Pilot" possess properties such as name, are often modelled as classes. We collect a corpus of commonly used noun words from WordNet [16] (approx. 5000 words) and manually label their category as either class or attribute. This labeled data is then used to train the classifier. The outputs from this step are the *Predicted Categories* and their corresponding probability scores for each *Encoded Concept*.

Concepts Type Classification: We use a multi-class classifier which predicts the type of a given *Encoded Concept* in the *Concepts Type Classification* step. The types could be *integer*, *string*, *float*, *date*, *time*, *enumeration* for an attribute concept and *class* for a class concept. For example, attribute concept "weight" is often modelled with type *float*. We use the same set of noun words as used in the *Concepts Category Classification* step and manually label their types. Though the type of class concept is not used in a domain model, we keep this category to reduce the number of none type instances, i.e., when a concept is not an attribute.

Relationships Classification: We use neural networks in this step to predict the relationship between the participating concepts (primary and secondary) and to predict their cardinalities for a given *Encoded Sentence*. The relationships – *association*, *generalization*, *composition*, *enumeration*, *attributes*, and the *Player-Role* pattern, are organized into different categories. Also, the possible directions of every relationship while reading a sentence is also considered while forming these categories. For example, the possible *Encoded Sentences* "Each $< e1 >$ student $< /e1 >$ has a unique student ID and an $< e2 >$ email $< /e2 >$" and "Each $< e2 >$ student $< /e2 >$ has a unique student ID and an $< e1 >$ email $< /e1 >$" fall into *has_attributes* and *attributes_of* categories, respectively. The cardinalities are also organized into discrete categories such as *zero-to-many* and *zero-to-one*.

Predictive Qualification: The *Prediction Results* are then processed in the *Predictive Qualification* step which represents a set of *Qualifiers* for the *PM* (Q_{pm}). Similar to the *Descriptive Qualifier*, each *Predictive Qualifier* is composed of a set of *Conditions* (C_a), *Rationale* (R_b), and *Actions* (A_c). A condition for a *PM* represents heuristics which consider all the prediction results and their probability scores to determine if a concept in the set of candidate domain concepts (which is provided by *DM*) is qualified to be modelled as a class or an attribute concept. Furthermore, a condition also determines if a predicted relationship and its predicted cardinalities are qualified to be included in the *PM*. For example, for the predicted relationship "composition", it is necessary that the predicted category for both primary and secondary concepts is class. Furthermore, the cardinality on the composite (whole) side of this relationship should be either *zero-to-one* or *one*. The definitions for the *Rationale*, *Action*, and *Activated Qualifier* remain the same as mentioned in Section II-A.

The *Activated Qualifiers* result in two artifacts – *PM* and *Predictive Trace Graph*. The *PM* represents the predictions for concepts category, relationships, and cardinalities on the source and target sides of a relationship, based on the actions performed under the *Activated Qualifiers*. The *Predictive Trace Graph* includes the meta-data associated with the prediction results such as probability scores and encoded sentences. We further explain the *Predictive Trace Graph* in Section II-D.

C. Prescriptive Layer

We call this layer prescriptive as it merges the *DM* and *PM* and makes decisions for the domain model to be generated. It is often possible that there are some conflicts between *DM* and *PM*. For example, for the sentence "A customer has multiple addresses", a *DM* generates an *attributes* relationship between "Customer" and "Address" concepts. In contrast, a *PM* predicts an *association* relationship by considering the context information of "Address" concept which indicates that it is possible for a customer to have more than one address such as temporary address and permanent address. To resolve this conflict, it is important to consider all the results together along with their probability and semantic similarity scores.

Decision Thresholds Calculation: In this step, we compute decision threshold values for different groups such as relationships and cardinalities. We use Youden index to calculate the decision thresholds for both *DM* and *PM*. Youden index represents the vertical distance between the diagonal line and the point on the receiver operating characteristic (ROC) curve. A point that maximizes this vertical distance indicates the decision threshold for that group. We plot ROC curves for *DMs* and *PMs* in each group for the validation data. We prepare this validation data by labeling the correct category in each group. The outputs of this step are *Decision Thresholds*.

Models Evaluation: Both *DM* and *PM* are now evaluated using the *Decision Thresholds* obtained from the previous step. We split the *DM* and *PM* into multiple model slices based on the source and target concepts of a relationship. A model slice holds the domain concepts and their relationships in addition to the corresponding probability scores. The equivalent model slices are then evaluated against each other. For example, a model slice *Customer-attributes-Address* in the *DM* and a model slice *Customer-association-Address* in the *PM* are considered equivalent model slices based on the same *source-target* concepts pair. Both of these slices are then evaluated by computing the difference between the probability score of the present relationship and the decision threshold value of this relationship. Furthermore, the remaining model slices which have different *source-target* pairs are also evaluated individually and we discard all those slices where the difference between the probability score of the present relationship and the decision threshold is negative. Finally, we create *Ranked Model Slices* based on their difference scores. The generation of *Ranked Model Slices* further distinguishes our approach proposed in this paper from our previous work [5], [12], [13].

Prescriptive Qualification: The *Ranked Model Slices* are processed in the *Prescriptive Qualification* step which

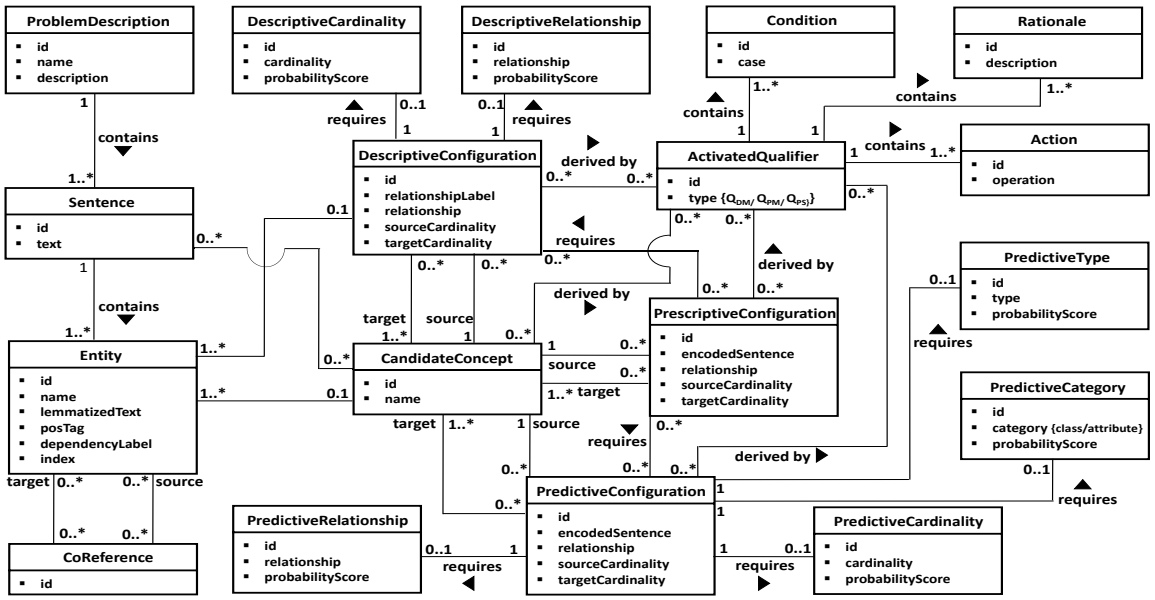

Fig. 3. Traceability Information Model.

represents a set of *Qualifiers* for the *Prescriptive Model (Q_{psm})*. Similar to the *Descriptive Qualifier* and the *Predictive Qualifier*, each *Prescriptive Qualifier* is composed of a set of *Conditions (C_a)*, *Rationale (R_b)*, and *Actions (A_c)*. A condition represents a domain modelling policy that transforms the *Ranked Model Slices* into a *Prescriptive Model (PSM)*. Domain modelling policies refer to general rules which a modeller follows while practicing domain modelling. For example, the policy "an object is composed in at most one other object" ensures that the model slices *University-composition-Department*, *Department-composition-Council*, and *University-composition-Council* lead to a configuration where "Council" is composed only once in "University". The definitions for the *Rationale*, *Action*, and *Activated Qualifier* remain the same as mentioned in Section II-A.

The *Activated Qualifiers* result in two artifacts – a *PSM* which describes the domain model and a *Prescriptive Trace Graph* which represents the meta-data of a *PSM* such as the source of each configuration (*DM* or *PM*) and their corresponding *Decision Thresholds*. We further explain the generation of *Prescriptive Trace Graph* elements in Section II-D.

D. Trace Layer

The knowledge about the domain model (to be built) evolves over multiple layers (from Descriptive to Prescriptive layers). Therefore, traceability relations are required to map the knowledge state at one layer to the knowledge state at another layer. In our proposed approach, the knowledge states are represented by the three trace graphs – *Descriptive*, *Predictive*, and *Prescriptive Trace Graphs* in the *Trace Layer*. In addition, the *Trace Layer* weaves these graphs and generates the *TKG* which enables modellers to trace domain modelling decisions from domain model elements to the problem description (backward traceability) and from problem description to model elements (forward traceability). The *TKG* conforms to the *Traceability Information Model (TIM)* as shown in Figure 3. The *TIM* is comprised of two types of elements – traceable artifacts and traceability relations between these artifacts. Each artifact is represented by a UML class and a valid trace between two artifacts is represented by a relation. During the instantiation of domain model extraction process for a problem description, we map automatically each class in the TIM to a node, the attributes of a TIM class to the properties of a node, and the relations between two classes to the node relationships in a trace graph. Each node in a graph has an *id* property which takes a unique integer. Though navigation is possible from either sides of a relationship in this graph due to its bidirectional nature, we show small arrows in the *TIM* to indicate the direction when interpreting the annotated relations.

Descriptive Trace Graph: For PD_{UES}, first, we create the *ProblemDescription* node with properties – *name* (University Enrolment System) and *description* (the text of PD_{UES}). Second, for each sentence PD_{UES}, we create a node *Sentence* with different properties (*id*, *text*). Also, we create a relationship between the *ProblemDescription* node and each *Sentence* node which represents a containment relationship (a problem description is composed of sentences). Third, we represent each word token in a sentence by individual *Entity* nodes. For example, the word "identified" in the third sentence of PD_{UES} is represented by the *Entity* node with properties – *name* (identified), *lemmatizedText* (identify), *posTag* ("VNB" which represents a verb), *dependencyLabel* ("ROOT" which represents a grammatical relation and points to the root of the sentence), and *index* (3). Fourth, we create nodes for references between source and target entities. For example, we create a

CoReference node that has relationships with the source *Entity* node ("students" from the first sentence) and the target *Entity* node ("They" from the second sentence).

Finally, we create a node for each *ActivatedQualifier* with properties, e.g., *type* ("Q_{dm}" which represents *Activated Descriptive Qualifier*). These qualifiers drive the construction of concepts (*CandidateConcept*) and their corresponding relationships (*DescriptiveConfiguration*). For example, to extract domain concepts (Student, id, and email) from the third sentence of PD_{UES}, we create an *ActivatedQualifier* node which has containment relationships with *Condition*, *Rationale*, and *Action* nodes. In this case, one possible *Condition* node has a property *case* (checks if entity exists in a noun chunk and if its *posTag* is noun-based such as 'NN' that represents noun, singular or mass). The corresponding *Rationale* node has a property *description* (entities in a noun chunk may represent a domain concept – class or attribute). In addition, the *Action* node has a property *operation* (create *CandidateConcept* nodes for "Student", "id", and "email" and create their relationships with corresponding entities). A similar process is followed for the *DescriptiveConfiguration* node with properties – *relationshipLabel* (identify), *relationship* ("attributes" between source [Student] and target [email, id]). Both *sourceCardinality* and *targetCardinality* properties are empty (" ") in this case as the relationship is of type "attributes". The *DescriptiveConfiguration* node requires the relationships and cardinalities from the *DescriptiveRelationship* and the *DescriptiveCardinality* nodes, respectively. The latter nodes also have a *probabilityScore* property which represents a real value between 0 and 1.

Predictive Trace Graph: This graph is comprised of nodes – *ActivatedQualifier* with a property *type* ('Q_{pm}" which represents *Activated Predictive Qualifier*), *Condition*, *Rationale*, and *Action* nodes. In addition, the *Predictive Trace Graph* includes *PredictiveConfiguration* nodes which represent predicted relationships. A *PredictiveConfiguration* node requires *PredictiveCategory*, *PredictiveType*, *PredictiveCardinality*, and *PredictiveRelationship* nodes for concepts category (class or attribute), concepts type (e.g., integer and float for attributes), cardinalities (e.g., zero-or-one), and relationships (e.g., generalization and composition), respectively. For example, the second sentence in PD_{UES} becomes "students can be part-time students or full-time students" after using the CoReference Resolution technique. One possible *encodedSentence* for this sentence is "<e1> students < /e1> can be part-time <e2> students < /e2> or full-time students". While forming encoded sentences, we only place tags around the base word (students) and not the whole noun chunk (part-time students) because neighbouring words in a noun chunk are often there just to describe the noun word (concept entity).

In this case, one of the *PredictiveRelationship* nodes has properties, e.g., *relationship* (Player-Role) and probabilityScore (a real-value between 0 and 1). One possible *Condition* node has a property *case* (if the predicted relationship is Player-Role then check if the category for all the participating concepts (source and target) is class). The *description* properties of the two corresponding *Rationale* nodes are "The source concept [Student] can play multiple roles during its lifetime. The use of the Player-Role pattern restricts the creation of different objects each time the source concept changes its role among target concepts [PartTimeStudent, FullTimeStudent]." and "We often model all the participating concepts in this relationship as classes.", respectively. These *Rationale* nodes also play an important role in providing insights into the modelling decisions of the implicit concepts, e.g., "Person" and "Role", to a modeller in NL.

In addition, the example requires multiple *Action* nodes with different properties. Some possible operations include the creation of a new *CandidateConcept* (Role) with no relationship with any *Entity* node and the creation of a *PredictiveConfiguration* node with properties – *relationship* ("generalization" between source [Role] and target ["PartTimeStudent"]). All nodes (artifacts) in the *Predictive Trace Graph* are traceable to the nodes (artifacts) of the *Descriptive Trace Graph* through *CandidateConcept* nodes. However, some concepts which are created in the *Predictive Layer* or *Prescriptive Layer*, e.g., "Role" concept (implicit concept), do not have any relationship with an *Entity* node. Therefore, we include an operation in the *Action* node for these newly created concepts to create a relationship between these concepts and the *Sentence* node. For the "Role" concept, a relationship is created with the second sentence of PD_{UES} (based on the *encodedSentence*) which is responsible for modelling this "Role" class.

Prescriptive Trace Graph: The *ActivatedQualifier* nodes with properties *id* and *type* ('Q_{psm}" which represents *Activated Prescriptive Qualifier*) derive the *PrescriptiveConfiguration* nodes of this graph. The other nodes in this graph are *Condition*, *Rational*, and *Action*. We do not include *Decision Thresholds* and *Ranked Model Slices* in the *TIM* as they represent the intermediate results while creating prescriptive configurations. The *PrescriptiveConfiguration* node requires *DescriptiveConfiguration* and *PredictiveConfiguration* nodes for models evaluation. For example, for a problem description "Sessions are of two types – online and classroom. The classroom sessions have a room number.", the *PrescriptiveConfiguration* gets three ranked model slices. First, enumeration relationship from the *DescriptiveConfiguration – Type[enumeration class]-enumeration-(Classroom,Online)[enumeration items]*. Second, generalization relationship from the *PredictiveConfiguration – Type[superclass]-generalization-(Classroom, Online)[subclasses]*. Third, attributes relationship from *PredictiveConfiguration – Classroom[class]-attributes-roomNumber[attribute]*. Though the rank of *DescriptiveConfiguration* relationship is higher than the other relationships, the above conflict (enumeration vs generalization) should be resolved with the correct relationship for the *PredictiveConfiguration* (generalization in this case). Therefore, a *Condition* with *case* (in case of enumeration vs generalization, check if concepts [subclasses or enumeration items] have different attributes or associations) is required. The corresponding *Rational* node has a property *description*

(Concepts exhibiting different characteristics or behaviour should be modelled with generalization solution). This further requires an *Action* node with a property *operation* (Change the ranking and model this scenario using generalization solution by creating required superclass and subclasses).

All nodes (artifacts) in the *Prescriptive Trace Graph* are traceable to the nodes (artifacts) of the *Descriptive Trace Graph* and the *Predictive Trace Graph* through *CandidateConcept* nodes (artifacts). We call the traceability relations across these three graphs *cross-graph traceability* (see Figure 2). The *Trace Layer* weaves all these three graphs and represents them together as the *TKG*. The *Modelling Decisions Traceability* exist between domain model elements and the *TKG* through the *Activated Qualifiers* as their actions create domain model elements as well as the *TKG* elements.

III. Experiment Design and Results

In this section, we first describe our tool and then discuss the three research questions that our evaluation addresses. Our tool and experiment data is available on the GitHub page [17].

A. Proposed Tool: We provide a proof-of-concept of our proposed approach in the form of a web-based prototypical tool. We use Neo4j [18] which is an open-source native graph knowledge database to persist the artifacts of the *TIM* and their relations in the form of *TKG*. In our tool, first, a modeller provides a problem description in NL. Second, the corresponding *Domain Model* and *TKG* are generated automatically. Finally, a modeller may select a modelling element in the generated *Domain Model* and request for its traceability using our tool. In response, our tool queries the *TKG* to retrieve the related sentences, entities, and rationale which are responsible for modelling the selected modelling element. The retrieved results are then presented back to the modeller by highlighting the associated sentences and words in the problem description. Also, the *Rationale* which was obtained during the extraction process is used as a template in the form of additional informative messages in NL to provide further insights into the modelling decisions to a modeller.

B. Research Questions: For each research question, we present our motivation, methodology, and results.

- **RQ1 (Effectiveness)** How effective is our approach in facilitating modelling decisions traceability?

Motivation: Our approach aims to enable traceability of modelling decisions in both directions – forward and backward. Some possible use case scenarios include: (a) a modeller selects an extracted model element to find the words or sentences in the problem description which are responsible for the selected model element or vice versa and (b) a modeller selects an extracted model element to find the similar model examples from the training data which enable the predictions from ML models for the selected model element.

Methodology: Since our approach uses a graph knowledge database (*TKG*), navigation (traceability) is trivially possible from both sides of a relationship. Therefore, we evaluate the traceability only in one direction (backward) which is currently implemented in our prototype tool. To assess the effectiveness of our approach, we use a set of nine problem descriptions (test dataset) which was never seen or touched while performing the design activities of our proposed approach. We collect these problem descriptions from the modelling exercises created by different instructors teaching software modelling courses at universities. These problems are free-form textual descriptions written in NL (English in our case) and they represent systems such as Bank Management System and Geographical Information System. The largest and smallest problem descriptions contain approx. 171 and 36 words, respectively.

First, we manually create a domain model for each problem description in the test dataset and trace the modelling elements back to the corresponding problem descriptions. Since the domain models and traceability relations constructed by one author are reviewed (corrected when required) by a second author, we consider this manually created data as the ground truth. Second, we manually calculate the *Total Relevant Links*, i.e., *TP* (True Positives) + *FN* (False Negatives), for each problem description. *TP* represents relevant relations in the retrieval and *FN* represents the relevant relations missing in the retrieval. Third, we use our tool for the test dataset and evaluate the traceability relations for each extracted domain model element. Fourth, we calculate the *Total Retrieved Links*, i.e., *TP* + *FP* (False Positives), and the *Total Relevant Retrieved Links*, i.e., *TP*, where *FP* represents the non-relevant relations in the retrieval. Finally, we compute precision [19], recall [19], and the F2 score [20]. We use F2 score (control parameter β is 2) to focus more on minimizing the relevant trace relations missing in our retrieval, as it is typically easier for a modeller to determine that an existing relation is incorrect than to detect missing relations. Moreover, we calculate the precision and recall scores of extracted models using our proposed approach and our previous work [13]. We consider our previous work as our baseline since the model extraction techniques in our proposed approach in this paper are based on our previous work with some differences which we discuss in Section VI.

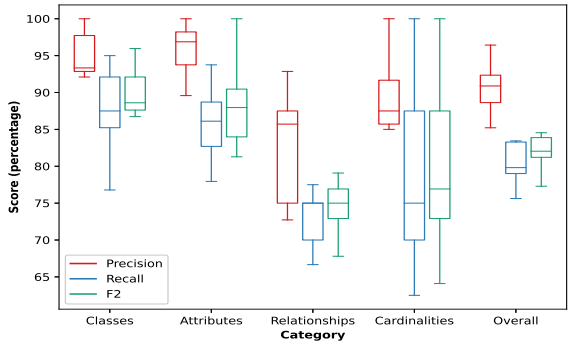

Fig. 4. Evaluation for Modelling Decisions Traceability.

Results: As shown in Figure 4, we use a boxplot graph to show the distribution of precision, recall, and F2 scores for each category. The traceability of *Attributes* achieves highest median precision (96.9) and that for *Classes* achieves highest

median recall (87.5) as well as F2 (88.61). The traceability of *Relationships* has the lowest median F2 score (75%). The median F2 scores for *Cardinalities* and *Overall* categories are 76.92% and 82.04%, respectively. Furthermore, precision (Pre), recall (Rec), and F2 ($F2$) scores for model extraction results using our approach (epr), extraction results using baseline (ebl), and traceability results (tr) for each problem description are shown in Table I. The average precision and recall scores of extracted models using our proposed approach are 90% and 77%, respectively, and those of extracted models using the baseline [5], [12] are 86% and 68%, respectively.

- **RQ2 (Performance)** To what degree does the model size influence the traceability results from our approach?

Motivation: In our tool, an event is triggered when a modeller performs an action such as selecting a domain model element to learn the extractor system's rationale in creating that element. The triggered event then generates a trace query to find the relevant nodes in the *TKG* including the *Rationale* nodes for the selected model element. The trace queries should return the results back to a modeller in a reasonable amount of time. Therefore, it is important to evaluate if the model size impacts the time performance of the trace queries.

Methodology: While calculating the metrics to answer the first research question (RQ1), we also compute the model size. For calculating the size of a model, we count the domain concepts (classes, attributes, relationships, and cardinalities) which are present in the ground truth domain model solutions for each problem description in the test dataset. Though a ground truth domain model represents an ideal solution for a problem description, multiple correct and equivalent model solutions with different model sizes may exist for the same problem description. Therefore, we report the minimum number of elements from these variations for each problem description in the test dataset. Finally, we use an automated timer which starts when we select a modelling element for trace in the domain model and ends when the trace query results are returned in the form of highlighted sections of problem description or the informative messages (*Rationale*).

Results: The minimum (11) and maximum (63) number of elements exist in the fifth and fourth problem description, respectively. The model extraction and the weaving of *TIM* together take 24.85 and 66.25 seconds for the fifth and fourth problem descriptions, respectively. The average time taken by trace queries for each problem description are shown in Table I. The time taken by trace queries for each problem description is around 50 milliseconds (ms). The trace queries take only 10 ms more in the fourth problem description. Therefore, the time performance of model extraction seems to be correlated highly and that of traceability seems to be correlated very weakly with the model size.

- **RQ3 (Usability)** What are the benefits and limitations of our prototypical tool?

Motivation: Our approach aims to assist modellers in extracting the domain models automatically and gaining insights into the modelling decisions taken by the extractor system.

Table I. MODEL SIZE IMPACT ON TRACEABILITY RESULTS

Category	Problem Descriptions (Test Dataset)								
Classes	11	9	5	15	3	7	8	12	5
Attributes	18	17	9	25	5	11	12	14	8
Relationships	10	7	3	9	1	4	5	7	2
Cardinalities	12	10	6	14	2	8	6	6	4
Total Elements	51	43	23	63	11	30	31	39	19
Pre_{epr} (%)	91	81	86	86	100	95	87	92	97
Pre_{ebl} (%)	81	77	79	79	100	88	81	90	97
Rec_{epr} (%)	71	67	77	62	88	84	78	80	84
Rec_{ebl} (%)	49	59	56	47	88	78	69	76	84
TP_{tr}	45	33	19	66	10	27	23	33	24
FP_{tr}	3	2	0	4	1	3	1	4	2
FN_{tr}	10	8	1	12	3	5	6	5	3
Pre_{tr} (%)	94	94	100	94	91	90	96	89	92
Rec_{tr} (%)	82	80	95	85	77	84	79	87	89
$F2_{tr}$ (%)	84	83	96	86	79	85	82	87	89
Time (ms)	40	48	38	50	40	57	60	39	40

Furthermore, we envision to enhance our tool in the form of a bot that can interact with modellers to improve their understanding of domain modelling decisions and update the domain models or problem descriptions when required. Therefore, an exploratory study is required to assess the benefits and limitations of our current tool. This will help in preparing our tool for a large-scale user study in the future.

Methodology: We conduct an exploratory user study where we hold semi-structured interviews with experienced modellers. We perform convenience sampling to recruit participants for this study. At a high level, the study is composed of three parts. The first part (5 minutes) focuses on information about the modellers, with demographic questions about their general modelling experience and their experience in domain modelling. The second part (10 minutes) demonstrates how our tool facilitates the traceability of modelling decisions. The third part (15 minutes) asks modellers to use our tool for first constructing a domain model automatically for a given problem description and then evaluating the usefulness and relevance of the generated trace links. Finally, the fourth part conducts an interview with the modellers.

Results: For our study, we invited 18 modellers who were never exposed to our tool's design activities. Among these 18 modellers, 11 participated (6 industry professionals and 5 graduate students) in the end. We observe that these participants have one to five years of general modelling experience (median of 3.5 years) and one to four years of domain modelling experience (median of 2 years). In the interview, the participants evaluate the trace query results and provide ratings for two aspects – usefulness and relevance, on a scale of 5 (1:low, 5:high). The relevance aspect evaluates quantitatively if the results of a trace query are applicable to the selected model element. By contrast, the usefulness aspect evaluates qualitatively if the results are information-rich (e.g., informative messages) to enlighten the modellers about the modelling decisions for the selected elements. Our tool achieves an average of 4.8 for the relevance aspect and 4

for the usefulness aspect. We discuss the lessons learned from this study in the next section.

IV. Discussion

In this section, we discuss the main take-away messages of our evaluation and exploratory study, in relation to the RQs.

A. Effectiveness (RQ1): The results of model extraction may impact the traceability results in some cases, e.g., required concepts are missing in the extracted model. In other cases, model extraction results may not have a direct impact on the traceability results. For example, a correctly identified domain concept (with retrieved relevant trace links) improves the traceability results but may impact the extraction results when not modelled correctly, e.g., a class concept is modelled as an attribute but the attribute is traced correctly, i.e., an incorrect model element with correct trace links improves the traceability results but impacts the extraction results. Moreover, the traceability results may be even worst for a correctly extracted domain model, i.e., when the artifacts are not woven well in the TIM. Therefore, we first evaluate the model extraction results of our approach and baseline. Our approach achieves average precision and recall scores higher than those of baseline by 4% and 9%, respectively.

Next, we evaluate the traceability results. Following our intuition, the evaluation results (see Figure 4) indicate that our approach works well for the domain concepts categories – classes and attributes. However, the F2 scores are lower for other categories – relationships and cardinalities. We further observe that the traceability results (particularly recall) depend on model extraction results. The root cause of missing traces lies in domain models extracted using our approach missing some relationships and cardinalities which are present in the ground truth domain models. Also, in some cases, the concepts are identified correctly but modelled incorrectly using relationships and cardinalities. Furthermore, in some cases, the results (highlighted words) are not presented properly even when the retrieved meta-data is correct. Therefore, it may be required to improve the model extraction process and the presentation of traceability results but not the construction of the *TKG* to use these trace links effectively.

B. Performance (RQ2): We execute the experiments on a machine with Ubuntu (20.04.2 LTS), an intel i7 processor with 2.70GHz, and 20Gb of RAM memory. The results from the second research question imply that the model size does not or very minimally affects the time performance of trace queries and the F2 scores, in the case of test dataset problem descriptions. To further evaluate the impact of model size on performance, we create a problem description by combining all problem descriptions so that the combined problem description contains a significantly higher number of domain concepts (i.e., more than 300). For this problem description, trace queries still only take an average of 54 ms.

C. Lessons Learned from Study (RQ3): First, the exploratory study identifies areas of improvement for the future large-scale study design. For example, an additional part can be added to the study which asks participants to manually establish the trace links and measures their time. This will help in comparing the times (manual tracing and automated tracing using our approach) and reflecting the time cost associated with manual domain modelling decisions traceability which may be avoided when using our automated approach. Second, participants agree that manual traceability is hard. Participant A commented that for relatively large problem descriptions "it is impossible to do [perform the traceability of modelling decisions] by hand". Third, participants agree that our tool generates the traceability information (highlighting sentences and words in problem description as well as the informative messages) in real-time. Participant E explains that the use of automated extraction methods can save time if one can also quickly validate models for their completeness and preciseness by learning the rationale behind modelling decisions. This will allow to update models or problem descriptions in a small amount of time. Fourth, participants agree that our tool improves their awareness of the modelling decisions taken by the extractor system. Participant C commented that the direct integration of traceability knowledge base into the model extraction process shows plenty promise. Finally, participants indicate other intuitive ways to present the scope of a modelling decision. For example, reflecting the scope of a decision that uses the *Player-Role* pattern by highlighting the required classes and their relationships (model fragment).

V. Threats to Validity

In this section, we discuss the validity factors most pertinent to the evaluation of our proposed approach.

A. Construct Validity: We construct the ground truth which represents the manually created domain model solutions as well as the traceability relations between domain model elements and problem description to calculate the *Total Relevant Links*. Threats to construct validity have to do with the construction of the ground truth data. First, the ground truth data may be the result of the same reasoning which we used while developing the heuristics for the model extraction process. We mitigate this threat by including variations in the ground truth domain model created by modellers or instructors other than the authors. While comparing the results of our proposed approach, we ensure that the results match the variations at least semantically if not structurally. Second, the construction of this ground truth may include some errors. However, we mitigate this threat by a second author reviewing the ground truth data created by one author so that the decisions can be cross-checked and confirmed. Also, if there are any conflicts then both authors discuss the conflicts and resolve them by making the required changes to the ground truth data.

B. External Validity: Threats to external validity have to do with the generalizability of our results to other problem descriptions. First, we show that median F2 scores are higher for concepts (classes and attributes) than other categories (relationships and cardinalities). It may be possible that for the problem descriptions in the test dataset, it is direct and easy to extract classes and attributes and to create traceability links for them. Though we ensure that the randomly selected problem

descriptions represent all the relationships and the *Player-Role* modelling pattern, these problems still cannot cover all possible situations in domain modelling. Second, the problem descriptions in the test dataset may not represent a wide sample of domain problems in terms of problem complexity and model size. To reduce this threat to some extent, we create a larger problem description with more than 300 model elements and evaluate the time performance. In the future, we plan to perform an evaluation with a bigger and diverse dataset.

C. Internal Validity: We perform convenience sampling to recruit participants for our study. Our tool shows promise based on the interview results. Threats to internal validity have to do with the participants' feedback as the participants may be biased towards providing positive feedback to us due to social pressure. For the future large-scale study, we plan to use other methods of recruitment, so that the participants are more representative of the general population of modellers.

VI. Related Work

We organize the related work into two areas: (1) model extraction and (2) traceability using trace links and artifacts.

A. Model Extraction: Several approaches exist that use NLP based techniques alone to extract models from text [6], [7], [8], [9], [21], [22], [23]. Arora et al. combine model extraction rules with complementary rules from the information retrieval literature to extract domain models [24]. In addition, the use of ML techniques exists in several works such as demarcating requirements in requirements specifications [25], automatically classifying content elements of NL requirements [26], and using neural networks for relation classification in NLP [27]. In contrast, our proposed approach is build upon our previous work for model extraction which combines NLP and ML techniques to generate domain models [5], [12], [13]. Our proposed approach differentiates our previous work in the following ways: (1) We use the CoReference Resolution technique and collect the generated references separately to be used during traceability; (2) In the *Prescriptive Layer*, we generate ranked model slices by using the classifiers' results sequentially based on their performances, e.g., using *Concepts Category* classifier followed by *Concepts Type* classifier and so on; (3) The goal of our previous work is to extract domain models with high accuracy [5], [12], [13], while in contrast, the main focus of our proposed approach is to facilitate traceability of modelling decisions in forward and backward directions; and (4) Our previous work envisions to create and use trace links which can be generated during model extraction process [5], [12], [13], [28], [29], while, we weave our proposed *TIM* directly with the model extraction process and instantiate it in the form of *TKG* in this paper.

B. Traceability of Requirements: In the literature, multiple definitions of ambiguities in NL and the challenges to detect them have been discussed [30], [31], [32], [33], [34], [35], [36], [37], [38]. Our work focuses on establishing traceability relations between extracted model elements and requirements to validate domain models for their completeness and preciseness based on the requirements in a problem description. Several works also exist which provide traceability between different artifacts, e.g., traceability between UML diagrams and target models [39], requirements-level and code aspects [40], class entities and sections in NL documents [41], and tracing information when concerns are reused [42]. Also, Hübner et al. develop an interaction log-based trace link creation approach [43] and Houmb et al. present a tracing approach for security requirements elicitation [44]. In contrast, our approach provides traceability between NL problem descriptions and the generated domain models. In addition, several works exist which define and use traceability metamodels, e.g., trace metamodel for Domain-Specific Modelling Languages [45], trace metamodel to capture non-functional requirements and their relations [46], model for defining traceability metagraphs [47], and specification of a metamodel in the context of software product line [48]. On the other side, in our work, we present a *TIM* for automated traceability of domain modelling decisions. The benefits of a *TIM* such as automated traceability handling, validation, and analyses have been discussed widely [49], [50], [51]. Schlutter and Vogelsang transform a set of heterogeneous NL requirements into a knowledge representation graph [52]. In contrast, we weave our proposed *TIM* with extracted model elements and their meta-data in the form of *TKG*. Finally, some work proposes traceability benchmarking for evaluation and comparison of traceability techniques [53], [54], [55]. To the best of our knowledge, no existing approach provides end-to-end traceability for domain modelling decisions.

VII. Conclusions and Future Work

The manual practice of domain modelling requires time, modelling skills, knowledge, and experience. Therefore, several approaches have been proposed to assist modellers by automating or semi-automating the construction of domain models. However, existing approaches do not provide any support for traceability of modelling decisions. In this paper, we propose an automated approach that enables modellers to trace modelling decisions of domain model elements. Furthermore, we evaluate our approach using an unseen set of problem descriptions where our approach achieves an overall median F2 Score of 82.04%. Also, we conduct an exploratory user study and present the lessons learned from this study.

In the future, first, we plan to provide tool support for forward traceability in terms of modeller-tool interactions and visualizations. Second, we plan to perform a large-scale user study with modellers so that the relevance and usefulness of our tool in facilitating the traceability of modelling decisions can be assessed more thoroughly. Third, we plan to evaluate our approach with more problem descriptions and also with those problem descriptions which are very long with model size beyond 100 elements. Fourth, we aim to improve the model extraction process by enhancing the qualifiers in all three layers so that the trace links can be used effectively. Finally, we plan to enhance our tool in the form of a bot with a question answering unit to have meaningful conversations with modellers about the modelling decisions, particularly with novice modellers to enhance their modelling knowledge.

REFERENCES

[1] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, "What are the used uml diagrams? a preliminary survey." *EESSMOD@ MoDELS*, vol. 1078, no. 10, 2013.

[2] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: The best is yet to come," *IEEE software*, vol. 35, no. 5, pp. 115–119, 2018.

[3] D. M. Berry and E. Kamsties, *Ambiguity in Requirements Specification*. Boston, MA: Springer US, 2004, pp. 7–44.

[4] 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.

[5] 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.

[6] M. Ibrahim and R. Ahmad, "Class diagram extraction from textual requirements using natural language processing (nlp) techniques," in *2010 Second International Conference on Computer Research and Development*. IEEE, 2010, pp. 200–204.

[7] M. Jyothilakshmi and P. Samuel, "Domain ontology based class diagram generation from functional requirements," in *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*. IEEE, 2012, pp. 380–385.

[8] M. Landhäußer, S. J. Körner, and W. F. Tichy, "From requirements to uml models and back: how automatic processing of text can support requirements engineering," *Software Quality Journal*, vol. 22, no. 1, pp. 121–149, 2014.

[9] O. Salih Dawood and A.-E.-K. Sahraoui, "From Requirements Engineering to UML using Natural Language Processing – Survey Study ," *European Journal of Industrial Engineering*, vol. 2, pp. 44–50, 2017.

[10] 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.

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

[12] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "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.

[13] 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

[14] spaCy., https://spacy.io/, Last Accessed 2021.

[15] 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

[16] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[17] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, https://rijul5.github.io/Domain-Modelling-Traceability/, Last Accessed 2021.

[18] Neo4j., https://neo4j.com/, Last Accessed 2021.

[19] Wikipedia., https://en.wikipedia.org/wiki/Precision_and_recall, Last Accessed 2021.

[20] ——, https://en.wikipedia.org/wiki/F-score, Last Accessed 2021.

[21] R. Sharma, P. K. Srivastava, and K. K. Biswas, "From natural language requirements to uml class diagrams," in *AIRE 2015*. IEEE, 2015, pp. 1–8.

[22] A. Montes, H. Pacheco, H. Estrada, and O. Pastor, "Conceptual model generation from requirements model: A natural language processing approach," in *International Conference on Application of Natural Language to Information Systems*. Springer, 2008, pp. 325–326.

[23] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via nlp," in *RE 2016*. IEEE, 2016, pp. 196–205.

[24] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting domain models from natural-language requirements: approach and industrial evaluation," in *MODELS 2016*. ACM, 2016, pp. 250–260.

[25] S. Abualhaija, C. Arora, M. Sabetzadeh, L. C. Briand, and E. Vaz, "A machine learning-based approach for demarcating requirements in textual specifications," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, pp. 51–62.

[26] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2016, pp. 39–45.

[27] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, "Classifying relations via long short term memory networks along shortest dependency paths," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1785–1794.

[28] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Teaching modelling literacy: An artificial intelligence approach," in *MODELS 2019 Companion*, Sep. 2019, pp. 714–719.

[29] R. Saini, "Artificial intelligence empowered domain modelling bot," 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/10.1145/3417990.3419486

[30] D. M. Berry, P. D. C. Science, M. M. Krieger, and P. D. Mathematics, "From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0," 2000.

[31] D. M. Berry and E. Kamsties, "Ambiguity in requirements specification," in *Perspectives on software requirements*. Springer, 2004, pp. 7–44.

[32] D. M. Berry, "Ambiguity in natural language requirements documents," in *Monterey Workshop*. Springer, 2007, pp. 1–7.

[33] A. K. Massey, R. L. Rutledge, A. I. Antón, and P. P. Swire, "Identifying and classifying ambiguity for regulatory requirements," in *2014 IEEE 22nd international requirements engineering conference (RE)*. IEEE, 2014, pp. 83–92.

[34] A. Ferrari, G. Lipari, S. Gnesi, and G. O. Spagnolo, "Pragmatic ambiguity detection in natural language requirements," in *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, 2014, pp. 1–8.

[35] V. Gervasi and D. Zowghi, "On the role of ambiguity in re," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2010, pp. 248–254.

[36] S. Ezzini, S. Abualhaija, C. Arora, M. Sabetzadeh, and L. Briand, "Using domain-specific corpora for improved handling of ambiguity in requirements," in *In Proceedings of the 43rd International Conference on Software Engineering (ICSE'21), Madrid 25-28 May 2021*, 2021.

[37] E. Kamsties and B. Peach, "Taming ambiguity in natural language requirements," in *Proceedings of the Thirteenth international conference on Software and Systems Engineering and Applications*, 2000.

[38] A. Ferrari and A. Esuli, "An nlp approach for cross-domain ambiguity detection in requirements engineering," *Automated Software Engineering*, vol. 26, no. 3, pp. 559–598, 2019.

[39] R. K. Stirewalt, M. Deng, and B. H. Cheng, "Uml formalization is a traceability problem," in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, 2005, pp. 31–36.

[40] A. Sardinha, Y. Yu, N. Niu, and A. Rashid, "Ea-tracer: identifying traceability links between code aspects and early aspects," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 1035–1042.

[41] X. Chen, J. Hosking, and J. Grundy, "A combination approach for enhancing automated traceability: (nier track)," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 912–915.

[42] M. Schöttle, N. Thimmegowda, O. Alam, J. Kienzle, and G. Mussbacher, "Feature modelling and traceability for concern-driven software development with touchcore," in *Companion Proceedings of the 14th International Conference on Modularity*, 2015, pp. 11–14.

[43] P. Hübner and B. Paech, "Interaction-based creation and maintenance of continuously usable trace links between requirements and source code," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4350–4377, 2020.

[44] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider, "Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec," *Requirements Engineering*, vol. 15, no. 1, pp. 63–93, 2010.

[45] E. Bousse, T. Mayerhofer, B. Combemale, and B. Baudry, "A generative approach to define rich domain-specific trace metamodels," in *European*

Conference on Modelling Foundations and Applications. Springer, 2015, pp. 45–61.

[46] M. Kassab, O. Ormandjieva, and M. Daneva, "A traceability metamodel for change management of non-functional requirements," in *2008 Sixth International Conference on Software Engineering Research, Management and Applications*. IEEE, 2008, pp. 245–254.

[47] J. Cleland-Huang, J. H. Hayes, and J. M. Domel, "Model-based traceability," in *2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, 2009, pp. 6–10.

[48] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.-C. Royer, A. Rummler, and A. Sousa, "A model-driven traceability framework for software product lines," *Software & Systems Modeling*, vol. 9, no. 4, pp. 427–451, 2010.

[49] F. A. Pinheiro and J. A. Goguen, "An object-oriented tool for tracing requirements," *IEEE software*, vol. 13, no. 2, pp. 52–64, 1996.

[50] P. Mader, O. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, 2009, pp. 21–25.

[51] S. Maro, J.-P. Steghofer, E. Knauss, J. Horkof, R. Kasauli, R. Wohlrab, J. L. Korsgaard, F. Wartenberg, N. J. Strøm, and R. Alexandersson, "Managing traceability information models: Not such a simple task after all?" *IEEE Software*, pp. 0–0, 2020.

[52] A. Schlutter and A. Vogelsang, "Knowledge representation of requirements documents using natural language processing," in *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018), Utrecht, The Netherlands, March 19, 2018*, ser. CEUR Workshop Proceedings, K. Schmid, P. Spoletini, E. B. Charrada, Y. Chisik, F. Dalpiaz, A. Ferrari, P. Forbrig, X. Franch, M. Kirikova, N. H. Madhavji, C. Palomares, J. Ralyté, M. Sabetzadeh, P. Sawyer, D. van der Linden, and A. Zamansky, Eds., vol. 2075. CEUR-WS.org, 2018. [Online]. Available: http://ceur-ws.org/Vol-2075/NLP4RE_paper9.pdf

[53] E. Ben Charrada, D. Caspar, C. Jeanneret, and M. Glinz, "Towards a benchmark for traceability," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, 2011, pp. 21–30.

[54] X. Chen, J. Hosking, J. Grundy, and R. Amor, "Development of robust traceability benchmarks," in *2013 22nd Australian Software Engineering Conference*. IEEE, 2013, pp. 145–154.

[55] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, O. Gotel, J. H. Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder, "Grand challenges, benchmarks, and tracelab: Developing infrastructure for the software traceability research community," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, ser. TEFSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 17–23. [Online]. Available: https://doi.org/10.1145/1987856.1987861