This version is a pre-print. Please refear to the publisher (Springer) to access the post-edition paper version here: https://doi.org/10.1007/978-3-031-29786-1_10
How to cite: Mosquera, D., Ruiz, M., Pastor, O., Spielberger, J. (2023). Ontology-Based Automatic Reasoning and NLP for Tracing Software Requirements into Models with the OntoTrace Tool. In: Ferrari, A., Penzenstadler, B. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2023. Lecture Notes in Computer Science, vol 13975. Springer, Cham. https://doi.org/10.1007/978-3-031-29786-1_10

Ontology-based Automatic Reasoning and NLP for Tracing Software Requirements into Models with the OntoTrace Tool

David Mosquera[1][0000-0002-0552-7878], Marcela Ruiz[1][0000-0002-0592-1779], Oscar Pastor[2][0000-0002-1320-8471], and Jürgen Spielberger[1][0000-0003-2617-3535]

[1] Zürich University of Applied Sciences, Gertrudstrasse 15, Winterthur 8400, Switzerland
{mosq, ruiz, spij}@zhaw.ch
[2] PROS-VRAIN: Valencian Research Institute for Artificial Intelligence - Universitat Politècnica de València, València, Spain
opastor@dsic.upv.es

Abstract. Context and motivation. Traceability is an essential part of quality assurance tasks for software maintainability, validation, and verification. However, the effort required to create and maintain traces is still high compared to their benefits. **Problem**. Some authors have proposed traceability tools to address this challenge, yet some of those tools require historical traceability data to generate traces, representing an entry barrier to software development teams that do not do traceability. Another common requirement of existing traceability tools is the scope of artefacts to be traced, hindering the adaptability of traceability tools in practice. **Principal ideas**. Motivated by the mentioned challenges, in this paper we propose OntoTraceV2.0: a tool for supporting trace generation of arbitrary software artefacts without depending on historical traceability data. The architecture of OntoTraceV2.0 integrates ontology-based automatic reasoning to facilitate adaptability for tracing arbitrary artefacts and natural language processing for discovering traces based on text-based similarity between artefacts. We conducted a quasi-experiment with 36 subjects to validate OntoTraceV2.0 in terms of efficiency, effectiveness, and satisfaction. **Contribution.** We found that OntoTraceV2.0 positively affects the subjects' efficiency and satisfaction during trace generation compared to a manual approach. Although the subjects' average effectiveness is higher using OntoTraceV2.0, we observe no statistical difference with the manual trace generation approach. Even though such results are promising, further replications are needed to avoid certain threats to validity. We conclude the paper by analysing the experimental results and limitations we found, drawing on future challenges, and proposing the next research endeavours.

Keywords: Traceability, Ontology, NLP, Automatic reasoning, OntoTrace

1 Introduction

Traceability in software development refers to generating, maintaining, and using traces between software artefacts [1, 2]. A trace is a triplet of elements composed of a source

artefact, a target artefact, and a trace link [2]. Software artefacts vary depending on the software development context and can be of different formats such as: textual requirements, source code, mock-ups, test cases, graphical software models, among others. Keeping such artefacts traced is essential to quality assurance tasks such as software maintainability, validation, and verification [3, 4]. However, in practice, the effort required to trace artefacts outweighs traceability benefits [5]. Thus, some authors have proposed novel approaches, especially for generating traces between software artefacts [5–15]. These proposals have attempted to decrease the effort required for generating traces between artefacts. Yet, some of them depend on historical traceability data—a.k.a. training traceability data—[5, 7, 9, 14], are fixed to specific artefact types [8, 10–13], and lack decision-making support techniques for trace generation [6, 15].

We had previously conceived OntoTrace: a tool for supporting trace generation of arbitrary software artefacts using ontology-based automatic reasoning [15] (see OntoTraceV2.0 research timeline in Fig. 1). In this paper, we evolve OntoTrace into OntoTraceV2.0 providing it with a Natural Language Processing (NLP) layer that support decision-making on generating traces between artefacts. Although OntoTrace supports trace generation of arbitrary software artefacts, we scope our research to trace software requirements—i.e., user stories—into software models—i.e., Existence Dependency Graph (EDG) models. Thus, OntoTrace users can use an automatic reasoner together with NLP to infer traceability-related information such as: i) which artefacts are not yet traced; ii) which are the traceable source/target artefacts; and iii) given a specific artefact, which are the possible recommended traces between it and other artefacts based on text-based similarity.

We conducted a quasi-experiment with 36 subjects to validate OntoTraceV2.0 in terms of subjects' efficiency, effectiveness, and satisfaction in the context of the rapid software prototyping course at the Zürich University of Applied Sciences (ZHAW). Experimental results show how OntoTraceV2.0 positively affects the subjects' efficiency and satisfaction during trace generation compared to a manual approach. Although the subjects' average effectiveness is higher using OntoTraceV2.0, we observed no statistical difference with the manual trace generation approach in terms of effectiveness. Even though such results are promising, we identified some validity threats such as maturity, low statistical power, and generality threats that requires further replications to validate our results. Finally, we discuss our conclusions and the subsequent challenges to a complete technology transference.

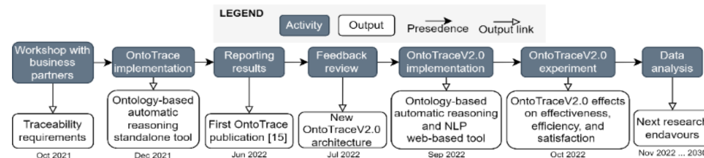

Fig. 1. OntoTraceV2.0 research timeline and overview.

The paper is structured as follows: in Section 2, we review the related works; in Section 3, we exhibit the problem scope, main definitions, and exemplify how to configure OntoTrace for tracing user stories and EDG models [16]; in Section 4, we present

all new features included in OntoTraceV2.0; in Section 5, we show the OntoTraceV2.0 validation results; and, finally, in Section 6, we discuss conclusions and future work.

2 Related works

Automating totally or partially the trace generation in software development has gained researchers' attention. Thus, they have proposed novel traceability tools. Some authors propose tools for generating traces between artefacts based on historical traceability data—a.k.a. training traceability data—such as: artificial neural networks [5, 14], historical-similarity-based algorithms [9], and Bayes classifiers [7]. Although these tools are helpful, they depend on extensive and well-labelled training data sets based on historical traceability data. This represents an entry barrier for software development teams that currently do not trace their artefacts. Other authors propose tools that do not rely on historical traceability data, such as ontology-based recommendation systems [12, 13], expert systems [8], pattern languages [11], and metamodel-based ontologies [10]. However, these tools are limited to generating traces between specific artefacts. Thus, software development teams cannot adapt such tools to their software development traceability needs. For instance, some tools [8, 12, 14] limit their source/target artefacts to text-based artefacts—e.g., source code, standards, and textual requirements. Therefore, non-textual artefacts such as models, UIs, and mock-ups are beyond their scope. Having that in mind, in previous work we have proposed OntoTrace as a tool for generating traces between arbitrary artefacts without the need to rely on historical traceability data [15]. Nevertheless, like the Capra tool proposed in [6], they both lack decision-making support for analysts to decide on which traces need to be generated—i.e., both lack support for recommending which artefacts should be traced.

To address such gaps, we propose to evolve OntoTrace to OntoTraceV2.0: an ontology-based automatic reasoning NLP (Natural Language Processing) tool for generating traces between software artefacts. Like its predecessor, OntoTraceV2.0 does not rely on historical traceability data and is not restricted to a specific set of traceable artefacts. In addition, we combine automatic reasoning with NLP to support decision-making on which traces should be generated between artefacts. Thus, OntoTraceV2.0 is a step forward in improving software trace generation, having such combination as the main technical novelty.

3 Problem Scope

The goal of this paper is: to ***analyse*** *the OntoTrace tool* **for the purpose of** *supporting software traceability* **with respect to** *effectiveness, efficiency, and satisfaction of OntoTrace users* **from the point of view of** *the researchers* **in the context of** *software trace generation tasks*. To address this goal, we have taken the following decisions:

- The OntoTrace tool proposed in [15] is founded on general traceability definitions taken from [1, 2, 17], which supports trace generation in any traceability context. We define ***traceability context*** as the set of SOURCE and TARGET software artefacts

to be connected by means of traces. For instance, the traceability context of this paper and the controlled quasi-experiment presented in Section 5 is the generation of traces between User Stories [18] as SOURCE and EDG models—a UML-class-diagram-like model [16]—as TARGET software artefacts. The reason is that User Stories and UML models are widely used by software development teams to document software requirements. Moreover, EDG models are supported by teaching and learning tools like Merlin, which are good fit for teaching and experimental purposes [16].
- We define ***traceability activity*** as any activity involved in the traceability process such as generating, using, and maintaining traces [2].
- We define ***OntoTrace user*** as any software development team role carrying out a traceability activity using OntoTrace [15].
- The traceability activity that we select for this paper is ***trace generation***. Other traceability activities are out of this paper's scope.

Based on these decisions, we propose the following research questions:
RQ1: How to improve OntoTrace to allow for automatic trace recommendations? We consider different NLP techniques [19–22] to provide trace recommendations between artefacts and refactor the OntoTrace architecture [15], reflecting all new features. As a result, we propose OntoTraceV2.0.
RQ2: When the subjects use OntoTraceV2.0, is their effectiveness, efficiency, and satisfaction in establishing traceability links among User Stories and EDG models affected? To answer this question, we conduct a quasi-experiment to compare effectiveness, efficiency, and satisfaction of subjects that did software traceability with OntoTraceV2.0 and the traditional way (without OntoTraceV2.0).

3.1 Traceability context: Tracing user stories and EDG models

In this Section we show the application of the method Ontology101 [23] to establish the traceability context for this paper: *Tracing User Stories* [18] *as SOURCE and EDG models* [16] *as TARGET software artefacts*. As a result, we create an ontology based on such traceability context, containing the structure of artefacts and traces. This ontology is the main input for using OntoTrace since it relies on automatic reasoning based on the defined ontological structure. We present a summary with the application of each Ontology101 step, a set of guidelines to specialise each Ontology101 step for establishing the traceability context, and the outcome of applying each guideline (see Table 1).

Table 1. Establishing traceability context:

Ontology101 Step (S)	Guidelines (G) for establishing traceability context	Result: traceability context user stories and EDG
S1: Determine the domain and scope of the ontology	**G1.** Specify context-dependent artefacts that require to be traced	User story parts EDG model elements
	G2. Classify the context-dependent artefacts into source and target artefacts	Source: User story parts Target: EDG model elements

S2: Consider reusing existing ontologies	**G3.** Reuse existing metamodels, tools, domain models, syntaxes, documentation, libraries, and vocabulary that describe context-dependent artefacts	We reuse the following ontology and metamodel: - Ontology for User Stories [18] - EDG metamodel [16]
S3: Enumerate important terms in the ontology	**G4.** List terms representing context-dependent source artefacts	User story role, user story action, user story goal, user story object
	G5. List terms representing context-dependent source artefacts	EDG object, EDG attribute, EDG dependency, EDG method
	G6. Specify context-dependent trace properties to link source and target artefacts	We propose the traceability matrix in Table 2 based on literature on transforming user stories into EDG models [24, 25] (EDG can be transformed into UML and vice versa [16]). This traceability matrix represents the context-dependent trace properties based on researchers' [24, 25] and authors' experience.
S4: Define the classes and the class hierarchy	**G7.** Use the class hierarchy for defining the *source/target* sub-classes based on the resulting terms from G4 and G5.	See Fig. 2.
	G8. Use the class hierarchy from G7 for defining the *trace* sub-classes based on the resulting trace properties from G6. Each sub-class relates to a traceability link defined as follows: *Trace hasSource some Source AND Trace hasTarget some Target.* Constraint: Define *trace* sub-classes until all possible trace properties resulting from G6 have been covered with at least one trace sub-class.	
S5: Define the properties of classes	**G9.** Define context-dependent traceability properties with the following naming: *has + Source/Target + Artefact Name.* Constraint: All target/source sub-classes must be related to at least one traceability-related property.	Property *hasSourceUserStoryRole*, inheriting from the *hasSource* property. Property *hasTargetEDGObject,* inheriting from the *hasTarget* property.
S6: Define the facets of the properties	**G10.** Define the range and domain of context-dependent traceability properties as follows: Set the domain as all possible trace sub-classes from G8 that have the source/target artefact as its range. Set the range as the source/target artefact of the trace.	***hasSourceUserStoryRole* property:** this property's domain is a *Trace Between User Story Role and EDG Object* class instance, and its range is *User Story Role* class instances. ***hasTargetEDGObject* property:** this property's domain is a *Trace Between User Story Role and EDG Object* class instance, and its range is *EDG Object* class instances.

S7: Create instances	G11. For each artefact, select one of the following individual instance creation strategies: *Manual:* Artefacts are difficult to access programmatically, such as physical documentation. *Automatic*: Artefacts are contained in accessible repositories allowing for programmatic retrieval operations.	***Automatic strategy*** for creating source artefacts since User Stories are stored digitally. ***Automatic strategy*** for EDG models since they are digital

Table 2. Traceability matrix in our running example: User story parts vs EDG model elements.

Source artefact	EDG: Target Artefact			
	Object	Attribute	Dependency	Method
User story role	✓			
User story action			✓	✓
User story object	✓	✓		
User story goal		✓	✓	

✓: Traceability link; **EDG**: Existence Dependency Graph

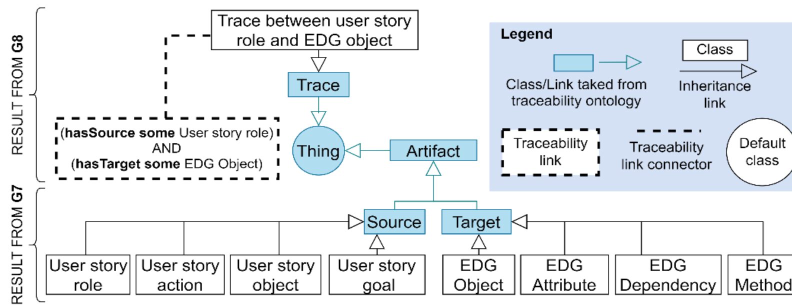

Fig. 2. Excerpt of trace sub-classes, hierarchy, and traceability links of our running example.

Having the strategy selected from G11, the context-dependent traceability ontology is ready to be translated into a computational-readable knowledge representation language as OWL (Ontology Web Language [26]) and then used with OntoTrace.

4 Evolving OntoTrace into OntoTraceV2.0[1]

In previous work, we proposed OntoTrace as an ontology-based automatic reasoning trace generation tool [15]. In this Section, we address RQ1 presented in Section 3, improving OntoTrace to allow for automatic trace recommendation. We show which are the new OntoTraceV2.0 architecture elements in Fig. 3 compared to OntoTrace. Moreover, we describe the new modules in the following paragraphs.

[1] OntoTraceV2.0 code is available here: https://tinyurl.com/4d45utrf

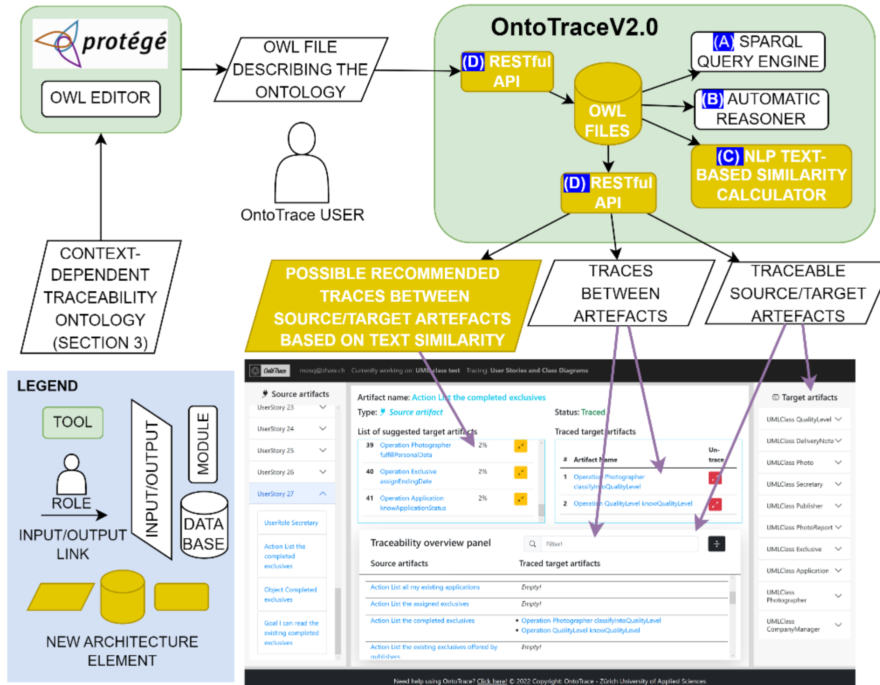

Fig. 3. OntoTraceV2.0 architecture overview

First, OntoTrace user specifies a context-dependent ontology (see Section 3.1) and creates an OWL file [26] describing it using an external tool such as Protégé. Then, the OntoTrace user provides this OWL file to OntoTrace to use the following modules:

- **Module A.** OntoTrace provide information about source and target artefacts using a set of SPARQL queries. All the information is retrieved using the context-dependent traceability ontology.
- **Module B.** OntoTrace uses an automatic reasoner together with the SPARQL query engine to answer the following traceability related questions: i) which source/target artefacts are traceable; ii) which are the traces between artefacts; iii) which possible traces exist between source/target artefacts.

Module A and B allow OntoTrace users to store traces between artefacts based on automatic reasoning. Nevertheless, OntoTrace cannot recommend which of the possible source/target artefacts are relevant to be traced. This is problematic, mainly when an artefact can be traced to many different artefacts, motivating us to evolve OntoTrace [15] and propose OntoTraceV2.0. Therefore, we create a new web-based user interface and include the following two modules to OntoTraceV2.0:

- **Module C.** We provide OntoTraceV2.0 with an NLP layer for suggesting traces between artefacts based on their text-based similarity (see Section 4.1), addressing the aforementioned gap.
- **Module D.** Now, OntoTraceV2.0 is a web-based tool instead of a standalone tool. OntoTrace users uses a RESTful API to access all OntoTraceV2.0 functionalities.

4.1 Combining NLP and Ontology-based automatic reasoning for supporting trace generation between user stories and EDG models

OntoTrace [15] have a limitation on not recommending which possible source/target artefacts are relevant to be traced. In this Section, we use the context-dependent traceability ontology defined in Section 3.1 to exemplify this limitation and show how NLP can solve it.

After having the context-dependent traceability ontology (see Section 3.1), OntoTrace users start populating OntoTrace with user story parts and EDG model elements. These artefact instances represent the set of all traceable artefacts A. We divide A into two subsets: user story parts (source artefacts) $S_A \subseteq A$ and EDG model elements (target artefacts) $T_A \subseteq A$. OntoTrace uses ontology-based automatic reasoning to answer traceability related questions, creating subsets of S_A and T_A. Specially, we focus on the following traceability-related question[2]: having selected a user story part $s_a \in S_a$, which is the set of possible EDG model elements $PT_a \subseteq T_a$ to trace? OntoTrace automatic reasoner answers this question creating the PT_a subset based on the context-dependent trace properties (see Table 2). Now, the OntoTrace user can select one of the possible EDG model elements $pt_a \in PT_a$ to create a trace with s_a. For instance, the OntoTrace user selects a User Story Role Secretary and OntoTrace answers based on the context-dependent trace properties (see Table 2) with the following possible EDG model elements PT_a to trace: EDG Object Aircraft Manager, EDG Object Aircraft, and EDG Object Secretary. Now, the OntoTrace user can select EDG Object Secretary as pt_a to create a trace with the User Story Role Secretary as s_a. We graphically show this example in Fig. 4.

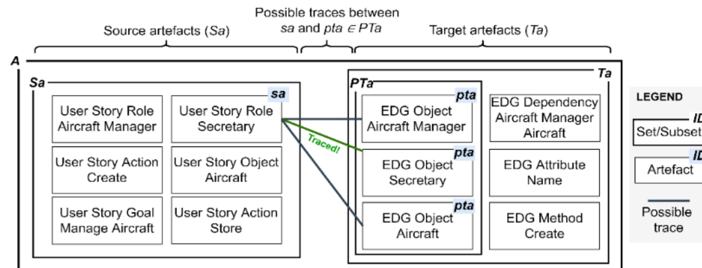

Fig. 4. Previous OntoTrace version automatic reasoning result.

Notice that OntoTrace's automatic reasoner filtered out all target artefacts that the OntoTrace user must not trace to a User Story Role, such as EDG Dependencies, Methods, and Attributes. However, the OntoTrace user still needs to decide which EDG model element pt_a from the PT_a subset is the correct one to trace. Whether all possible EDG model elements $pt_a \in PT_a$ are equally valid is a problem that limits the scope of the automatic reasoner for trace generation. To address such a problem, we provide

---

[2] Notice that this question can also be written as: having selected a EDG model element $t_a \in T_a$, which is the set of possible user story parts $PS_a \subseteq S_a$ to trace? However, we use the source-to-target variant instead of target-to-source variant for simplicity.

OntoTraceV2.0 with an NLP layer to recommend which EDG model element pt_a is relevant to be traced to a selected user story part s_a based on text-based similarity. OntoTraceV2.0's NLP layer comprises three sub-layers: extracting artefacts' text data, processing extracted text, and calculating the similarity between artefacts. As a result, OntoTraceV2.0 provide a similarity value with the possible traces between artefacts. We show how OntoTraceV2.0 transforms ontology-based automatic reasoning output using the NLP layer in Fig. 5.

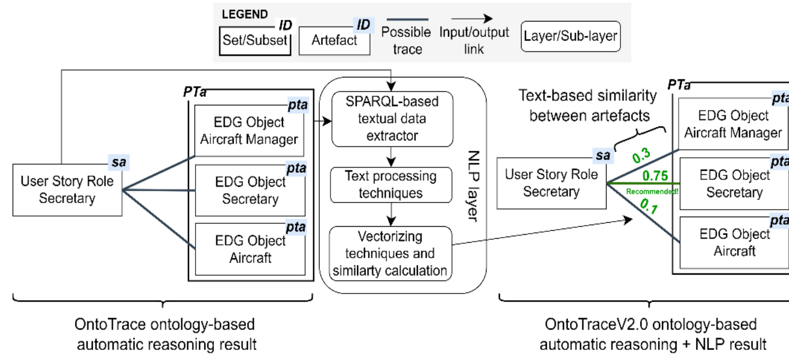

Fig. 5. OntoTraceV2.0 NLP layer and sublayers explained.

In the first sub-layer, we extract the textual data from the selected user story part s_a and all possible EDG model elements $pt_a \in PT_a$. We gather artefacts' relevant data from s_a and each $pt_a \in PT_a$ using SPARQL [27] queries. Then, we transform the information retrieved by the SPARQL queries into a textual description as input for the next sub-layer. In the second sub-layer, we process the textual description resulting from last layer. We apply text processing techniques [22], such as removing punctuation, lowercasing, tokenization, stop word removal, and lemmatization. In the third sub-layer, we receive the processed text and calculate the cosine similarity between the angle of the s_a and pt_a vectors [19], having as a result a text-based similarity value between 0 to 1. Then, we provide the possible traces between the selected user story part s_a and all possible EDG model elements $pt_a \in PT_a$ with the text-based similarity value. Finally, OntoTraceV2.0 recommend tracing s_a to a $pt_a \in PT_a$ if the calculated text-based similarity is higher or equal to a recommendation threshold. We show a detailed example on how this text go through all three NLP sub-layers in Fig. 6.

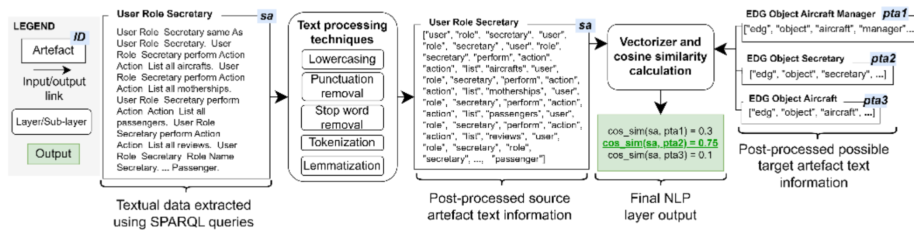

Fig. 6. Detailed NLP layer example.

So far, we briefly discussed the technical details of each NLP sub-layer. In this paper, we specially focus on the third sub-layer's vectorizing techniques [20, 21] and how such techniques affect the OntoTraceV2.0 recommendation accuracy. To do so, we test four vectorizing techniques: Count Vectorizer, TFIDF Vectorizer, Doc2Vec, and Universal Sentence Encoder. Count and TFIDF Vectorizers are mathematical-based techniques for vectorizing text, using word frequency to create a vector representation [21]. The Doc2Vec and the Universal Sentence Encoder are machine-learning-based techniques for vectorizing text using word embeddings [20, 21]. To compare them, we gather and process the text from each s_a with their PT_a set using the first two NLP sub-layers. We calculate the cosine similarity between vector representations for each vectorizing technique—i.e., we calculate cos_sim (s_a, pt_a) for each $pt_a \in PT_a$. Finally, we calculate the *recommendation accuracy* to compare the vectorizing techniques. We define *recommendation accuracy* based on [28] as:

$$Recommendation\ accuracy = \frac{Number\ of\ successful\ recommentations}{Total\ number\ of\ recommendations} * 100\% \quad (1)$$

We calculate the average recommendation accuracy to compare the four vectorizing techniques and report the results in Table 3.

Table 3. Vectorizing techniques and their recommendation accuracy.

Vectorizing technique	Recommendation threshold	Recommendation Accuracy (AVG)
Count vectorizer	0.9025	71.80%
TFIDF vectorizer	0.9139	75.89%
Doc2Vec	0.9965	20.21%
Universal Sentence Encoder	0.9625	60.37%

We observe that the TFIDF vectorizer has the highest recommendation accuracy in average among vectorizing techniques. Therefore, we select the TFIDF vectorizer to implement the third sub-layer vectorizing technique. For the sake of space, we include a detailed description on how we designed the SPARQL queries, implemented the text processing techniques, and selected the vectorizing techniques and similarity calculation method in an annexe repository[3]. However, evaluating the effect on recommendation accuracy with different similarity calculation formulas, text processing techniques, SPARQL queries, and other NLP techniques—e.g., using transformer models—is still work in progress.

5 Evaluating OntoTraceV2.0

We have conducted a quasi-experiment to measure the extent OntoTraceV2.0 affects trace generation effectiveness, efficiency, and satisfaction. We design and execute this quasi-experiment based on Wohlin et al. [29] and Moody's [30] Technology Acceptance Model (TAM), addressing RQ2 presented in Section 3. Our quasi-experiment is fixed to User Stories and EDG models. However, we consider answering RQ2 as the

[3] https://doi.org/10.5281/zenodo.7589791

first step for future experiment replications with other artefacts and as a first step to find general conclusions.

5.1 Experimental design

The experimental goal according to the Goal/Question/Metric template [31] is to **analyse the use of** *OntoTraceV2.0* **for the purpose of** *trace generation between user stories and EDG models* **with respect to** *effectiveness, efficiency, and satisfaction* **from the point of view of** *engineering bachelor students* in **the context of** *a bachelor course on rapid software prototyping (RASOP) at the ZHAW in Switzerland.*

Experimental Subjects. The quasi-experiment was conducted with 36 subjects, all of them engineering students enrolled in the RASOP course. The subjects are part of diverse engineering programs such as: Information technology (IT; 36.1%), Computer Sciences (30.6%), Industrial engineering (8.3%), Systems engineering (5.6%), Mechanical engineering (5.6%), Business engineering (5.6%), Aviation (2.8%), Electrical engineering (2.8%), and Energy and Environmental Engineering (2.8%). More than a half (58.3%) of the subjects have between 0.5 to 10 years of industry experience (2.8 years average $\pm$ 3.1 years std) in field such as software engineering, data mining, mechanics, and semi-conductor industry, among others. However, only one subject (2.8%) has one year of previous experience on software traceability. The other subjects (97.2%) have no previous experience on software traceability. Subjects were informed about data collection, and they executed the experimental tasks as part of the course graded activities. Nevertheless, we inform them that there are no direct benefits in the grades to let us collect their data.

Variables. We consider *one independent variable*: generating traces with and without OntoTraceV2.0. On the other hand, we consider *three independent variables* grouped by effectiveness, efficiency, and satisfaction based on Moody's evaluation model [30]. For effectiveness, we decide to measure subject's precision during trace generation. For efficiency, we plan to measure subject's number of generated traces per minute. For satisfaction, we propose to measure three qualitative variables based on a 1-to-5 Likert scale: Perceived ease of use (PEU), perceived usefulness (PU), and Intention to Use (ITU).

Hypotheses. We define null hypotheses (represented by a 0 in the subscript) stating that OntoTraceV2.0 do not affect the trace generation effectiveness, efficiency, and satisfaction. The alternative hypotheses (represented by a 1 in the subscript) suppose there is an influence. We show our hypotheses in Table 4, alternative hypotheses are omitted.

Table 4. Null hypothesis (H_0) description.

H_0	**Statement:** The use of OntoTraceV2.0 does not affect the subject's ...
$H1_0$	…**effectiveness** when generating traces between user stories and EDG models.
$H2_0$	…**efficiency** when generating traces between user stories and EDG models.
$H3_0$	… **PEU** when generating traces between user stories and EDG models.
$H4_0$	… **PU** when generating traces between user stories and EDG models.
$H5_0$	… **ITU** when generating traces between user stories and EDG models.

PEU: Perceived Ease of Use; **PU**: Perceived Usefulness; **ITU**: Intention to Use

5.2 Procedure and data analysis[4]

We conducted the quasi-experiment following a blocked subject-object study having one factor with two treatments experimental design [29]. Hence, we propose randomly dividing subjects into two balanced groups: GR1 and GR2. Both groups received training on traceability during the RASOP lectures. We design two experimental objects (O1 and O2) that both GR1 and GR2 will face in two different sessions. Subjects receive a set of user stories as source artefacts and an EDG as target artefact, having as a task generating the traces between artefacts. Moreover, source and target artefacts are previously labelled (see Fig. 7).

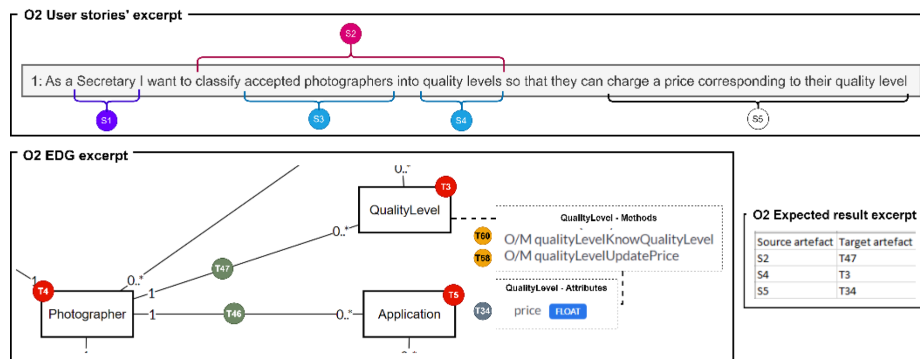

Fig. 7. O2 experimental object excerpt. **S#**: source artefact; **T#**: target artefact.

Before each session, we introduce the experimental task using an experimental training object (O0). Data from O0 is not collected nor evaluated since it is just for training subjects. During the first session, GR1 works on O1 and GR2 works on O2, both groups without using OntoTraceV2.0—i.e., using a manual traceability strategy. During the second session, GR1 works on O2 and GR2 works on O1, but now using OntoTraceV2.0. We decide to evaluate OntoTraceV2.0 against manual traceability rather than OntoTraceV1.0 since OntoTraceV2.0 contains all features from OntoTraceV1.0, allowing us to assess not only the new NLP feature but also the ontology-based automatic reasoning feature. At the end of each session, we ask subjects to provide us with the ending time, the generated traces, and a satisfaction questionnaire.

Using the previously discussed configuration, quasi-experiment findings are not entirely dependent on the experimental object since we use two experimental objects. Moreover, we avoid the between-session experimental object learning effect since subjects work on different experimental objects in each session. Furthermore, session 1 and session 2 were performed with a one-week time difference, decreasing the effect on satisfaction variables by the time between sessions. However, we could not prevent a between-task learning effect—i.e., even if we did not reveal the correct results between sessions, subjects learn how to perform the traceability task from session 1 and

[4] To facilitate further replications, all material related to the experimental objects, demographics, and results can be found at https://doi.org/10.5281/zenodo.7360221.

use that knowledge in session 2—due to time and infrastructure limitations. As a disclaimer, such a learning effect can affect effectiveness and efficiency metrics, requiring further replications to validate our results. We deeply discuss this and other threats to validity in more detail in Section 5.3. Finally, all subjects are used in both sessions, avoiding variability among subjects.

Data analysis. We analyse the descriptive statistics, comparing means of dependent variables (see Fig. 8). Moreover, we run a generalised linear model to test the hypothesis (see Table 5).

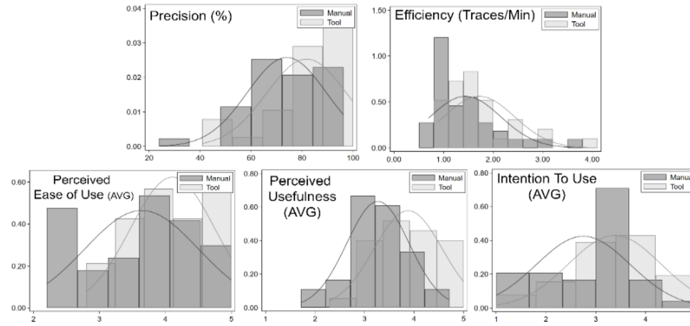

Fig. 8. Quasi-experiment results' distributions, having y-axis as the probability density.

Table 5. Statistical generalized linear model test results.

Independent variables	Dependent variables				
	Precision	Efficiency (Traces/min)	PEU (AVG)	PU (AVG)	ITU (AVG)
OntoTraceV2.0	0.0165 (0.0539)	0.670*** (0.230)	0.489* (0.267)	0.649*** (0.231)	0.600* (0.331)
Experimental Object	-0.099* (0.0513)	0.807*** (0.219)	0.256 (0.255)	0.246 (0.221)	-0.167 (0.316)
OntoTraceV2.0 & Experimental Object	0.121 (0.0749)	-0.783** (0.319)	-0.0438 (0.372)	-0.105 (0.322)	0.0569 (0.461)

Standard errors in parentheses; ***: $p < 0.01$; **: $p < 0.05$, *: $p < 0.1$

Effectiveness. We observe subject's effectiveness in terms of precision without OntoTraceV2.0 is in average 73.99% ± 15.77% compared to 81.74% ± 15.49% with OntoTraceV2.0. This means that subjects identify 7.75% more correct traces with OntoTraceV2.0 compared to a manual strategy on average. However, we observe that OntoTraceV2.0 has no statistical representative effect into subject's precision. Similarly with the interaction between OntoTraceV2.0 and the experimental object. Therefore, we cannot reject $H1_0$. On the other hand, we observe the experimental object has a negative representative effect into subject's precision. This could indicate that one experimental object is more challenging that the other.

Efficiency. We observe subject's efficiency in terms of trace/min without OntoTraceV2.0 is in average 1.44 traces/min ± 0.71 traces/min compared to 1.72 traces/min ± 0.71 traces/min with OntoTraceV2.0. This means that subjects create 0.28 traces/min

(16.8 traces/hour) faster compared to a manual strategy on average. Moreover, we observe that OntoTraceV2.0 has a positive statistical representative effect into subject's efficiency. Similarly with interaction between OntoTraceV2.0 and the experimental object. Therefore, we reject $H2_0$ with a 99% of confidence. As a disclaimer, this result could be due to between-task learning validity threat—i.e., a maturity threat—as we previously mentioned. Thus, a double check with future replicas is needed. On the other hand, we observe the experimental object has a positive representative effect into subject's efficiency. This seems to confirm what we identified with effectiveness, where one experimental object seems to not require as much effort as the other.

Satisfaction. We observe that subject's satisfaction in terms of PEU, PU, and ITU without OntoTraceV2.0 is in average 3.64 $\pm$ 0.86, 3.27 $\pm$ 0.63, 2.75 $\pm$ 0.94 respectively. Furthermore, we observe that subject's satisfactions in terms of PEU, PU, and ITU with OntoTraceV2.0 is in average 4.11 $\pm$ 0.64, 3.88 $\pm$ 0.69, 3.38 $\pm$ 0.93 respectively. This means that subjects perceived a better satisfaction in terms of PEU, PU, and ITU using OntoTraceV2.0 on average—specifically, OntoTraceV2.0 increase PEU, PU, and ITU on average 0.47, 0.61, and 0.63 points respectively. Moreover, we observe that OntoTraceV2.0 has a positive statistical representative effect into PEU, PU, and ITU. Therefore, we reject $H3_0$, $H4_0$, and $H5_0$ with a 90%, 99%, and 90% of confidence respectively. On the other hand, we observe that the experimental object nor the interaction between experimental object and OntoTraceV2.0 has statistical representative effect into subject's satisfaction in terms of PEU, PU and ITU.

5.3 Threats to Validity

Internal Validity. GR1 and GR2 group subjects could share information about their experimental objects, materializing a *diffusion* threat. Due to that, we prepared two versions of our experimental objects O1.1, O2.1 and O1.2, O2.2. Thus, we minimize the effect of *diffusion* about experimental objects between sessions since subjects always face new experimental objects. In terms of *maturity*, subjects were able to improve their tracing skills between sessions affecting their efficiency and effectiveness. To minimise this threat, we do not reveal the results of their performance until the end of the second session, avoiding subjects learn from the first session results. However, subjects still could learn how to generate traces between session 1 and session 2—e.g., subjects could learn how to trace more efficiently even if they do it in a wrong way because they do not know the correct result. In our quasi-experiment, we did not assess this *maturity* threat that could affect especially effectiveness and efficiency. We plan to verify such results in further experiment replications.

External Validity. We involved students from different engineering bachelor programs as experimental subjects. This could represent an *interaction of selection and treatment* threat where subjects are not representative of the population we want to generalize. However, all students participating in RASOP lecture are interested on software development and software quality assurance tasks such as traceability based on RASOP syllabus. Thus, we minimize considering RASOP students as potential population to use OntoTraceV2.0. Nevertheless, we acknowledge the limits in the *generalization of the experiment results* since we did not include other subjects of interest such

as traceability experts. We plan to replicate this quasi-experiment to generalize our results, including traceability experts and software development teams working in industry.
Construct validity. Subjects could be afraid of being evaluated affecting their results, materializing an *evaluation apprehension* threat. We minimize this threat letting subjects know that all data is anonymous, and no benefit/penalization is made for letting us collect their data.
Conclusion validity. Although we conducted a quasi-experiment with 36 subjects, the sample size is still small. This represents a *low statistical power* threat. To mitigate this threat in the future, we plan to replicate this quasi-experiment increasing the sample size. Moreover, there are external *experimental setting* threats we could not mitigate that can affect the experiment results. For instance, RASOP lecture is scheduled from 17:45 to 21:00. During the evening subjects are tired and that can affect their results.

6 Conclusions and future work

In this paper, we propose OntoTraceV2.0: an ontology-based automatic reasoning and NLP-based tool for generating traces between software artefacts. OntoTraceV2.0 is built on top of previous work by including an NLP layer for supporting decision-making on generating traces between artefacts—i.e., for recommending traces between artefacts. Then, OntoTraceV2.0 users can use an automatic reasoner together with NLP to infer traceability-related information such as: i) which artefacts are not yet traced; ii) which are the traceable source/target artefacts; and iii) given a specific artefact, which are the possible recommended traces between it and other artefacts based on text-based similarity.

We conducted a quasi-experiment with 36 subjects to analyse OntoTraceV2.0 effect on effectiveness, efficiency, and satisfaction on trace generation. We observed OntoTraceV2.0 positively affects the subjects' efficiency and satisfaction during trace generation compared to a manual approach. However, although the subjects' average effectiveness is higher using OntoTraceV2.0, we observed no statistical difference with the manual trace generation approach in terms of effectiveness. The lack of significant effect in terms of effectiveness is a limitation. This indicates we still need to improve OntoTraceV2.0 trace recommendation techniques. In the future, we will improve trace recommendations by devising new techniques, combining NLP and machine learning algorithms. In addition, we identified some threats to validity that can affect our results, especially in terms of effectiveness and efficiency. We plan to replicate this quasi-experiment having in mind threats to validity such as *maturity, low statistical power,* and *generalisation of experimental results* to validate our results.

Acknowledgments: This research is fully funded by the ZHAW Institute for Applied Information Technology (InIT), the Innosuisse Flagship SHIFT project, and the ZHAW School of Engineering. Moreover, we would like to thank all RASOP course students for actively participating on the quasi-experiment, allowing us to gather all the data we used to build our research.

References

1. Charalampidou, S., Ampatzoglou, A., Karountzos, E., Avgeriou, P.: Empirical studies on software traceability: A mapping study. Journal of Software: Evolution and Process. 33, (2021).
2. Cleland-Huang, J., Gotel, O., Zisman, A.: Software and Systems Traceability. Springer, London (2012).
3. Antoniol, G., Canfora, G., de Lucia, A.: Maintaining traceability during object-oriented software evolution: a case study. In: IEEE International Conference on Software Maintenance - 1999 (ICSM'99). pp. 211–219. (1999).
4. Sundaram, S.K., Hayes, J.H., Dekhtyar, A., Holbrook, E.A.: Assessing traceability of software engineering artifacts. Requir Eng. 15, 313–335 (2010).
5. Lin, J., Liu, Y., Zeng, Q., Jiang, M., Cleland-Huang, J.: Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 324–335. IEEE (2021).
6. Maro, S., Steghofer, J.-P.: Capra: A Configurable and Extendable Traceability Management Tool. In: 2016 IEEE 24th International Requirements Engineering Conference (RE). pp. 407–408. IEEE (2016).
7. Nagano, S., Ichikawa, Y., Kobayashi, T.: Recovering Traceability Links between Code and Documentation for Enterprise Project Artifacts. In: 2012 IEEE 36th Annual Computer Software and Applications Conference. pp. 11–18. IEEE (2012).
8. Guo, J., Cleland-Huang, J., Berenbach, B.: Foundations for an expert system in domain-specific traceability. In: 2013 21st IEEE International Requirements Engineering Conference (RE). pp. 42–51. IEEE (2013).
9. Javed, M.A., UL Muram, F., Zdun, U.: On-Demand Automated Traceability Maintenance and Evolution. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 111–120. Springer Verlag (2018).
10. Narayan, N., Bruegge, B., Delater, A., Paech, B.: Enhanced traceability in model-based CASE tools using ontologies and information retrieval. In: 2011 4th International Workshop on Managing Requirements Knowledge. pp. 24–28. IEEE (2011).
11. Javed, M.A., Stevanetic, S., Zdun, U.: Towards a pattern language for construction and maintenance of software architecture traceability links. In: Proceedings of the 21st European Conference on Pattern Languages of Programs. pp. 1–20. ACM, New York, NY, USA (2016).
12. Huaqiang, D., Hongxing, L., Songyu, X., Yuqing, F.: The Research of Domain Ontology Recommendation Method with Its Applications in Requirement Traceability. In: 2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES). pp. 158–161. IEEE (2017).
13. Hayashi, S., Yoshikawa, T., Saeki, M.: Sentence-to-Code Traceability Recovery with Domain Ontologies. In: 2010 Asia Pacific Software Engineering Conference. pp. 385–394. IEEE (2010).
14. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically Enhanced Software Traceability Using Deep Learning Techniques. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 3–14. IEEE (2017).
15. Mosquera, D., Ruiz, M., Pastor, O., Spielberger, J., Fievet, L.: OntoTrace: A Tool for Supporting Trace Generation in Software Development by Using Ontology-Based Automatic Reasoning. In: CAiSE'22. pp. 73–81. Springer (2022).
16. Snoeck, M.: Enterprise Information Systems Engineering. Springer International Publishing, Cham (2014).

17. Guo, J., Monaikul, N., Cleland-Huang, J.: Trace links explained: An automated approach for generating rationales. In: 2015 IEEE 23rd International Requirements Engineering Conference (RE). pp. 202–207. IEEE (2015).
18. Thamrongchote, C., Vatanawood, W.: Business process ontology for defining user story. In: 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS). pp. 1–4. IEEE (2016).
19. Li, B., Han, L.: Distance Weighted Cosine Similarity Measure for Text Classification. In: 2013 International Conference on Intelligent Data Engineering and Automation Learning. pp. 611–618. Springer (2013).
20. Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. st., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B., Kurzweil, R.: Universal Sentence Encoder. (2018).
21. Singh, L.: Clustering Text: A Comparison Between Available Text Vectorization Techniques. In: 3rd ICSCSP - Soft Computing and Signal Processing. pp. 21–27. Springer (2022).
22. Hickman, L., Thapa, S., Tay, L., Cao, M., Srinivasan, P.: Text Preprocessing for Text Mining in Organizational Research: Review and Recommendations. Organ Res Methods. 25, 114–146 (2022).
23. Noy, N.F., McFuiness, D. la: Ontology Development 101: A Guide to Creating Your First Ontology, https://protege.stanford.edu/publications/ontology_development/ontology101.pdf, last accessed 2021/11/29.
24. Bragilovski, M., Dalpiaz, F., Sturm, A.: Guided Derivation of Conceptual Models from User Stories: A Controlled Experiment. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 131–147. Springer Science and Business Media Deutschland GmbH (2022).
25. Nasiri, S., Rhazali, Y., Lahmer, M., Chenfour, N.: Towards a Generation of Class Diagram from User Stories in Agile Methods. Procedia Comput Sci. 170, 831–837 (2020).
26. Web Ontology Language (OWL), https://www.w3.org/OWL/, last accessed 2021/11/29.
27. SPARQL query language, https://www.w3.org/2001/sw/wiki/SPARQL, last accessed 2021/11/29.
28. Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A., Kashef, R.: Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities. Applied Sciences. 10, 7748 (2020).
29. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2012).
30. Moody, D.L.: The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods. In: ECIS 2003 Proceedings. pp. 79–96 (2003).
31. van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal Question Metric (GQM) Approach. In: Encyclopedia of Software Engineering. John Wiley & Sons, Inc., Hoboken, NJ, USA (2002).