

CiRA: A Tool for the Automatic Detection of Causal Relationships in Requirements Artifacts

Jannik Fischbach^a, Julian Frattini^b and Andreas Vogelsang^c

^aQualicen GmbH, Germany

^bBlekinge Institute of Technology, Sweden

^cUniversity of Cologne, Germany

Abstract

Requirements often specify the expected system behavior by using causal relations (e.g., If A, then B). Automatically extracting these relations supports, among others, two prominent RE use cases: automatic test case derivation and dependency detection between requirements. However, existing tools fail to extract causality from natural language with reasonable performance. In this paper, we present our tool CiRA (Causality detection in Requirements Artifacts), which represents a first step towards automatic causality extraction from requirements. We evaluate CiRA on a publicly available data set of 61 acceptance criteria (causal: 32; non-causal: 29) describing the functionality of the German Corona-Warn-App. We achieve a macro F_1 score of 83 %, which corroborates the feasibility of our approach.

Keywords

Causality, Requirements Engineering, Tool Demo, Natural Language Processing

1. Introduction


Conditional clauses are prevalent for the specification of system behavior, e.g., “If the user enters an incorrect password, an error message shall be displayed” (REQ 1). Such conditionals are a widely used linguistic pattern in both traditional requirements documents [1] as well as agile requirement artifacts, such as acceptance criteria [2]. Semantically, conditional clauses can be understood as a causal relation between the antecedent (in case of REQ 1: *incorrect password is entered*) and the consequent (*error message is displayed*). Understanding and extracting such causal relations offers great potential for Requirements Engineering (RE) as it supports among others the following use cases [3]:


Use Case 1: Automatic Test Case Derivation From Requirements To derive test cases from natural language (NL) requirements, the specified system behavior and the combinatorics behind the requirement need to be understood. Specifically, we need to understand the embedded causal relation to determine the correct combination of test cases that cover all positive and negative scenarios. Currently, practitioners have to extract the causal relation manually and

In: F.B. Aydemir, C. Gralha, S. Abualhaija, T. Breaux, M. Daneva, N. Ernst, A. Ferrari, X. Franch, S. Ghanavati, E. Groen, R. Guizzardi, J. Guo, A. Herrmann, J. Horkoff, P. Mennig, E. Paja, A. Perini, N. Seyff, A. Susi, A. Vogelsang (eds.): *Joint Proceedings of REFSQ-2021 Workshops, OpenRE, Posters and Tools Track, and Doctoral Symposium, Essen, Germany, 12-04-2021*

✉ jannik.fischbach@qualicen.de (J. Fischbach); julian.frattini@bth.se (J. Frattini); vogelsang@cs.uni-koeln.de (A. Vogelsang)

ORCID 0000-0002-4361-6118 (J. Fischbach); 0000-0003-3995-6125 (J. Frattini); 0000-0003-1041-0815 (A. Vogelsang)

 © 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

determine the combinations of causes and effects that need to be covered by test cases [4]. This is not only cumbersome but also becomes increasingly error-prone with growing requirements complexity [5]. We argue that causality extraction combined with existing automatic test case derivation methods contributes to the alignment of RE and testing (e.g., by mapping the causal relation to a Cause-Effect-Graph from which test cases can be derived automatically [2]).

Use Case 2: Automatic Dependency Detection Between Requirements As modern systems are becoming more and more complex, the number of requirements and their relations is constantly increasing. Practitioners fail in keeping an overview of the relationships between the requirements [6]. This may lead to undetected redundancies and inconsistencies within the requirements and consequently to faults in the system design [7]. We argue that an automatic causality extraction from requirements can help to compare the semantics by analyzing the different embedded causal relations. As a result, relations between requirements can be identified automatically (e.g., contradictory and redundant requirements).

Existing approaches [8] fail to extract causality from NL with a performance that allows for use in practice. Therefore, we argue for the need of a novel method for the extraction of causality from requirements [9]. We understand causality extraction as a two-step problem: We first need to detect whether requirements contain causal relations. Second, if they contain causal relations, we need to locate and extract them. In this paper, we present a demo of our tool CiRA (Causality detection in Requirements Artifacts), which forms a first step towards causality extraction from NL requirements [1]. In the remainder of this paper we provide an overview¹ of the functionality of CiRA (Section 2) and outline how we plan to conduct the demo at the workshop (Section 3).

2. The CiRA Approach

CiRA has been trained to solve causality detection as a binary classification problem. Specifically, CiRA is capable of classifying single sentences or even multiple concatenated sentences written in unrestricted natural language into two categories: 1) the input contains a causal relation or 2) the input does not contain a causal relation. The classification is performed in four steps:

- Ⓐ **Tokenization of the Text Input** First, the text input must be decomposed into individual tokens. Since we use the Bidirectional Encoder Representations from Transformers (BERT) model [10] as the foundation for CiRA, the input must be brought to a fixed length (maximum 512 tokens). For sentences that are shorter than this fixed length, padding tokens (PAD) are inserted to adjust all sentences to the same length. In addition to the PAD token, SEP tokens are inserted as special separator tokens and the CLS (classification) token is inserted as the first token in the input sequence. The CLS token represents the whole sentence (i.e., it is the pooled output of all tokens of a sentence). Our experiments [1] revealed that CiRA performs best with a fixed length of 384 tokens.
- Ⓑ **Enriching the Text Input with Syntactic Information** In this step, we provide knowledge about the grammatical structure of the sentence to the classifier. For this purpose, we add the corresponding Dependency (DEP) tag to each token by using the spaCy NLP library. Our experiments [1] demonstrated that adding the DEP tags leads to a performance gain

¹In this paper, we only provide a high level overview of the architecture of CiRA. For a detailed description of the training and tuning of our approach, please refer to our paper at the REFSQ Research Track [1]

Acceptance Criterion	Label
The prompt no longer appears after the first time the app is used.	1
The terms of use can be displayed within the app.	0
The consent prompt is shown only the first time a user launches the app.	1
An explanation of the app’s various functions will be provided.	0
There is a “Publication information” item in the menu.	0
The IDs can be sent to the Warn server pseudonymized.	0

	Precision	Recall	F1
Causal (Support: 32)	0.92	0.75	0.83
Non-causal (Support: 29)	0.77	0.93	0.84
Accuracy (Support: 61)		0.84	

Table 1: Data Set (above) and Evaluation Results (below)

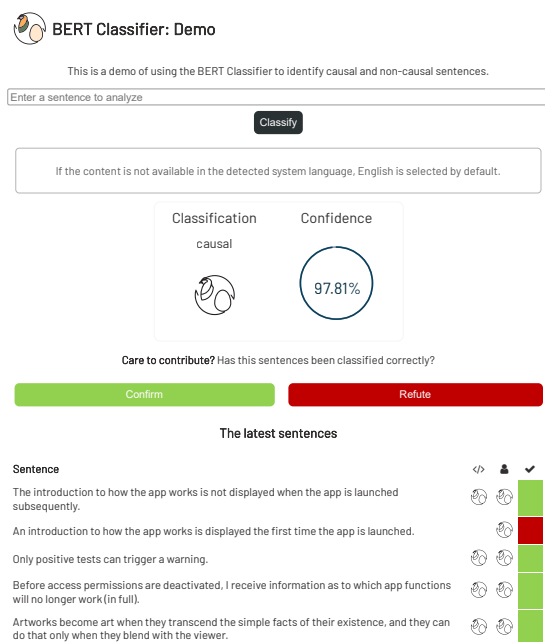


Figure 1: UI of CiRA

compared to adding Part-of-Speech (POS) tags to the text input or the usage of the vanilla BERT model.

- ③ **Generate Sentence Embedding** After the pre-processing, the tokens are fed into the BERT model, which generates the corresponding embeddings. For our classification tasks, we are mainly interested in the CLS token and its embedding. Since the CLS token represents the whole sentence, the embedding created by BERT represents a sentence embedding that can be easily used for classification.
- ④ **Softmax Classification** Finally, the sentence embedding is fed into a single-layer feed forward neural network that uses a softmax layer, which calculates the probability that a sentence is causal or not.

3. Demo Plan

Technical Setup & User Interface During the workshop we will use our online demo of CiRA (www.cira.bth.se/bert). The website is built as a restful node.js server utilizing the Express framework. The backend’s main purpose is to execute a Python script, which acts as a wrapper around the classifier: our pre-trained binary-file classifier is loaded, the sentence classified, and the resulting classification alongside the classifier’s confidence returned. The UI provides a text input field, where an arbitrary NL sentence can be entered (see top of Fig. 1). On pressing the “classify”-button, the sentence is sent to the backend where it is processed by the aforementioned, wrapped classifier. On return of the REST call, the classification and confidence of the model

are rendered in the UI. The user may confirm or correct the classifiers choice. The entered sentence and the optional user confirmation or correction is then stored in the backend, in order to (1) display the five most recently entered sentences (see bottom of Fig. 1), (2) provide preliminary insight into the performance of the classifier on unseen sentences, and (3) preserve sentences for future training of the classifier. Currently, we only support batch learning, but we plan to implement an online learning algorithm in future research to leverage the collected data directly for enhancing CiRA.

Evaluation On Unseen Real World Data In the NLP4RE-workshop, we will demonstrate that CiRA is suitable for practical use. For this purpose, we evaluate CiRA on unseen real word data to simulate its application in the intended context. We use a publicly available data set of acceptance criteria² provided by SAP, which describe the functionality of the German “Corona-Warn-App”. The data set consists of 32 user stories containing a total of 61 acceptance criteria. In order to measure the performance of CiRA, we manually annotated all acceptance criteria and classified them into two categories: causal (Label: 1) and non-causal (Label: 0). We then classified each acceptance criterion using CiRA and compared the results with ground truth (see Tab. 1). During the workshop, we will present both true and false predictions and discuss the performance, challenges, and possibilities of CiRA with the other participants.

References

- [1] J. Fischbach, J. Frattini, A. Spaans, M. Kummeth, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, Automatic detection of causality in requirement artifacts: the cira approach, in: REFSQ, 2021.
- [2] J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, D. Freudenstein, Specmate: Automated creation of test cases from acceptance criteria, in: ICST, 2020.
- [3] J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, A. Vogelsang, Towards causality extraction from requirements, in: RE, 2020.
- [4] V. Garousi, S. Bauer, M. Felderer, NLP-assisted software testing: a systematic review, CoRR (2018).
- [5] J. Fischbach, H. Femmer, D. Mendez, D. Fucci, A. Vogelsang, What makes agile test artifacts useful? an activity-based quality model from a practitioners’ perspective, in: ESEM, 2020.
- [6] F. Dalpiaz, A. Ferrari, X. Franch, C. Palomares, Natural language processing for requirements engineering: The best is yet to come, IEEE Software 35 (2018).
- [7] A. Vogelsang, Feature dependencies in automotive software systems: Extent, awareness, and refactoring, Journal of Systems and Software 160 (2020).
- [8] N. Asghar, Automatic extraction of causal relations from natural language texts: A comprehensive survey, ArXiv abs/1605.07895 (2016).
- [9] J. Frattini, M. Junker, M. Unterkalmsteiner, D. Mendez, Automatic extraction of cause-effect-relations from requirements artifacts, in: ASE, 2020.
- [10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.

²The data set can be found at https://github.com/corona-warn-app/cwa-documentation/blob/master/scoping_document.md.