CiRA: An Open-Source Python Package for Automated Generation of Test Case Descriptions from Natural Language Requirements

1st Julian Frattini
3rd Andreas Bauer
Blekinge Institute of Technology
Karlskrona, Sweden
{firstname}.{lastname}@bth.se

2nd Jannik Fischbach
Netlight Consulting GmbH and fortiss GmbH
Munich, Germany
jannik.fischbach@netlight.com

***Abstract*—Deriving acceptance tests from high-level, natural language requirements that achieve full coverage is a major manual challenge at the interface between requirements engineering and testing. Conditional requirements (e.g., "If A or B then C.") imply causal relationships which—when extracted—allow to generate these acceptance tests automatically. This paper presents a tool from the CiRA (Causality In Requirements Artifacts) initiative, which automatically processes conditional natural language requirements and generates a minimal set of test case descriptions achieving full coverage. We evaluate the tool on a publicly available data set of 61 requirements from the requirements specification of the German Corona-Warn-App. The tool infers the correct test variables in 84.5% and correct variable configurations in 92.3% of all cases, which corroborates the feasibility of our approach.**

***Index Terms*—requirements engineering, natural language processing, acceptance test, test case description, BERT**

I. Introduction

A key challenge of high-level acceptance testing—where the behavior of a system is compared to the behavior specified by the requirements—is generating a set of test cases that fully cover a requirement [1]. A requirement is fully covered when the test cases associated with it assert all configurations of input and output variables implied by the requirement.

Conditional requirements (e.g., "**When** the red button is pushed **or** the power fails **then** the system shuts down.") are often used to specify functional requirements [2] and make the causal relationship between input and expected output explicit [3]. This causal relationship can be used to construct a minimal set of test cases fully covering the requirement [4].

Since these conditional requirements can be detected with a reasonable accuracy [5], approaches to automatically extract the causal relationship from the natural language (NL) requirement have been developed [6]. We present a tool to automatically generate a minimal set of test case descriptions from an NL requirement, ensuring its full coverage. We disclose all source code and demonstrate the tool's applicability by generating test case descriptions for the Corona-Warn-App.

This work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

II. Tool Introduction

The goal of the CiRA tool is to generate a minimal but fully covering set of test case descriptions from a single-sentence, natural language requirement specification implying a causal relationship. To achieve this goal, the tool needs to (1) identify whether the sentence contains a causal relationship, (2) assign each word in the sentence to its role in the causal relationship, (3) transform the NL sentence into a cause-effect graph (CEG), and finally (4) derive a minimal set of test case descriptions from the CEG. We illustrate these four steps of the tool's pipeline with the example sentence from the introduction.

Step 1: Identifying causal relationships. First, the CiRA tool performs a binary classification to categorize the NL requirements specification as either `causal` or `non-causal` [5]. Only causal sentences can be processed further. In this example, the *CiRA classifier* module categorizes the sentence as `causal` with 98% confidence.

Step 2: Assigning roles to words. Secondly, the CiRA tool performs a multi-labeling task to assign to each word of the sentence its respective role in the causal relationship. There are two levels of labels: (1) *event labels*, which determine to which event (cause or effect) a word belongs, or *junctors*, which determine how events are related, and (2) *sub-labels*, which determine the part of an event (variable or condition) a word belongs to. Figure 1 visualizes how the *CiRA Labeler* annotates the example sentence: it identifies three events, and the two cause-events are related with a disjunction (labeled ∨). In each event, its variable is distinguished from its condition.

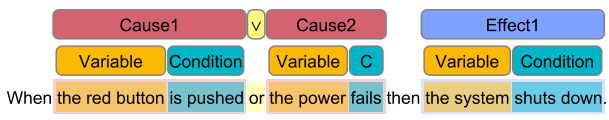


Fig. 1. Labeled requirements specification

Step 3: Constructing a graph. Thirdly, the CiRA tool constructs a Cause-Effect-Graph (CEG) from the labeled sentence. The *CiRA Graph Constructor* generates one node for each event and connects events with directed edges according to

their relationship. Figure 2 shows the translation of the labeled sentence into a CEG.

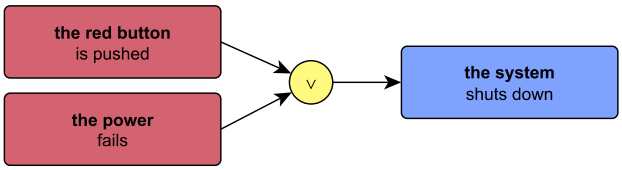


Fig. 2. Generated cause-effect graph

Step 4: Deriving test cases. Finally, the CiRA tool derives a minimal set of test case descriptions from the CEG, which fully covers the initial requirements specification. For this, the *CiRA Test Case Generator* considers all events to be Boolean (i.e., they can be evaluated to true or false) and determines all configurations of the cause-events necessary to evaluate the effect-events to both true and false. Table I shows the test suite generated for the example sentence. To evaluate the effect-event to be false, both cause-events need to be false as well (TC3). To evaluate the effect-event to be true, either one of the cause-events must be true (TC1 and TC2) given their connection via a disjunction (i.e., logical or). A fourth test case where both cause-events are true is unnecessary, as TC1 and TC2 assert that either of the two events can cause the effect-event to be evaluated as true.

TABLE I
GENERATED TEST SUITE OF TEST CASE DESCRIPTIONS

ID	the red button	the power	the system
TC1	is pushed	*not* fails	shuts down
TC2	**not** is pushed	fails	shuts down
TC3	**not** is pushed	**not** fails	**not** shuts down

In previous work [5], [6], we developed a prototype of this tool for evaluation purposes. This paper presents a maintainable, usable, and open-access evolution of that prototype. The CiRA tool—visualized in Figure 3—consists of the CiRA core[1], which offers the functionality of the described pipeline. The classification and the labeling task utilize pre-trained, fine-tuned machine learning models based on BERT and RoBERTa [2], [6]. A REST API provides an interface to the functionality of the CiRA tool, which is used by a dedicated graphical user interface (GUI)[2] for human interaction[3]. Figure 4 shows the full GUI with another, more complex example.

III. DEMONSTRATION

To demonstrate the usability and performance of the CiRA tool, we evaluate its capability on an unseen set of natural language requirements[4]. Similar to the previous version of the tool [7], we chose the requirements of the German Corona Warn App[5] as a target system due to its recency, openness, and realism. The system contains ten epics, 32 user stories connected to these epics, and in total, 61 acceptance criteria.

The subject of the evaluation is the 72 natural language sentences constituting the acceptance criteria. We manually classified each sentence as either *causal* (N=26) or *non-causal* (N=46) and created an expected set of test cases for each of the causal sentences. Then, we executed both the CiRA classifier and the CiRA Test Suite Generator and compared the automatically generated results with the manual ones. We evaluate the classifier using the macro F1-score [7] and the test suite generator regarding its ability to infer correct events from the causal sentence and its ability to determine the correct test value configurations.

The CiRA Classifier achieves a macro f1-score of 79.26%. The CiRA test suite generator infers the correct test variables in 84.5% and the correct test value configurations in 92.3% of all cases. The results show that CiRA can automate a large part of manual test case generation but struggles with implicitly causal or ill-structured sentences.

While the CiRA tool was mainly designed for test case generation, the intermediate pipeline steps allow using it in further use cases. Future research could include completeness assessment (i.e., assessing the output of the labeling to evaluate whether all components of a causal relationship (variables and conditions) are contained in the sentence) and dependency detection (i.e., comparing the CEGs of multiple requirements to identify dependencies). Finally, connecting the pipeline to another tool that converts the test case descriptions into executable test cases will fully automate the acceptance testing process.

[1]Source code at https://zenodo.org/badge/latestdoi/456568427

[2]Source code at https://zenodo.org/badge/latestdoi/571614601

[3]Online demo at www.cira.bth.se/demo

[4]Data and code at https://zenodo.org/badge/latestdoi/650042294

[5]Requirements specification at https://github.com/corona-warn-app/cwa-documentation/blob/main/scoping_document.md

REFERENCES

[1] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proceedings of the 2006 international symposium on Software testing and analysis*, 2006, pp. 25–36.

[2] J. Frattini, J. Fischbach, D. Mendez, M. Unterkalmsteiner, A. Vogelsang, and K. Wnuk, "Causality in requirements artifacts: prevalence, detection, and impact," *Requirements Engineering*, vol. 28, no. 1, pp. 49–74, 2023.

[3] J. Fischbach, J. Frattini, D. Mendez, M. Unterkalmsteiner, H. Femmer, and A. Vogelsang, "How do practitioners interpret conditionals in requirements?" in *Product-Focused Software Process Improvement: 22nd International Conference, PROFES 2021, Turin, Italy, November 26, 2021, Proceedings 22*. Springer, 2021, pp. 85–102.

[4] J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, "Towards causality extraction from requirements," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 388–393.

[5] J. Fischbach, J. Frattini, A. Spaans, M. Kummeth, A. Vogelsang, D. Mendez, and M. Unterkalmsteiner, "Automatic detection of causality in requirement artifacts: the cira approach," in *Requirements Engineering: Foundation for Software Quality: 27th International Working Conference, REFSQ 2021, Essen, Germany, April 12–15, 2021, Proceedings 27*. Springer, 2021, pp. 19–36.

[6] J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz *et al.*, "Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study," *Journal of Systems and Software*, vol. 197, p. 111549, 2023.

[7] J. Fischbach, J. Frattini, and A. Vogelsang, "Cira: A tool for the automatic detection of causal relationships in requirements artifacts," *arXiv preprint arXiv:2103.06768*, 2021.

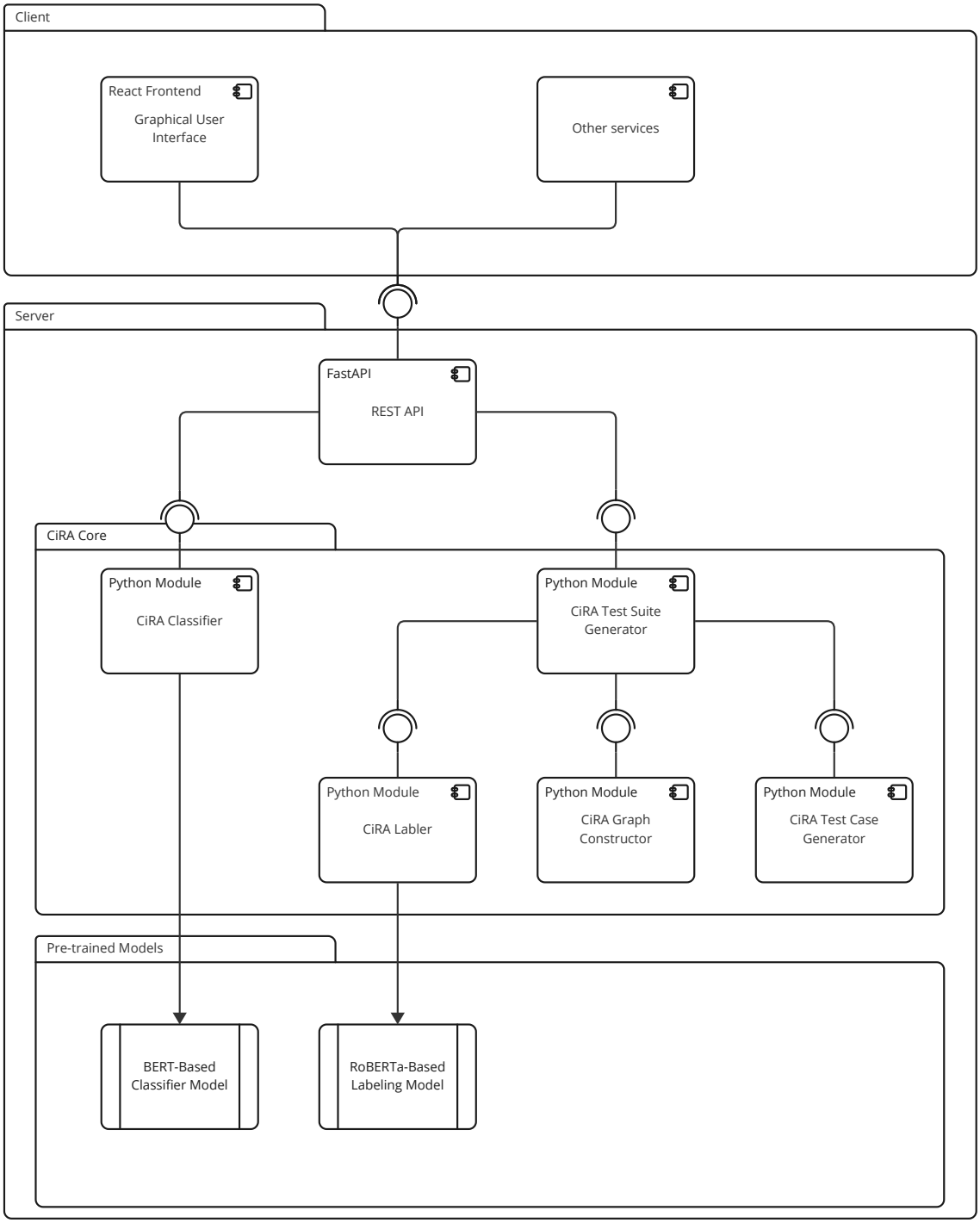


Fig. 3. Architecture of the CiRA Tool

CiRA Demonstrator

CiRA core version 0.9.4

Enter a causal requirements and click 'analyze' to automatically identify the embedded causal relationship.

When the red button is pushed or the power fails and backup power is not available then the system saves all files and shuts down.

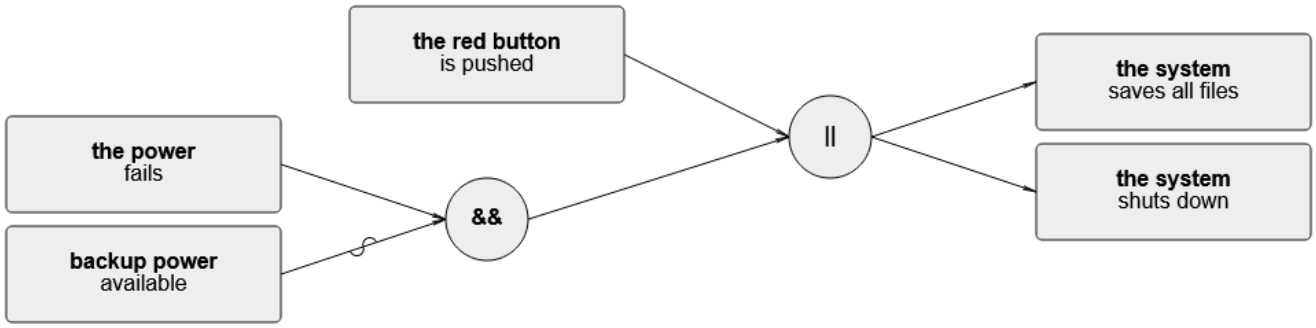
Analyze

Labels

Cause1 Variable Condition || Cause2 Variable c && Cause3 Variable n Condition Effect1 Variable Condition && Effect2 Condition

When the red button is pushed or the power fails and backup power is not available then the system saves all files and shuts down.

Cause-Effect-Graph



Test Suite

ID	Input			Output	
	the red button	the power	backup power	the system	the system
1	is pushed	not fails	not available	saves all files	shuts down
2	is pushed	fails	available	saves all files	shuts down
3	not is pushed	fails	not available	saves all files	shuts down
4	not is pushed	not fails	not available	not saves all files	not shuts down
5	not is pushed	fails	available	not saves all files	not shuts down

Fig. 4. User Interface for the CiRA tool