INDIRECT: Intent-driven Requirements-to-Code Traceability

Tobias Hey
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology,
Karlsruhe, Germany
hey@kit.edu

***Abstract*—Traceability information is important for softwar maintenance, change impact analysis, software reusability, an other software engineering tasks. However, manually generatin this information is costly. State-of-the-art automation approache suffer from their imprecision and domain dependence. I propos INDIRECT, an intent-driven approach to automated requirement to-code traceability. It combines natural language understandin and program analysis to generate intent models for both re quirements and source code. Then INDIRECT learns a mappin between the two intent models. I expect that using the two inten models as base for the mapping poses a more precise and genera approach. The intent models contain information such as th semantics of the statements, underlying concepts, and relation between them. The generation of the requirements intent mode is divided into smaller subtasks by using an iterative natura language understanding. Likewise, the intent model for source cod is built iteratively by identifying and understanding semanticall related source code chunks.**

***Index Terms*—Requirements Traceability, Traceability Link Recovery, Natural Language Understanding, Program Analysis**

I. Introduction

Despite broad agreement on its usefulness, traceability information is still absent in most software projects. This is mainly due to the high manual effort for generation. Traceability information offers access to deeper analyses that need information on the relations between requirements and source code. Thus, being able to automatically link source code to requirements and vice versa empowers tasks such as change impact analysis, requirements validation, or software reusability analysis [1].

For more than a decade approaches to automated traceability link recovery made use of information retrieval (IR) techniques [2]. Recent approaches apply deep learning [3], [4]. Despite great advances the approaches still lack the precision necessary for use in practice. They struggle to link syntactically unrelated artifacts. An example is depicted in Figure 1. The requirement, “The system shall be able to combine cells with elementary arithmetic.” is supposed to be linked to the method `apply(range:Cell[],op:Operator)`. To identify the link between these two artifacts an approach needs to understand that elementary arithmetic should be used for an operation on cells. But neither the operation mentioned (*combine*) nor the concept of elementary arithmetic is used in the source code. A mapping between the two artifacts is only

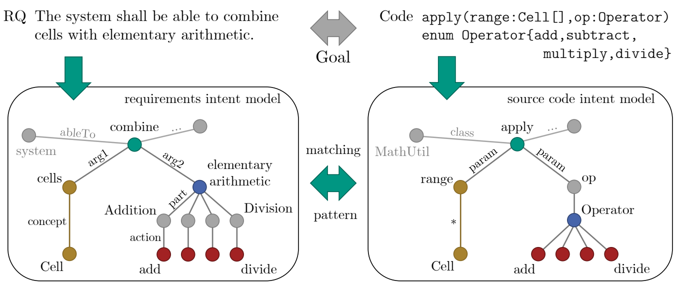

Fig. 1. Requirements-to-code trace with exemplary intent models and matching components (similarly colored).

possible by understanding the intent of *combine* and `apply` and map `Operator` to *elementary arithmetic*. Humans solve this scenario by understanding that the meaning of applying an arithmetic operator on several cells is similar to a combination of cells with elementary arithmetic.

Research on programming in natural language faces similar challenges. During the work on PARSE we proposed to explicitly model the intent of natural language input by using iterative natural language understanding (NLU) to gain insight into the semantics [5]. By dissembling the generation of the model into subtasks we reduced the extent of each task and were able to generate intent models iteratively.

I propose INDIRECT, an INtent-DrIven REquirements-to-Code Traceability approach. The idea is to model the intent of both, natural language requirements and source code, to model the stakeholders and the developers intents. To overcome the limitations of present approaches for automated traceability link recovery, INDIRECT uses the intent models as intermediate representation instead of mapping between artifacts directly. An intent model is a graph; it contains knowledge such as underlying concepts and relations between them. By learning a mapping between these two intent models I expect to achieve a more precise and general approach. This approach leads to the following research questions:
RQ1: Can the intent of natural language requirements be captured just as well as the intent of spoken utterances by generating an explicit intent model with iterative NLU?
RQ2: How can an intent model of source code be generated?
RQ3: Does utilizing requirements and source code intent models for mapping achieve more precise automated traceability

link recovery?

II. Intent-driven Requirements-to-Code Traceability

INDIRECT generates requirements and source code intent models and establishes mappings between these models to generate traceability links. It targets requirements written in unrestricted natural language and object-oriented source code.

Requirements Intent Model (RQ1): The requirements intent model is a graph that represents semantic relations between elements in the requirements. The semantic relations comprise information such as underlying concepts and the connection between entities and statements. The relations represent knowledge useful to interpret and link requirement elements to each other. The model is generated by an iterative natural language understanding inspired by the approach taken in PARSE [5]. This enables the analysis to benefit from information gained in previous iterations.

The analysis steps include syntactic analyses, knowledge enhancement, and conceptualization. The first copes with the grammatical structure of the requirements and connects the entities and statements to each other. It generates information such as that *combine* in Figure 1 is the action to perform and its arguments are *cells* and *elementary arithmetic*. The knowledge enhancement step extends the intent model with knowledge gained from world and domain knowledge bases. This knowledge comprises information such as that addition and division are operations in elementary arithmetic. The conceptualization enhances the intent model with information on the underlying concepts; information such as that each instance of a checkbox is part of a form.

Source Code Intent Model (RQ2): The source code intent model resembles the structure of the requirements intent model but represents relations between parts of the source code. Those relations include structural information on the source code and semantic information gained from analyzing attached resources such as comments, documentation, and commits. The building blocks of the source code intent model are semantically related chunks of source code. Parts of a chunk share a certain intent. As semantically related code can be spread across source code parts (cross-cutting concerns), the chunks can either consist of consecutive or spread parts of code. For the task of generating the intent model I plan to analyze the following resources: the structure of the source code and its comments, available documentation, commits, and test code. The structure of the source and test code as well as files associated to certain commits provide insights into the relations among source code parts. Those relations are used to determine the extent of semantically related source code chunks. The comments and documentation are analyzed by a natural language understanding unit similar to the one used for the requirements. The results are combined with the structural and relational information to form the source code intent model. The generation of the source code intent model is also iterative. This approach is able to cope with different amounts of available inputs (projects might offer no documentation or comments).

Traceability Link Recovery (RQ3): The two intent models are used for automated traceability link recovery. The models serve as intermediate representations between which the traceability links are identified. The identification process itself is conducted by learning patterns that indicate a similarity in the represented intents. As the intent models form an abstract representation of the actual instances of requirements and source code, I expect a learning approach to learn more general patterns. An example for such a pattern is depicted in Figure 1. The similarly colored parts of the intent models form a similar structure with matching subgraphs.

Contributions and Validation: The contributions of this research are: a method that generates explicit intent models for natural language requirements, a method to identify semantically related source code chunks and model their intents, and a tool that utilizes the models to identify traceability links between requirements and source code. To assess the efficiency of the proposed approach a data set that includes at least natural language requirements, source and test code, and traceability links between these artifacts is necessary. Developing or discovering such a data set will be part of the proposed research. Based on the included gold standard traceability links precision and recall of the approach will be assessed.

III. Conclusion and Future Work

To overcome the limitations of present approaches to automated traceability link recovery I propose INDIRECT, an approach that uses intent models for identifying traceability links. It integrates both natural language understanding and program analysis to explicitly model the intents of requirements and source code. The two intent models are leveraged to map between requirements and source code. Thereby, I expect to learn a more domain-independent and precise traceability link recovery approach.

The requirements are in the focus now and first steps towards the generation of the source code intent model are taken. The mapping between the two models will follow consecutively. First results on interpreting requirements are promising but the success of the approach has still to be quantified.

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[2] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, Dec. 2014.

[3] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically Enhanced Software Traceability Using Deep Learning Techniques," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 3–14.

[4] W. Wang, N. Niu, H. Liu, and Z. Niu, "Enhancing Automated Requirements Traceability by Resolving Polysemy," in *2018 IEEE 26th International Requirements Engineering Conference*, 2018, pp. 40–51.

[5] S. Weigelt, T. Hey, and W. F. Tichy, "Context Model Acquisition from Spoken Utterances," in *The 29th International Conference on Software Engineering & Knowledge Engineering,*, Jul. 2017, pp. 201–206.