

The ARDoCo Tool Landscape: REST API, TraceView, and TraceViz for Architecture Traceability

Jan Keim
Dominik Fuchß
jan.keim@kit.edu
dominik.fuchss@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Sophie Corallo
Tobias Hey
sophie.corallo@kit.edu
hey@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Julian Winter
Kevin Feichtinger
julian.winter@student.kit.edu
kevin.feichtinger@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Abstract

Context and Problem. Software development produces interrelated artifacts like software architecture documentation (SAD), software architecture models (SAMs), and source code, whose relationships are essential for maintenance and consistency checking. However, automatically recovering links between these artifacts (traceability link recovery (TLR)) remains difficult to deploy in practice.
Method and Aim. We present an accessible tool landscape for ARDoCo's TLR approaches: the **ARDoCo REST API** exposes four TLR pipelines (SAD-SAM, SAM-Code, SAD-Code, and SAD-SAM-Code) via HTTP endpoints with asynchronous execution and caching; **TraceView** is a browser-based frontend with a guided wizard and interactive multi-panel exploration of recovered links and inconsistencies; and **TraceViz**, which is a VS Code extension that overlays trace links directly onto documentation in the IDE.
Results and Conclusion. All three components are publicly deployed and usable. A preliminary study for TraceViz's in-IDE visualization confirmed that it improves developer comprehension during software understanding tasks. The tool landscape makes state-of-the-art TLR accessible to architects, developers, and tool integrators.
Video. We provide a screencast of our ARDoCo Tool Landscape and how it is used here: https://youtu.be/IOTEPZQ3tVs

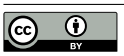
CCS Concepts

• **Software and its engineering** → *Software design engineering*; **Documentation**; *Maintaining software*; *Software evolution*; **Software architectures**; • **Computing methodologies** → **Natural language processing**; **Information extraction**.

Keywords

software traceability, software architecture, documentation

ACM Reference Format:
Jan Keim, Dominik Fuchß, Sophie Corallo, Tobias Hey, Julian Winter, and Kevin Feichtinger. 2026. The ARDoCo Tool Landscape: REST API, TraceView, and TraceViz for Architecture Traceability. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3832783.3834621



This work is licensed under a Creative Commons Attribution 4.0 International License.
ASE '26, Munich, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3834621

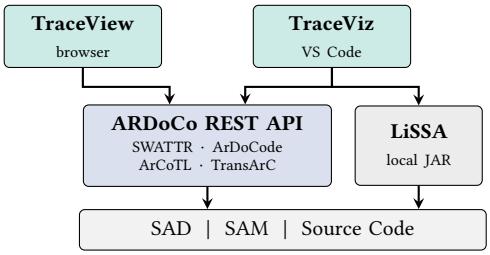


Figure 1: ARDoCo tool landscape: TraceView and TraceViz serve different user needs. Both use the REST API. TraceViz additionally supports LiSSA. Arrows indicate dependencies.

1 Introduction

Software systems are documented at multiple levels of abstraction: architects write natural-language software architecture documentations (SADs) to record principal design decisions, create software architecture models (SAMs) to formally specify components and their interactions, and developers implement the design in *source code*. Keeping these artifacts consistent and understandable, especially with regard to their mutual relations, is a prerequisite for effective maintenance, change impact analysis, and evolution [1, 12]. Yet, manually maintaining trace links is tedious and error-prone, which makes automated traceability link recovery (TLR) essential in practice. Automated TLR is inherently difficult due to the abstraction gaps between natural language, models, and code.

The ARDoCo project[1] develops and maintains a family of complementary, empirically validated TLR approaches: SWATTR (SoftWare Architecture Text Trace link Recovery) [11] for SAD-SAM links; an inconsistency detection approach based on SWATTR [9]; ArCoTL (Architecture–Code Trace Links) and TransArC (Transitive Architecture-to-Code) [8] for SAM-Code and transitive SAD-Code links; and LiSSA (Linking Software System Artifacts) [3] for retrieval-augmented generation (RAG)-based TLR across artifact types. Recent work uses large language models (LLMs) to extract component names and identify named architecture entities [4, 5].

Despite strong empirical results, these approaches are available only as Java libraries or command-line tools, requiring technical expertise to configure and run. This barrier can prevent their adoption by architects and developers, who would benefit the most.

[1] https://ardoco.de

This paper closes that gap with a tool landscape that exposes the ARDoCo TLR family through three complementary interfaces tailored to different user needs (Figure 1):

(1) **ARDoCo REST API** for developers and tool integrators: a Spring Boot service exposing four TLR pipelines via HTTP with asynchronous execution and result caching.
(2) **TraceView** for architects and non-developers: a zero-install browser frontend that shows all three artifact types side by side and exposes ARDoCo's inconsistency detection. TraceView is optimized for SAD-SAM-Code TLR.
(3) **TraceViz** for developers: a VS Code extension optimized for SAD-Code TLR that overlays trace links as gutter markers, enabling one-click navigation from documentation to linked code elements [13].

The interfaces are publicly deployed and require little to no local installation, making TLR accessible via browser, IDE, or REST API.

2 Background: The ARDoCo TLR Approaches

ARDoCo addresses TLR between three categories of software artifacts: SAD (natural language), SAM (e.g. UML or Palladio component models), and source code. The approaches below form the algorithmic foundation of the tool landscape presented in this paper.

SWATTR (SAD-SAM). SWATTR [11] is a framework that uses natural language processing (NLP) and heuristics in a multi-stage pipeline to recover trace links between sentences in a SAD and components in a SAM. We later extended SWATTR with detection of two types of artifact inconsistencies [9]: Text Entity Absent from Model (TEAM), where a text entity has no counterpart in the model, and Model Entity Absent from Text (MEAT), where a model entity has no counterpart in the text.

ArCoTL (SAM-Code). ArCoTL [8] recovers trace links between SAMs and source code by transforming both into intermediate representations and combining text similarity with various heuristics.

TransArC / ArDoCode (SAD-Code). TransArC [8] composes the approaches SWATTR and ArCoTL transitively (SAD → SAM → Code) to recover direct trace links between SADs and source code. ArDoCode is a simpler variant that applies SWATTR heuristics directly to code without requiring a SAM. The approach is easier to deploy but with lower performance.

LiSSA (generic). LiSSA [3] is a generic RAG-based TLR approach: for each source artifact, it uses information retrieval (IR) to retrieve candidate targets, then queries an LLM to confirm trace links, supporting multiple artifact type pairs. Unlike the heuristic-based approaches, LiSSA is not limited to the SAD-SAM-Code pipeline and has demonstrated strong performance on e.g. requirements-to-requirements TLR [7]. However, it requires access to an LLM.

All heuristic-based approaches were evaluated on a common benchmark [2] of five open-source Java projects: MediaStore, TeaStore, TEAMMATES, BigBlueButton, and JabRef.

3 The ARDoCo Tool Landscape

The ARDoCo tooling addresses architecture traceability for different user needs through three complementary components: the **ARDoCo REST API** as the shared computational backend, **TraceView** as an installation-free browser frontend optimized for SAD-SAM-Code TLR with simultaneous multi-artifact visualization and inconsistency detection, and **TraceViz** as a VS Code extension optimized for SAD-Code TLR with in-IDE navigation from documentation to code and support for our most recent TLR approach LiSSA. All components are distributed as Docker images or extension packages and are publicly available.

3.1 ARDoCo REST API

The ARDoCo REST API provides programmatic access to ARDoCo's TLR pipelines over HTTP, enabling any application to trigger TLR analyses and retrieve structured results without bundling the Java libraries. It is implemented in Java using Spring Boot and is publicly deployed[2]. The API exposes the four ARDoCo TLR pipelines as HTTP services organized in four controllers:

- `sad-sam`: SAD-SAM TLR with TEAM/MEAT inconsistency detection (SWATTR)
- `sam-code`: SAM-Code TLR (ArCoTL)
- `sad-code`: SAD-Code TLR without SAM (ArDoCode, baseline)
- `sad-sam-code`: transitive SAD-Code TLR via SAM (TransArC)

Each controller exposes multiple endpoints: `start` (fire-and-forget, returns a run ID), `start-and-wait` (synchronous), `get-result` (retrieve result by ID), and `wait-for-result` (long-poll for completion). Pipeline runs are identified by a hash of the request content, enabling *transparent caching*. Meaning, results are stored in a Redis instance, and are reused for identical inputs without re-running the pipeline. Cached results expire after 24 hours. All endpoints return structured JSON and can be explored interactively via the integrated Swagger UI.

3.2 TraceView

TraceView[3] is a browser-based, installation-free tool optimized for SAD-SAM-Code TLR. It displays all three artifact types simultaneously and exposes ARDoCo's inconsistency detection, making it the primary tool for architects exploring full-stack traceability. TraceView is implemented with Next.js and React (TypeScript), and utilizes the ARDoCo REST API.

Users create a new TLR project through a four-step wizard: (1) *Upload files*: upload your artifacts (e.g., a plain-text SAD, an UML or Palladio Component SAM, and a code model); (2) *Project details*: enter a project name; (3) *Configure*: select the TLR pipeline based on the submitted artifact types (SWATTR, ArDoCode, ArCoTL, or TransArC); (4) *Summary*: review the configuration before start.

After submission, TraceView polls the REST API and notifies the user when results are ready. Results are shown in a configurable *multi-panel view* with up to three resizable panels allowing the user to display the SAD, the SAM, and the source code as well as the detected trace links or inconsistencies side by side (cf. Figure 2). Selecting an element in any panel highlights its linked counterparts in the other panels, enabling cross-artifact navigation.

[2] https://rest.ardoco.de
[3] https://tv.ardoco.de

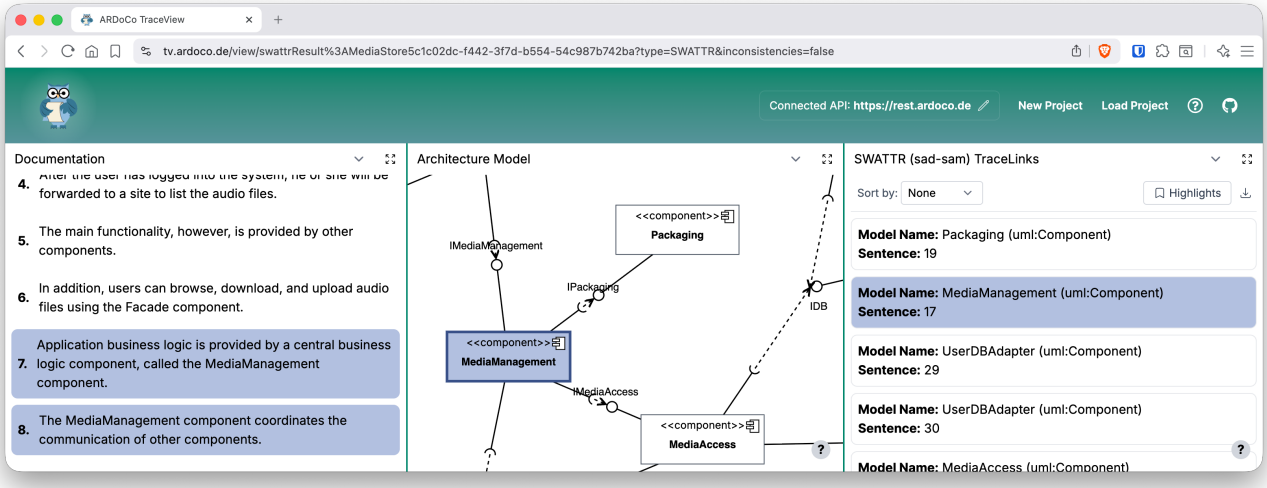


Figure 2: TraceView: SAD documentation (left), SAM (center), and recovered SAD-SAM trace links (right) shown side by side. Selecting a trace link highlights the linked SAD sentence and SAM component across all panels.

3.3 TraceViz

TraceViz[4] is a VS Code extension optimized for SAD-Code TLR, placing trace links directly in the developer's editing context. It visualizes links between natural language documentation and source code as inline gutter markers. Thus, enabling the navigation from any SAD sentence to the linked code files and back without leaving the IDE. Accessible from the VS Code activity bar, it provides a split sidebar: the *Traceability Approaches* panel for configuring and triggering TLR, and the *Trace History* panel listing past runs that can be re-activated with one click.

TraceViz supports three trace link sources: (i) the **ARDoCo REST API** (support for ArDoCode and TransArC); (ii) **CSV import** from any TLR tool matching the format that the gold standards have [2], making TraceViz approach-agnostic; and (iii) **LiSSA** via local JAR invocation.

Once loaded, lines with links are marked with a colored *gutter dot* (cf. Figure 3). Hovering over these gutter dots reveals the link count, and clicking (or using the CodeLens annotation or status bar button) opens a Quick Pick menu for navigating to linked artifacts. Up to two trace link sets can be displayed simultaneously in selectable colors for side-by-side comparison. A *directory heuristic* replaces per-file markers with a single directory-level gutter dot when all files in a folder link to the same line, reducing visual noise.

4 Evaluation

Evaluations results for ARDoCo TLR approaches. The TLR algorithms exposed by the REST API have been evaluated in depth in their respective publications [8, 9]. On the ARDoCo benchmark [2] of five open-source Java projects (MediaStore, TeaStore, TEAMMATES, BigBlueButton, JabRef), SWATTR achieves an average F1 of 0.81 for SAD-SAM, ArCoTL achieves 0.98 for SAM-Code, and TransArC achieves 0.82 for SAD-Code, significantly outperforming the best baseline (ArDoCode, F1 = 0.37) [8]. Thus, according to the classification scheme of Hayes et al. [6], our best TLR approaches can achieve excellent performance. The inconsistency detection achieves F1 = 0.89 for MEAT [9].

Preliminary User Study of TraceViz. To assess the usefulness of IDE-integrated trace link visualization, Winter [13] conducted a think-aloud user study of TraceViz ($n = 7$; three doctoral candidates, three master's graduates, one industry developer with more than ten years of experience). Participants completed two software comprehension tasks on the TEAMMATES project: one without visualization (raw CSV trace links) and one aided by TraceViz with SAD-Code links.

Six of the seven participants found the visualizations *helpful* in completing the tasks, and all seven agreed that the visualizations *supported their comprehension process*. Think-aloud recordings revealed that without visualization, participants relied on repetitive manual tree traversal, whereas TraceViz enabled direct jumps from documentation sentences to relevant source files. Three of the seven participants stated that they would use this visualization in a real-world context (four were undecided), highlighting its potential for onboarding and for navigating large code bases. Despite the limited number of participants, the results indicate that trace link visualization is perceived as a meaningful improvement to efficiency and accuracy when navigating source code.

5 Conclusion

We presented the ARDoCo tool landscape, making a family of TLR approaches available to software developers and architects through three complementary interfaces: the **ARDoCo REST API**, which exposes four TLR pipelines via HTTP endpoints with asynchronous execution and Redis-backed caching; **TraceView**, a browser-based wizard with an interactive multi-panel result view; and **TraceViz**, which integrates trace links directly into Visual Studio Code for

[4] https://github.com/ardoco/traceviz

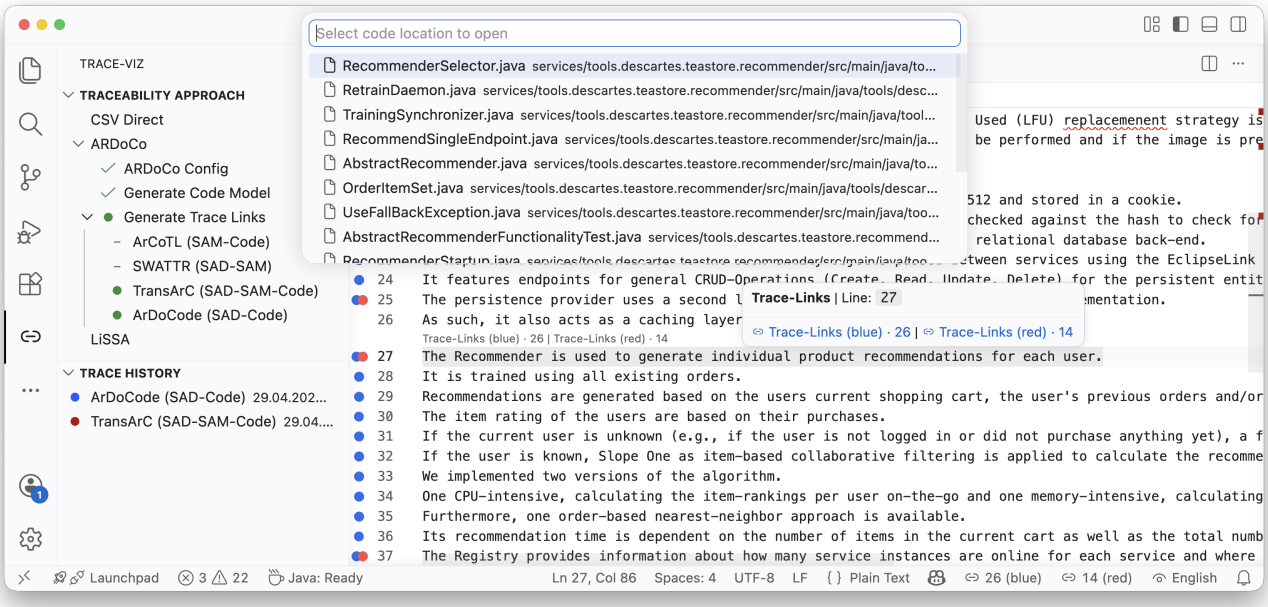


Figure 3: TraceViz in VS Code: colored gutter dots mark SAD-Code trace links. Hovering line 27 reveals per-set counts (blue: 26, red: 14) and a Quick Pick listing linked code files for one-click navigation. The sidebar shows TLR approaches and history.

immediate navigation from documentation to linked code. A preliminary think-aloud user study of TraceViz ($n = 7$) indicates that the in-IDE visualization improves developer comprehension [13].

In future work we plan to integrate the LLM-based TLR approach LiSSA and the LLM-based architecture component name extraction ExArch [4] into the REST API, making them accessible through both TraceView and TraceViz. Further, we aim at extending support for additional artifact types (e.g., requirements), improving visualization for large projects with many trace links, and conducting larger, controlled user studies across all ARDoCo tools.

Data Availability Statement

We provide a snapshot of our tools at [10]. This snapshot includes the current source code of ARDoCo, ARDoCo REST API, TraceView, and TraceViz. Instructions on how to execute them are included.

Acknowledgments

This work was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 – 501798263, the Topic Engineering Secure Systems of the HGF, and supported by KASTEL Security Research Labs, Karlsruhe. Generative AI tools were used for copy editing.

References

[1] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman (Eds.). 2012. *Software and Systems Traceability*. Springer London, London. doi:10.1007/978-1-4471-2239-5

[2] Dominik Fuchß, Sophie Corallo, Jan Keim, Janek Speit, and Anne Koziolek. 2023. Establishing a Benchmark Dataset for Traceability Link Recovery Between Software Architecture Documentation and Models. In *Software Architecture. ECSA 2022 Tracks and Workshops*, Thais Batista, Tomáš Bureš, Claudia Raibulet, and Henry Muccini (Eds.). Springer International Publishing, Cham, 455–464.

[3] Dominik Fuchß, Tobias Hey, Jan Keim, Haoyu Liu, Niklas Ewald, Tobias Thirolf, and Anne Koziolek. 2025. LiSSA: Toward Generic Traceability Link Recovery Through Retrieval-Augmented Generation. In *IEEE/ACM 47th International Conference on Software Engineering (ICSE)* (Ottawa, Canada). IEEE, 1396–1408. doi:10.1109/icse55347.2025.00186

[4] Dominik Fuchß, Haoyu Liu, Sophie Corallo, Tobias Hey, Jan Keim, Johannes von Geisau, and Anne Koziolek. 2026. Who's Who? LLM-assisted Software Traceability with Architecture Entity Recognition. *ACM Trans. Auton. Adapt. Syst.* (April 2026). doi:10.1145/3807453

[5] Dominik Fuchß, Haoyu Liu, Tobias Hey, Jan Keim, and Anne Koziolek. 2025. Enabling Architecture Traceability by LLM-based Architecture Component Name Extraction. In *IEEE 22nd International Conference on Software Architecture (ICSA)*. IEEE, 1–12. doi:10.1109/icsa65012.2025.00011

[6] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. 2006. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *IEEE TSE* 32, 1 (Jan. 2006), 4–19. doi:10.1109/TSE.2006.3

[7] Tobias Hey, Dominik Fuchß, Jan Keim, and Anne Koziolek. 2025. Requirements Traceability Link Recovery via Retrieval-Augmented Generation, In Lecture Notes in Computer Science, Anne Hess and Angelo Susi (Eds.). *Requirements Engineering: Foundation for Software Quality*. doi:10.1007/978-3-031-88531-0_27

[8] Jan Keim, Sophie Corallo, Dominik Fuchß, Tobias Hey, Tobias Telge, and Anne Koziolek. 2024. Recovering Trace Links Between Software Documentation And Code, In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal). *ICSE*, 1–13. doi:10.1145/3597503.3639130

[9] Jan Keim, Sophie Corallo, Dominik Fuchß, and Anne Koziolek. 2023. Detecting Inconsistencies in Software Architecture Documentation Using Traceability Link Recovery. In *IEEE 20th International Conference on Software Architecture (ICSA)*. IEEE, 141–152. doi:10.1109/icsa56044.2023.00021

[10] Jan Keim, Dominik Fuchß, Sophie Corallo, Tobias Hey, Julian Winter, and Kevin Feichtinger. 2026. *Replication Package for "The ARDoCo Tool Landscape"*. doi:10.5281/zenodo.21533985

[11] Jan Keim, Sophie Schulz, Dominik Fuchß, Claudius Kocher, Janek Speit, and Anne Koziolek. 2021. Trace Link Recovery for Software Architecture Documentation. In *Software Architecture*, Stefan Biffl, Elena Navarro, Welf Löwe, Marjan Sirjani, Raffaela Mirandola, and Danny Weyns (Eds.). Springer International Publishing. doi:10.1007/978-3-030-86044-8_7

[12] Patrick Mäder and Alexander Egyed. 2012. Assessing the effect of requirements traceability for software maintenance. In *2012 28th IEEE International Conference on Software Maintenance*. doi:10.1109/ICSM.2012.6405269

[13] Julian Robin Winter. 2025. *Guided Exploration and Visualization of Trace Links in Visual Studio Code*. Bachelor's Thesis. Karlsruher Institut für Technologie (KIT). doi:10.5445/IR/1000192928