Antora Tracer

Architecture

Version 0.7.0, 2026-07-29

This document describes the architecture of the Antora Tracer extension using selected arc42 sections. Each section is a traceable `[item]` that references the requirements it addresses.

1. Introduction and Goals

ARC-001 — Project scope and architectural goals

Project scope and architectural goals

The Antora Tracer is an Antora extension that adds role-based requirements traceability to AsciiDoc documentation. It replaces fixed macro types with a single configurable `[item]` macro.

Architectural goals:

- Enable users to define their own traceability domain model (roles, relations, matrices) through YAML configuration
- Maintain a clean separation between parsing, graph storage, validation, matrix generation, and export
- Ship with built-in presets for common domains without requiring users to write configuration
- Integrate with Antora's event pipeline without imposing runtime dependencies on users who don't need specific features
- Provide a standalone CLI for use outside Antora

Addresses

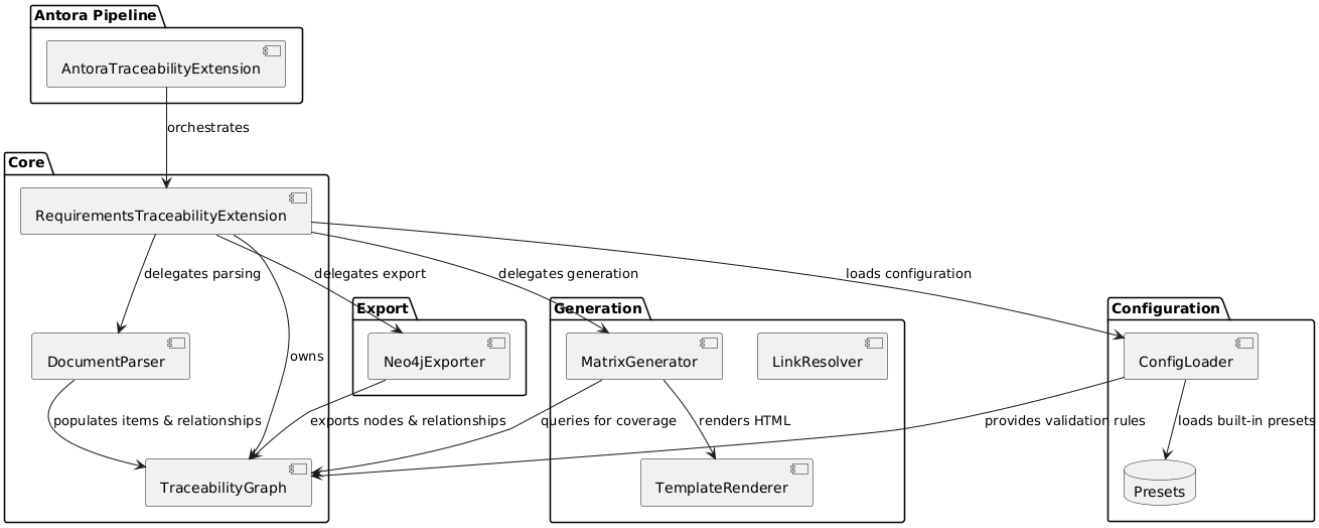
- REQ-001 — Single `item` macro replaces all block macros
- REQ-006 — Configuration defines roles
- REQ-009 — Configuration is required for validation
- REQ-013 — Role-based relation validation
- REQ-018 — Built-in preset configurations
- REQ-024 — Neo4j CSV export
- REQ-035 — Antora extension integration
- REQ-036 — CLI commands

[Relationship graph for ARC-001] |
https://kroki.io/graphviz/svg/eJyd09Fq2zAUBuD7PsXBu03Ajh1jUzxIs8EGZWzp2E06qCIdO-pkyTuSu3RjsIfYE_ZJRhR7JCwrie-E_R999tGRkBWxZg0fiXFkK6mke4QfFwDE9BchqbheXF4AaCMQlnbNGixWZjMC6x4VFgGZVgsUo1IqhSIYQWm006zGIniD6gGd5Cz4vN0BRYWwPPZ6V2Tldyyi0GeD2WI-DsMogOV2Y26UoSJ4MQ2vsnTe5XcPv62lwxEotkJV9GW3ulvA06_f8J7MPXIHlpsGgWkBjPi2jLuWuo8LFq8_HAGTWR6-yp8Bu7Jb3S08eCN1pRDupMP6DmrGyQBhoxhHC4dgOgxMezD14NzoUlYtMSeNBoGl1GiBjEJ7oOXDtLzX8iOatED4tZWEAkpDsA9G8SAwijswij24MArHK2ZRAKHasQ9MSeGXB2A2DMx6MPPgVSuVG0sNDaFFB3z_jw9aOkkGgZOkAyeJB9-hSe5hfvMJcNMYcvtEPB1ExNOOiKeemGlniAFuHGrrD047rOifFsbDRjLuRzLuRvL6LXBT10yLvmF_7_T45f5963ZhQhBai8-k07PS-TlpP6mnp7Nz0n5GTk774z49_b-e_PwDH4nKVA

2. Building Block View

ARC-002 — Component-level architecture

The extension is composed of eight modules organized in a layered architecture:



Component	Responsibility
`AntoraTraceabilityExtension`	Antora entry point. Registers event handlers, creates `RequirementsTraceabilityExtension`, reads playbook config.
`RequirementsTraceabilityExtension`	Orchestrator. Owns the graph, delegates to parser, matrix generator, and Neo4j exporter. Public API for programmatic use.
`DocumentParser`	Regex-based AsciiDoc parser. Two-pass: extracts `[item]` blocks, then scans for inline `relation\:TARGET[]` macros.
`TraceabilityGraph`	In-memory directed graph. Items stored as `Map<string, Item>`, relationships as `Map<string, ItemRelationship>`. Provides query, validation, path finding, and impact analysis.
`MatrixGenerator`	Config-driven matrix generation. Reads matrix definitions from configuration, queries the graph, computes coverage, produces CSV output. Delegates HTML to TemplateRenderer.
`TemplateRenderer`	Mustache wrapper. Loads templates from `src/templates/`, supports custom template directories with fallback to built-ins.
`LinkResolver`	Path resolution for matrix-to-item deep links. Strips `pages/\` prefix and `.adoc\` extension from sourceFile values, generates `../../component/page.html#ID\` links using a configurable `relativePathPrefix\`.
`Neo4jExporter`	Graph-to-Neo4j export. Produces `nodes.csv`, `relationships.csv`, and `import.cypher`. Properly escapes special characters.
`ConfigLoader`	YAML configuration loader. Loads and validates config, merges with presets, provides role/relation validation queries to the graph.

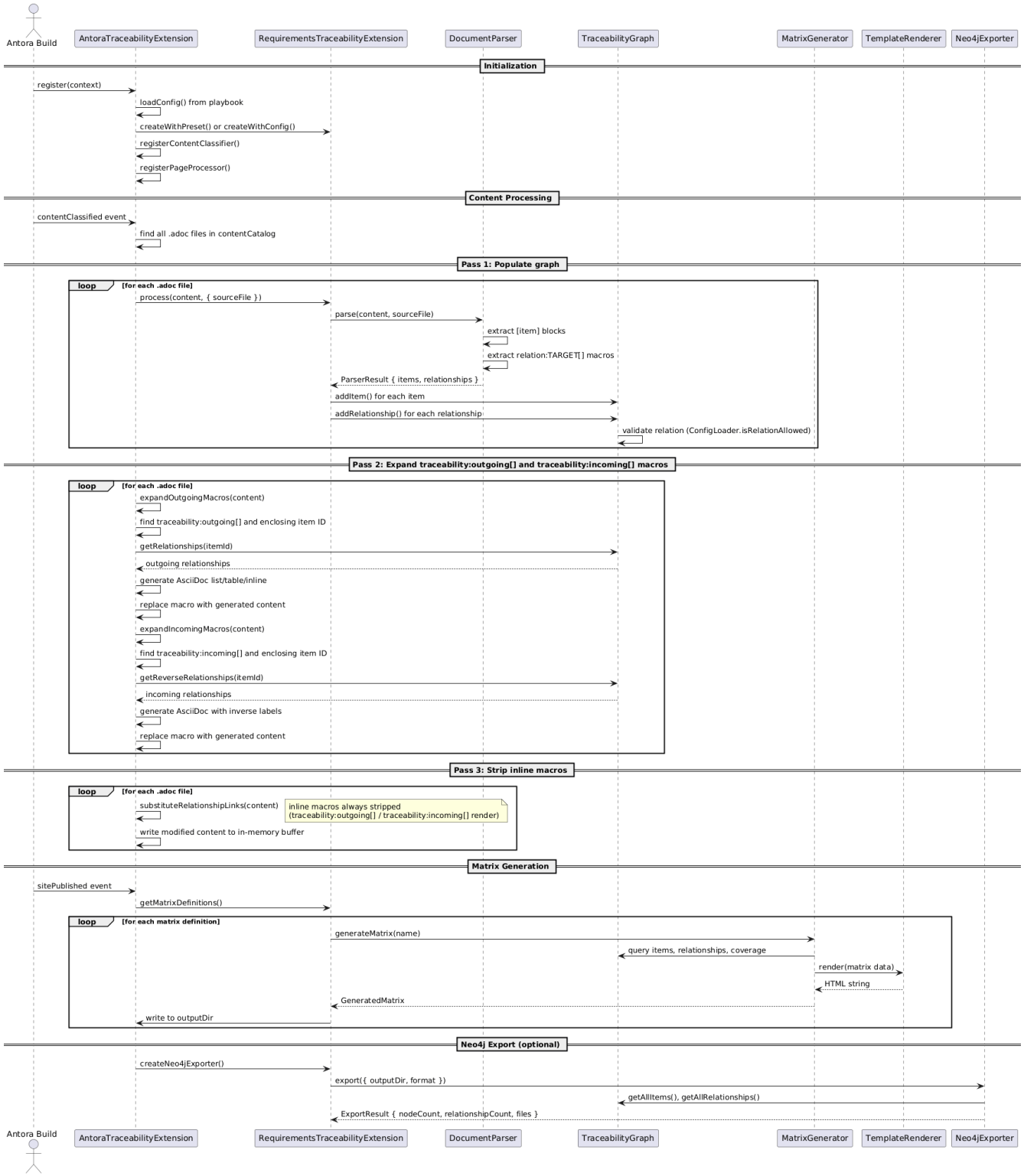
Addresses

- REQ-001 — Single `item` macro replaces all block macros
- REQ-002 — `item` macro accepts role attribute
- REQ-003 — `item` macro maintains existing attributes
- REQ-004 — `item` macro supports block content
- REQ-006 — Configuration defines roles
- REQ-007 — Configuration defines relations
- REQ-008 — Configuration defines matrices
- REQ-011 — Configuration validation
- REQ-024 — Neo4j CSV export
- REQ-025 — Neo4j Cypher export
- REQ-029 — Configurable matrix generation with coverage calculations
- REQ-034 — Mustache template rendering

3. Runtime View

ARC-003 — Processing flow during Antora build

The following sequence shows how an AsciiDoc file flows through the system during an Antora build:



Key timing characteristics:

- **Fire-and-forget async init**: `AntoraTraceabilityExtension` constructor starts `initializeAsync()` without awaiting it. Event handlers are registered after config loads. In practice, the Antora

build pipeline provides sufficient latency.

- **Single-pass processing**: `process()` call does both parsing and graph population in one call. No separate relationship processing pass needed.
- **Lazy rendering**: Templates are loaded and compiled on first use, cached thereafter.

4. Architecture Decisions

ARC-004 — Key architectural decisions and their rationale

This section captures the design decisions from `openspec/changes/unified-item-architecture/design.md`.

Decision	Rationale	Alternatives Considered
Single `[item]` macro with `role` attribute	Simplifies mental model, enables user-defined roles without code changes, reduces parser duplication	Keep separate macros (two ways to do things), role as positional parameter (less readable)
Configuration-based roles and relations (YAML)	Separates concerns, enables validation, allows presets, version-controllable	Hardcoded roles (inflexible), JSON (less readable), JS config (more complex)
Role-based relation validation at processing time	Catches errors early, prevents invalid graphs, provides clear messages	No validation (allows invalid graphs), warning only (users miss issues), runtime validation (too late)
Mustache templates for HTML	Separates presentation from logic, enables user customization, already working from prior work	Embedded HTML generation (code duplication), other template engines (Mustache was already in place)
Neo4j export (not custom query engine)	Mature graph query capabilities, Cypher is industry standard, no need to build query language	Custom DSL (high effort), GraphQL (not graph-optimized), Gremlin (less popular)
Built-in presets for common domains	Gives starting points, reduces config burden, demonstrates best practices	No presets (must define from scratch), default roles (conflicts with flexibility goal)
No default roles	Forces explicit configuration, prevents domain model assumptions	Default roles (simpler but less flexible), fallback behavior (complex transition logic)
Unknown roles = warnings (not errors)	Graceful degradation, partial config allowed, incremental role adoption	Error on unknown (too strict), silent ignore (users miss config issues)
Manual regex parsing over Asciidoctor.js extensions	Simpler, version-independent, easier to test, works consistently across Asciidoctor.js versions	Asciidoctor.js block processors (complex, version-dependent API)

Decision	Rationale	Alternatives Considered
Config-driven matrix generation	Users define which roles go on rows and columns, which relations count as coverage	Hardcoded matrix types (req-impl, req-test, full) — inflexible for domain models with different roles
In-memory source substitution for clickable links	After processing, replace `` with Asciidoctor xrefs in the content catalog buffer. Two-pass approach ensures cross-file targets are resolved. No disk writes.	HTML post-processing (PDF-incompatible), Asciidoctor inline macro extension (API dependency)
Dedicated LinkResolver component	Separates path resolution from matrix generation, makes link generation configurable and testable in isolation. Robust `itemToHtmlPath` strips `pages/` prefix and `.adoc` extension from any sourceFile format.	Link generation in MatrixGenerator (mixes concerns), template-level logic (Mustache cannot do complex path manipulation)