

An automated software metadata harvesting pipeline

CLARIAH FAIR Tool Discovery

Maarten van Gompel, KNAW Humanities Cluster

October 7th, 2024

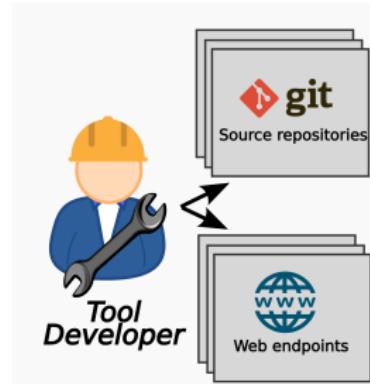
Introduction

- ▶ **Objective:** For scholars it is important to find and identify tools suitable for their research.
 - ▶ Users must be able to make an informed enough judgement
- ▶ **Problem:** Many software catalogues exist, but:
 - ▶ Information is often outdated
 - ▶ manually curated once, version information, provenance, broken links
 - ▶ Information is often incomplete
 - ▶ maintenance status, documentation
 - ▶ Information is duplicated and not aligned
 - ▶ multiple metadata schemes

Introduction (2)

- ▶ We developed a pipeline for **automatic** software metadata harvesting.
- ▶ We **periodically** harvest software metadata **from the source**
- ▶ This is not another software catalogue system; we aim to feed data to various catalogues

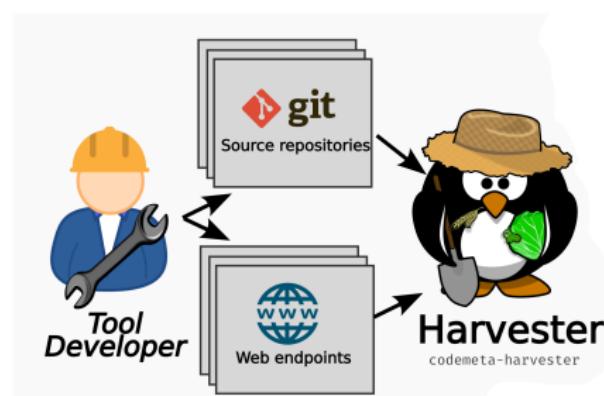
Bottom-up harvesting from the source (1)



- ▶ Software metadata is kept at the source
 1. Metadata lives alongside the source code in a version controlled repository (e.g. git) and published in a forge (e.g. GitHub).
 2. (SaaS) metadata provided directly by web endpoints

Advantages: Full authorship with the developers, versioned, no duplication, no intermediaries

Bottom-up harvesting from the source (2)



- ▶ A minimal manually-curated *source registry* contains all repositories/endpoints to harvest
- ▶ Our harvester periodically checks all of these.
- ▶ For each, it finds and collects various existing and heterogeneous software metadata schemes
 - ▶ Reuse what developers already use, avoid any duplication of metadata.
 - ▶ (e.g `pyproject.toml`, `project.json`, `pom.xml`, badges in `README.md`, `LICENSE`, `AUTHORS`)

Bottom-up harvesting from the source (3)

https://github.com/annotation/stam-rust

Work □ Branches □ Tags □ Go to file □ Add file □ Code □

prycon version bump b89464 2 days ago 1,010 Commits

github/workflows.github d: upgrade checkout action 8 months ago

benches.cleanup 2 weeks ago

sro.query: added missing constraint 5 days ago

tests.renamed AnnotationStore.add() to AnnotationStore with _le... 2 weeks ago

.gitignore.gitignore: ignore env/ last year

CONTRIBUTING.mdadded the formatter to the guidelines as well last year

Cargo.tomverson bump 3 days ago

LICENSEadded README and license 2 years ago

README.mdREADME: updated due to API changes regarding add_data... 2 weeks ago

codemeta-harvest.jsonmetadata: updated technology readiness level 9 months ago

Readme GPL-3.0 license

Activity Custom properties

7 stars 5 watching 0 forks

Report repository

Releases v0.15.3 (Latest) 3 days ago + 22 releases

Packages No packages published Publish your first package

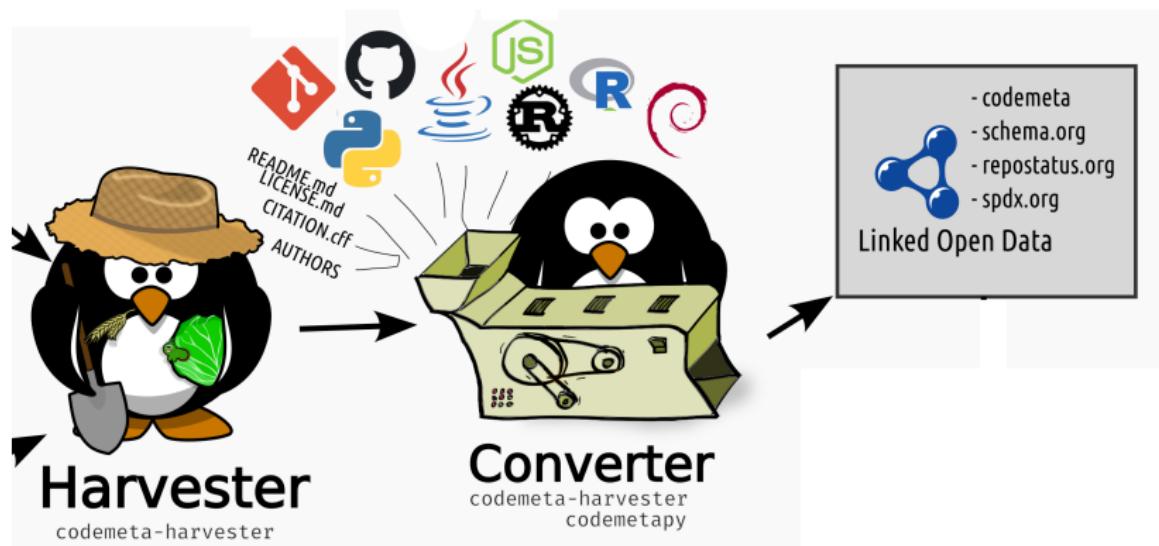
Contributors prycon Maarten van Gompel hayco Hayco de Jong

Languages Rust 100.0%

STAM Library

Many instances of metadata can be found!

Metadata conversion to codemeta



- ▶ Existing heterogeneous software metadata schemes are converted into a uniform linked-open-data representation
 - ▶ CodeMeta and schema.org as our base vocabulary
 - ▶ Additional vocabularies on top (e.g. SPDX, Repostatus.org)
- ▶ The result of the conversion is a single codemeta.json file (JSON-LD)
 - ▶ May be committed back into the source repo

Publishing in catalogues

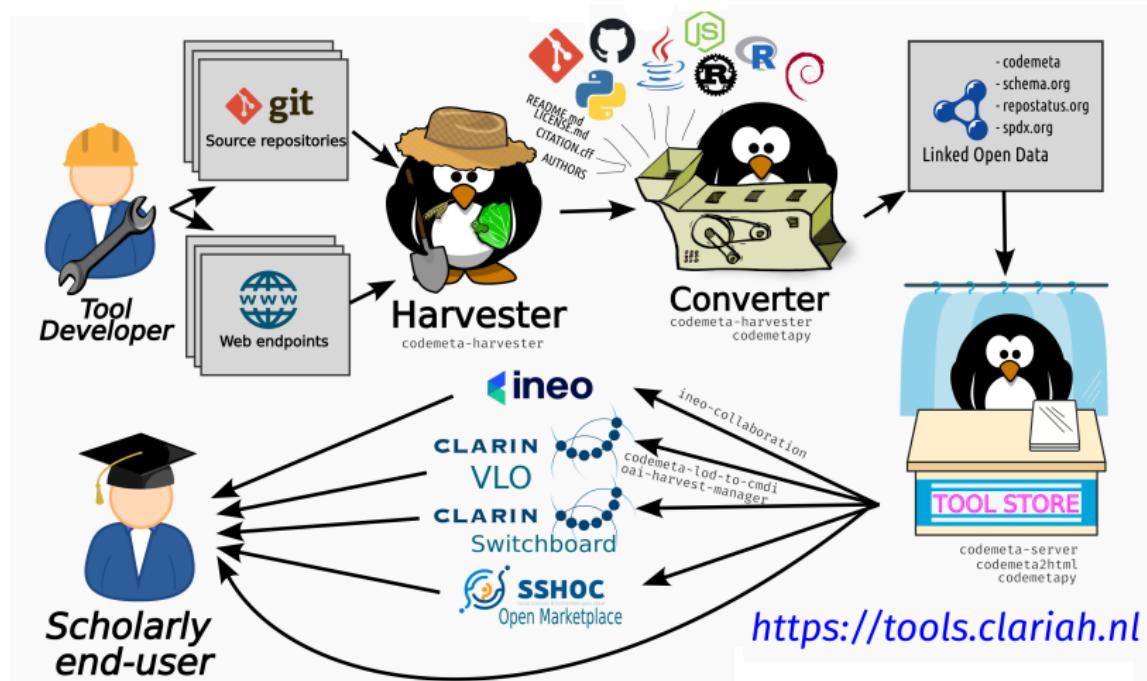


Figure 1: The full architecture

Use-case: CLARIAH (114 source repositories, 34 web endpoints)

Publishing in catalogues (1)

Clariah Tools

Toggle sidebar All tools Services only Table view SPARQL

Table of Contents Filters

- **Alpino**
 - Alpino
 - Alpino-Webservice
- **AlpinoGraph**
- **aluid**
- **analitici**
- **AnnoRepo**
 - AnnoRepo
 - annorepo-client
- **asservice**
- **auchann**
- **Automatic Speech Recognition for Dutch**
- **Blacklab & Corpus Search**
 - A BlackLab Server CLARIN
 - FCS 2.0 endpoint
 - BlackLab Corpus Search
 - INT Corpus Frontend
- **Broccoli**
- **burgerLinker**
- **CHAMD**
- **CLAM**
- **CLARIAH LD Proxy**
- **CLARIAH Tool Discovery**
 - CLARIAH Tool Discovery
 - codemeta-harvester
 - codemeta-lod-to-cmdl
 - codemeta-server
 - codemeta2html
 - CodeMetaPy
- **CMD2RDF**
- **COBALT**
- **Colibri Core**
- **Corpus Editor for Syntactically Annotated Resources (Cesar)**
- **cov_csvw**
- **DANE**
 - DANE
 - dane-asr-worker
 - dane-download-worker
 - DANE-server
 - dane-workflows
- **deepfog**
- **Dexter**
- **did-summarizer**
- **Dutch_FrameNet_Lexicon**
- **Electronisch woordenboek van de Achterhoekse en IJsselmee dialecten**
- **Electronisch woordenboek van de Gelderse dialecten**
- **Electronisch woordenboek van de Gelderse dialecten**
- **Electronisch woordenboek van de Limburgsche dialecten**
- **FLAT**
 - FoLA-Linguistic-Annotation-Tool
 - foliadicserve
- **FoLiA**
 - folia

Here you find all tools (i.e. software and software services) developed in the CLARIAH project, as well as some tools from predecessors and sister projects. Our tools are designed for researchers and developers in the Humanities and Social Sciences. Not all tools are suitable for all audiences and not all tools are mature and stable, this information should be clearly indicated for each tool, so you can make an informed judgement whether a tool might be suitable for you.

This list is automatically harvested from the tool producers and providers themselves, and updated daily.

Are you a CLARIAH developer and is your tool not included in the index yet or do you have questions or comments on the metadata? Please read our [contribution guidelines](#)

Alpino

Command-line Application

Alpino 0.0.0
Gertjan Van Noord
Computationele Taalkunde, Faculteit der Letteren, Rijksuniversiteit Groningen , Groningen University

Alpino parser and related tools for Dutch [view more]

Linguistics nwo-ComputationalLinguisticsandPhilology Software for humanities

Structural Analysis

Docker Linux

Alpino 0.0.0 ★

repo status Active

Web Application

Alpino-Webservice 2.4
Maarten van Gompel
KNAW Humanities Cluster & CLST, Radboud University

Alpino is a dependency parser for Dutch, developed in the context of the PIONIER Project Algorithms for Linguistic Processing, developed by Gertjan van Noord at the University of Groningen. This is the webservice for it. You can upload either tokenised or untokenised files (which will be automatically tokenised for you using ucto), the output will consist of a zip file containing XML files, one for each sentence in the input document. [view more]

Internet > WWW/HTTP > WSGI > Application Text Processing > Linguistic

dependency parsing folia linguistics nlp syntax

Bsd Linux Macos Python

Alpino-Webservice 2.4 ★

repo status Active

Go to Alpino Webservice (WebApplication)
<https://alpinowebservices.clariflow.nl/api/>

Alpino is a dependency parser for Dutch, developed in the context of the PIONIER Project Algorithms for Linguistic Processing, developed by Gertjan van Noord at the University of Groningen. You can upload either tokenised or untokenised files (which will be automatically tokenised for you using ucto), the output will consist of a zip file containing XML files, one for each sentence in the input document.

Created: 2015-09-08 Modified: 2023-11-01

Figure 2: Our own catalogue frontend

Publishing in catalogues (2)



All tools Services only Table view SPARQL

GaLAHaD

GaLAHaD (Generating Linguistic Annotations for Historical Dutch) allows linguists to compare taggers, tag their own corpora, evaluate the results and export their tagged documents.

Provided tools & services

GaLAHaD

Go to the WebApplication
<https://portal.clarin.nl/vdnt.org/galahad>

Type: Web Application
Service Provider: Instituut voor de Nederlandse taal

GaLAHaD API

Note: No URL was registered for this service (yet)

Type: Web API
Documentation: <https://portal.clarin.nl/vdnt.org/galahad/api/swagger-ui/index.html>
Service Provider: Instituut voor de Nederlandse taal
Input data:
Type: TextDigitalDocument
Encoding Format: <https://github.com/newsreader/NAF>

Type: TextDigitalDocument
Encoding Format: <https://universaldependencies.org/format.html>

Type: TextDigitalDocument
Encoding Format: application/tei+xml

Metadata Properties

Version

1.2.2 [\(release notes\)](#)

Interface types

Server Application Software Image Web API Web Application

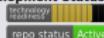
Source code repository

<https://github.com/INL/galahad> stars 1

Category

Analyzing Annotating Artificial intelligence, export systems Comparing Computational linguistics and philology Converting Enriching Lemmatizing Linguistics Machine Learning Merging POS-Tagging Software for humanities Tagging Textual and linguistic corpora

Development Status



repo status Active

Issue Tracker (Support)

<https://github.com/INL/galahad/issues> issues 1 open issues 1 closed

Documentation

README
 GaLAHaD Help

License

Apache License 2

Author(s)

Vincent Prins
 Tim Brouwer

Maintainer(s)

Vincent Prins

Contributor(s)

Vincent Prins

Producer

Instituut voor de Nederlandse taal

Programming Language

Javascript
Kotlin
TypeScript

Continuous Integration Tests

<https://github.com/INL/galahad/actions>

Runtime Platform

JVM
Node

Operating System

Linux

Software dependencies

js-yaml vite json-loader @typescript-eslint/parser content-disposition mutationobserver-shim @vue/eslint-config-typescript node-sass

Figure 3: Our own catalogue frontend

Publishing in catalogues (3)



Resource finder Types ▾ Projects Labels ▾

About

Tools

GaLAHaD

GaLAHaD (Generating Linguistic Annotations for Historical Dutch) allows linguists to compare taggers, tag their own corpora, evaluate the results and export their tagged documents.

Go to resource ↗

GaLAHaD

GaLAHaD (Generating Linguistic Annotations for Historical Dutch) allows linguists to compare taggers, tag their own corpora, evaluate the results and export their tagged documents.

Research activities

Analyzing Annotating Comparing Converting
Enriching Lemmatizing Machine Learning Merging
POS-Tagging Tagging

Research domains

Artificial intelligence, export systems
Computational linguistics and philology Linguistics
Software for humanities Textual and linguistic corpora

Access

Open Access

Version

1.0.1

Status

Active

Programming language

Javascript
Kotlin

Figure 4: CLARIAH's INEO frontend

Challenges

- ▶ Consolidating distinct vocabularies
- ▶ Extra domain-specific vocabulary
 - ▶ e.g. TaDiRaH for research activities
 - ▶ NWO Research Domains
 - ▶ Technology Readiness Level
- ▶ Validation & Curation

Validation & Curation (1)

- ▶ How to allow curation/collaboration?
 - ▶ Forges like GitHub have solid contributor mechanisms in-place
- ▶ How to assure metadata quality?
 - ▶ Harvested metadata is evaluated against a schema (SHACL) that tests metadata fields (completeness, accuracy).
 - ▶ The schema is made on the basis of carefully formulated *software metadata requirements*
 - ▶ The result of the evaluation is an automated compliance report and a compliance score on a 0-5 scale.
 - ▶ The report and score itself become part of the metadata.
 - ▶ Set a threshold for propagation to catalogues

Validation & Curation (2): Metadata Requirements

CLARIAH Software Metadata Requirements and Instructions

Introduction

This document specifies requirements and instructions for software metadata for CLARIAH software and services. It is aimed at the developers of software and attempts to provide clear documentation.

The software metadata from all of the sources within CLARIAH is harvested automatically, and periodically, by [a harvesting tool](#). Subsequently, the harvested metadata (codemeta) is made available in the tool store, which offers an API that is in turn used by portals to present the tools.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#).

Source Code Metadata Requirements

For all of the mentioned properties it in this section it holds that you **MUST NOT** use other vocabularies to exclusively express these properties.

1. Software metadata **MUST** be kept as close to the source code as possible

The main principle underlying the way we keep our software metadata is that **metadata is kept at the source and by the maintainers of the software**. This means that the tool developers themselves are responsible for providing accurate metadata and that the best place to keep this metadata is *with the source code* of the software, in a version-controlled repository.

This is in line with current best-practices as most programming language ecosystems already have their own ways of describing software metadata alongside the source code. Further details on these best-practices and the format of the metadata is expressed in other points below.

This also entails that if a third party wants to correct/revise the metadata, this is done via the tool developers, typically a third part can simply do a pull/merge request on the source repository.

2. All tools **MUST** be registered in the Tool Store Registry

Tools **MUST** be *registered* with the harvester via the [Tool Source Registry](#). This simple registry tells the harvester where to find, for each tool, both the source code repository *and* any service endpoint(s) where the software is hosted as a service (optional). In order to register your tool, you issue a simple pull request on our git repository as explained in [these contribution guidelines](#).

Validation & Curation (3): Validation report

Name
Automatic software metadata validation report for hypodisc 0.1.0

Author
codemetary validator using software.ttl

Date
2024-10-07 03:09:26

Review

Please consult the CLARIAH Software Metadata Requirements at <https://github.com/CLARIAH/clariah-plus/blob/main/requirements/software-metadata-requirements.md> for an in-depth explanation of any found problems

Validation of hypodisc 0.1.0 was successful (score=3/5), but there are some warnings which should be addressed:

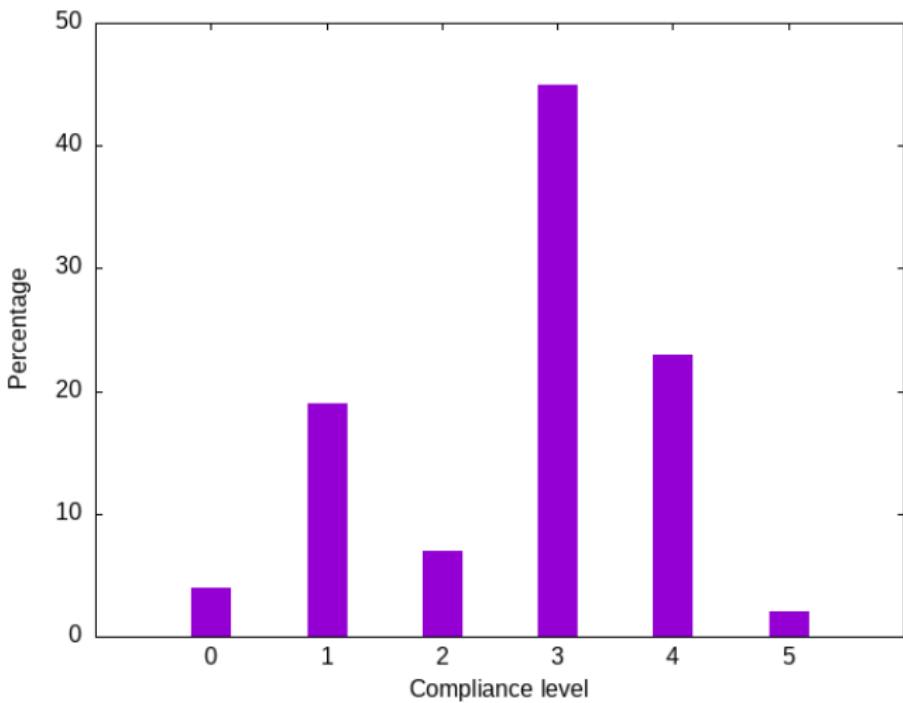
1. Info: Software source code *SHOULD* link to a continuous integration service that builds the software and runs the software's tests (This is missing in the metadata)
2. Info: An interface type *SHOULD* be expressed: Software source code should define one or more target products that are the resulting software applications offering specific interfaces (The metadata does express this currently, but something is wrong in the way it is expressed. Is the type/class valid?)
3. Warning: Documentation *SHOULD* be expressed (This is missing in the metadata)
4. Info: Reference publications *SHOULD* be expressed, if any (This is missing in the metadata)
5. Info: The funder *SHOULD* be acknowledged (This is missing in the metadata)
6. Info: The technology readiness level *SHOULD* be expressed (This is missing in the metadata)

Rating
★ ★ ★ ☆

Figure 6: An automatically generated validation report

Validation & Curation (4): Findings

- ▶ Human compliance is the biggest hurdle
- ▶ Hard to get developers to provide extra metadata
- ▶ Metadata compliance ranking in CLARIAH (114 tools):



Questions?