Software Provenance and its implementation in LaMachine

Maarten van Gompel

Centre for Language and Speech Technology, Radboud University Nijmegen

Software Provenance

Data Provenance

- **Data Provenance** describes the path data has taken through various layers of tooling
- This presentation is not about data provenance as such, but software provenance is a necessary complement for proper data provenance
  - E.g: data provenance should make reference to the **used version of each tool** to be meaningful

Software provenance aka Software Metadata

- **Software provenance** encompasses the origin of software and its licensing terms *[cit. wikipedia]*
  - Where is the source code? How to obtain it?
  - What is the license?
  - Who wrote the software? *(developers, institution, etc)*
  - What systems does the software run on? *(operating system, language ecosystem)*
  - What other software does the tool depend on? *(dependencies)*
  - What kind of interfaces does the software provide? *(command line, web-app, GUI, REST API, programming library, etc..)*
  - What is the current version/release of the software?

Version information in Software Metadata

- Accurate **software version information** is needed to:
  - **record** version information in a **data provenance chain**
  - **reinstall** the software version for scientific reproducibility
- How to get version information?
  - Differs greatly per tool
  - A common metadata approach is needed
- Software tools do not exist in isolation but live in a context:
  - Accurate **dependency** information should be available in order to:
    - **record** dependency versions in the provenance chain
    - allows **fingerprinting** of the entire dependency tree

The importance of recording dependencies

Example:

- NLP tool X relies on computation library Y.
  - An experiment was conducted with tool X v1 and (dynamically) linked to library Y v1
  - The library gets updated to v2 to fix a bug (retains API/ABI compatibility with v1)
  - Tool X v1 and library Y v2 no longer yields the same results as in the experiment

CodeMeta as a Software Metadata scheme

With codemeta, we want to formalize the schema used to map between the different services (GitHub, figshare, Zenodo) to help others plug into existing systems. Having a standard software metadata interoperability schema will allow other data archivers and libraries join in. This will help keep science on the web shareable and interoperable! [from https://codemeta.github.io]

Codemeta:

- is simple and minimalistic
- aimed at scientific software and enabling citability (DOI)
- is Linked Open Data
  - serialises to JSON-LD
  - collaborates with schema.org
- is an existing effort, grew out of "Code as a Research Object", a Mozilla Science project with Github and Figshare
  - provides a mapping to other systems (DOAP, Debian Packages, DataCite, WikiData, Maven, NodeJS, Python distutils, R, Ruby gems)

Metadating software with CodeMeta

Principles:

- provide metadata **as close to the source as possible**
  - ideally *WITH* the source by providing a `codemeta.json` in the source code repository itself (under proper version control)
  - *why?* Ensures there is less chance of the two going out of sync
- prevent duplication, auto-generate from metadata already present in an established scheme in the language's ecosystem:
  - `codemetapy`: Generate CodeMeta for Python Packages (Python Distutils/`pip`/Python Package Index)
    - https://github.com/proycon/codemetapy
  - `codemetar`: Generate CodeMeta for R Packages
    - https://github.com/ropensci/codemetar

Limits

- CodeMeta describes software metadata, not APIs
  - in contrast to: OpenAPI/Swagger, CLAM

Software Provenance in LaMachine

What is LaMachine?

- A **software (meta) distribution** for **open-source NLP software**
  - installation and configuration recipes for software *(Ansible)*
  - especially useful in case of *complex inter-dependent* software setups
  - facilitates installation on a variety of platforms
  - various **flavours**: *Virtual Machine*, *Docker container*, *local environment*, *remote provisioning*
- A kind of **Virtual Research Environment** in its own right
  - initially geared towards more tech savvy researchers, aka “the 20%”
  - **But:** also includes webservices and web applications
    - webserver with simple **portal** application
    - software configured out of the box
    - web-based scripting environment *(Jupyter Lab)*

Target and audience

- For **data scientists, developers, hosting providers** (e.g. CLARIAH centres)
- Supports several major **Linux** distributions *(Debian/Ubuntu, RedHat/CentOS, Arch)*
- Also support for Mac OS X (to a more limited degree)
- Windows users can use the VM or the Windows Linux Subsystem

Software Metadata in LaMachine

During installating/bootstrapping, LaMachine:

- Takes the software metadata from each tool's source repository if available
- otherwise: converts metadata from the upstream source *(Python Package Index, CRAN, CPAN, Maven Central)*
- Augments the metadata where needed with installation specific information:
  - to register web-based entrypoints as provided by LaMachine
  - with extra information specified in the (Ansible) build recipes
- Builds a software registry of all installed software *(JSON-LD graph)*
- Provides a portal web-application on the basis of this metadata *(Labirinto)*
  - Example: https://webservices-lst.science.ru.nl

Reproducibility

- An installation manifest and version overview can be extracted from any LaMachine installation, allowing **reconstruction** of such a LaMachine environment for scientific reproducibility
  - Within certain limits, this is not an archiving solution
  - For full reproducibility, just archive the exact VM/container image

Back to data provenance

- Other applications (like the CLARIAH WP3 VRE) can leverage the software metadata registry of a LaMachine installation to obtain software information necessary for provenance logging.

Links

- **CodeMeta:** `https://codemeta.github.io`
- **LaMachine:** `https://proycon.github.io/LaMachine`