ExtractIng - Automated metadata extraction for computational engineering applications and high-performance computing

Björn Schematic (schematic@hlrs.de)
RDA Research Data Management in Engineering IG
Seminar Series Exploring Annotation and Metadata Initiatives for Engineering Data
September 15th, 2020

The slides are licensed under CC BY-NC-SA 4.0
Outline

Introduction

ExtractIng - Automated metadata extraction
  Introduction to ExtractIng
  Extractability of metadata information
  Architecture and Implementation
  Configuration
  Running

Evaluation

Conclusion and Future Work
Introduction
Introduction

- (Explicit) Metadata is a main contributor to FAIR data management
- However metadata annotation is a burden
- Low incentives due to low scientific recognition in computational engineering
- Manual metadata tagging is bothersome

Figure: FAIR data principles in a nutshell (http://www.openaire.eu)
ExtractIng - Automated metadata extraction
Introduction to ExtractIng

Use Case
- High-Performance Computing
- Engineering Applications, in particular
  - Thermodynamics
  - Aerodynamics

Role of the metadata model EngMeta
- Serves as a convention
- ExtractIng can also be seen as a use case of EngMeta

Figure: ExtractIng in the existing RDM ecosystem
Introduction to ExtractIng

Some metadata is already available

▶ Explicit and implicit file attributes
▶ Metadata in (output) files of the simulation codes, schedulers, ...
  ▶ In standardized file formats such as HDF5 or NetCDF
  ▶ In non-standardized file formats
  ▶ In job or log files of simulation codes (z.B. nodes, version)
▶ Lots of semi-structured metadata available

Figure: Head of a GROMACS Log file.
Metadata model EngMeta – Four metadata categories

Figure: EngMeta, with categories. https://www.izus.uni-stuttgart.de/fokus/engmeta/
Extractability of the different metadata categories

<table>
<thead>
<tr>
<th>Type of metadata</th>
<th>Extractability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical metadata</td>
<td>high, as available via file attributes</td>
</tr>
<tr>
<td>Process metadata</td>
<td>medium, as available in log-, job- or system files</td>
</tr>
<tr>
<td>Domain-specific metadata</td>
<td>medium, as available in log- or output files</td>
</tr>
<tr>
<td>Descriptive metadata</td>
<td>poor, as it’s a description from a higher level</td>
</tr>
</tbody>
</table>

Table: Extractability of the different metadata categories. It is strongly dependent on the field of science.
Approach of ExtractIng

Output and log files
/group/project/user/simulation/run/box.gro
/group/project/user/simulation/run/run.trr
/group/project/user/simulation/run/run.mdp
...

Job files
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=24:00:00
#PBS -N Protein_RUN

module purge
module load chem/gromacs/4.6.7-gnu-4.9
...

Extracting

Aggregation
file attributes, job-, log files, ...

Parsing

EngMeta XSD

Metadata according EngMeta convention
<?xml version="1.0" encoding="UTF-8" ?>
<dataset xmlns:pm="http://www.loc.gov/premis/v3" x
[..]
<environment>
  <name>bwUniCluster</name>
  <compiler>
    <name>/opt/bwhpc/common/compiler/gnu/7.1.0/bin/gcc
GNU 7.1.0</name>
    <flags>-mavx -O3 -DNDEBUG -funroll-all-loops
-fexcess-precision=fast</flags>
  </compiler>
  <nodes>10</nodes>
  <ppn>2</ppn>
  <cpu>Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz</cpu>
[..]

Figure: Architecture of the metadata extraction
ExtractIng: Implementation

- Based on Java in two variants
  - Native: Java Scanner API
  - Parallel: Spark Data Analytics Framework
- Run of ExtractIng refers to a directory
- A subdirectory `.metadata` then stores the metadata information in XML

Figure: Directory with parsed metadata and a part of the EngMeta XML file.
ExtractIng: Configuration

- Everything regarding the extraction is configured externally
- External configuration file based on the EngMeta convention
- Syntax:
  
  `<EngMetaKey>,<filename>,<searchKey>,<delimiter>,<semantics>`

**Figure**: Sample part of a configuration file for GROMACS.
ExtractIng: Running

ExtractIng uses a wrapper script to shield some preparatory steps.

Listing 1: Syntax of ExtractIng

```
./fdm.sh -c <configFile> -p <directory>"<dir1> <dir2> ..." \\
-m [scanner|spark] [-e <executorCores>]
```

Listing 2: Sample call of the metadata extraction

```
./fdm.sh -c fdm.conf -p /mnt/lustre/data/educt_hexane/300_020_080/run/ \\
-m scanner
```
Evaluation
Evaluation: ExtractIng – Adaptability

- Adaptability to other simulation codes: configuration file
  - Tested:
    - GROMACS
    - NS3D (EAS3)
    - CCSM 3.0 (NetCDF in CF-Convention)
  - The more standardized, the easier to configure
  - Strongly depended on the output of the simulation code

- Adaptability to metadata models
  - Implementation of the model as Java class
  - Can partly be automated with JAXB
## Extractable metadata from GROMACS

<table>
<thead>
<tr>
<th>Metadata key (according toEngMeta)</th>
<th>Appearance</th>
<th>search key/line</th>
</tr>
</thead>
<tbody>
<tr>
<td>processingStep.date</td>
<td>*.mdp</td>
<td>At date</td>
</tr>
<tr>
<td>controlledVariable.name</td>
<td>*.usermd</td>
<td>var1.name</td>
</tr>
<tr>
<td>controlledVariable.value</td>
<td>*.mdp</td>
<td>ref_t</td>
</tr>
<tr>
<td>controlledVariable.name</td>
<td>*.usermd</td>
<td>var2.name</td>
</tr>
<tr>
<td>controlledVariable.value</td>
<td>*.mdp</td>
<td>tcoupl</td>
</tr>
<tr>
<td>controlledVariable.name</td>
<td>*.usermd</td>
<td>var3.name</td>
</tr>
<tr>
<td>controlledVariable.value</td>
<td>*.mdp</td>
<td>ref_p</td>
</tr>
<tr>
<td>controlledVariable.name</td>
<td>*.usermd</td>
<td>var4.name</td>
</tr>
<tr>
<td>controlledVariable.value</td>
<td>*.mdp</td>
<td>pcoupl</td>
</tr>
<tr>
<td>processingStep.tool.name</td>
<td>*.log</td>
<td>GROMACS</td>
</tr>
<tr>
<td>processingStep.tool.softwareVersion</td>
<td>*.log</td>
<td>GROMACS version</td>
</tr>
<tr>
<td>processingStep.tool.operatingSystem</td>
<td>*.log</td>
<td>Build OS/arch</td>
</tr>
<tr>
<td>processingStep.executionCommand</td>
<td>*.log</td>
<td>gmx_mpi mdrun</td>
</tr>
<tr>
<td>processingStep.executionCommand</td>
<td>*.log</td>
<td>gmx_mpi grompp</td>
</tr>
<tr>
<td>processingStep.environment.compiler.name</td>
<td>*.log</td>
<td>C++ compiler</td>
</tr>
<tr>
<td>processingStep.environment.compiler.flags</td>
<td>*.log</td>
<td>C++ compiler flags</td>
</tr>
<tr>
<td>processingStep.environment.compiler.name</td>
<td>*.log</td>
<td>C compiler</td>
</tr>
<tr>
<td>processingStep.environment.compiler.flags</td>
<td>*.log</td>
<td>C compiler flags</td>
</tr>
<tr>
<td>processingStep.environment.nodes</td>
<td>*.job</td>
<td>nodes</td>
</tr>
<tr>
<td>processingStep.environment.ppn</td>
<td>*.job</td>
<td>ppn</td>
</tr>
<tr>
<td>processingStep.environment.cpu</td>
<td>*.log</td>
<td>Build CPU brand</td>
</tr>
<tr>
<td>system.grid.countX</td>
<td>*.gro</td>
<td>last line</td>
</tr>
<tr>
<td>system.grid.countY</td>
<td>*.gro</td>
<td>last line</td>
</tr>
<tr>
<td>system.grid.countZ</td>
<td>*.gro</td>
<td>last line</td>
</tr>
<tr>
<td>system.temporalResolution.numberOfTimesteps</td>
<td>*.mdp</td>
<td>nsteps</td>
</tr>
<tr>
<td>system.temporalResolution.interval</td>
<td>*.mdp</td>
<td>dt</td>
</tr>
</tbody>
</table>
## Evaluation: ExtractIng – Adaptability

<table>
<thead>
<tr>
<th>Compute Environment</th>
<th>native Scanner</th>
<th>parallel Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worskstation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ubuntu 18.04</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Windows 10</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>bwUniCluster</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RHEL 7.5</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Cray XC40</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CLE 6.0.UP05</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Cray URIKA</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Urika-GX-2.2UP00</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table: ExtractIng adaptability to compute environments*
Figure: Performance comparison of native (Scanner) and parallel (Spark) implementation. Measured on Cray URIKA.
Evaluation: Integration – Scientific Workflow

▶ Extraction can be integrated to the job script, see script:

```
#!/bin/bash
#PBS -N Aero_Simulation
#PBS -l nodes=1:ppn=24
#PBS -l walltime=00:20:00
#PBS -M schembera@hlrs.de
module load java

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

# Launch the parallel job and the metadata collection right after
aprun -n 24 -N 24 ~/promotion/aeroCode > my_output_file 2>&1
~/harvester/fdm.sh ~/harvester/fdm_iag_eval_conf.scanner
```

▶ Then, data + metadata can be pushed to a repository, such as DaRUS.
Conclusion and Future Work
Conclusion and Future Work

Conclusion and Findings
▶ Metadata annotation as a burden, however as a key to FAIR data
▶ ExtractIng tries improve the situation by automated extraction
▶ It is designed not to alter the specific scientific workflow
▶ ExtractIng is available on https://github.com/bjschembera/ExtractIng
▶ This is a proof-of-concept implementation, lots of improvements to be done...
▶ The project provided lots of findings regarding usage and extractability of metadata

Limitations and Future Work
▶ Limited to extraction of `<key>` `<delimiter>` `<value>` patterns
▶ Extraction of unstructured data is not possible
▶ Hierarchical information is hard to extract
▶ Extraction function is currently limited to lines
References


https://www.ub.uni-stuttgart.de/forschen-publizieren/forschungsdatenmanagement/projekte/dipl_ing/materials/metadata/index.html


