Christoph Böhme

Analysis of library metadata with Metafacture
Agenda

13:00 — a short introduction to Metafacture
13:30 — warm-up exercises
14:30 — triples and counting
15:00 — exercises on counting data
   (incl. 30 min coffee break at 15:30)
17:00 — joining data sets and analysing them
17:30 — exercises on joining data
18:50 — wrapping up
Part 1

A short introduction to Metafacture
Overview of Metafacture

Stream modules

- DSL* for constructing processing flows
- Building blocks for processing flows

Flux

Metamorph

Stream module with a DSL* for metadata transformation

*DSL: Domain specific Language
The basic building block of Metafacture

Receives typed input:
• strings
• triples
• objects
• metadata events

Sends typed output:
• strings
• triples
• objects
• metadata events

Processes input to create some output. Modules usually perform rather small tasks to foster reusability.
A simple processing flow

Read and print a file containing pica records:

- `open-file` as `file handle`
- `as-lines` as `string` metadata events
- `decode-pica` metadata events
- `encode-formeta` string
- `write("stdout")`

String

- `file name`
- `file handle`
Module configuration

- either a single mandatory value
- or optional key-value pairs

Stream module
Describing flows with Flux

```
"file.name"
|open-file
|as-lines
|decode-pica
|encode-formeta(style="multiline")
|write("stdout");
```

- A string as the initial input
- Modules are connected with a pipe character
- Key-value based configuration
- Mandatory parameter
- Flow ends with a semi-colon
Variables and comments in Flux

Define default values for the variables \texttt{in} and \texttt{out}

\begin{verbatim}
default in = "file.name";
default out = "stdout";
\end{verbatim}

Comments start with two slashes

Use variable instead of directly entering a string

\begin{verbatim}
in |open-file
// ... 
|write(out);
\end{verbatim}
Running Flux scripts

- Flux script must be selected in the IDE
- Choose “Run with Flux” to execute the selected Flux script
- “Flux Help” outputs a list of all supported modules
Representation of metadata in Metafacture: a stream of events

Pica record
003@
  $0 2809
033A
  $n Publisher
  $p Location

Sequence of metadata events
- Start record 2809
- Start entity 003@
- Literal 0: 2809
- End entity
- Start entity 033A
- Literal n: Publisher
- Literal p: Location
- End entity
- End record
Processing metadata events with Metamorph

- Start record id
- Start entity 021A
- Literal a: *The Trial*
- End entity
- End record

Listen for 021A.a
Output as Title

- Start record id
- Literal Title: *The Trial*
- End record
Metamorph: data statements

```xml
<?xml version="1.0" encoding="UTF-8"?>

<metamorph xmlns="http://www.culturegraph.org/metamorph"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1" entityMarker=".">
  <rules>
    <data source="021A.a" name="Title" />
  </rules>
</metamorph>
```

Separator for entities and literal names

Name of the literal to listen for

Name of the literal that is output
Metamorph: modifying data

...<rules>

  <data source="021A.a" name="Title">
    <regexp match="^(The) (.*)" format="${2}, ${1}" />
  </data>

</rules>

...Process the data value before outputting it. You can specify multiple functions here
Metamorph: combining data

...<rules>

<combine name="Publisher" value="${Pub}: ${Loc}" >
    <data source="033A.n" name="Pub" />
    <data source="033A.p" name="Loc" />
</combine>

</rules>

...

Name of the generated literal. It can include variables, too

Literal value constructed from the variables from the data statements below

The data statements do not generate output but create variables instead
Exercises part 1

Warm-up
Part 2

Triples and counting
The triple

Inspired by RDF triples but subject and predicate do not need to be URIs
Generating triples

Metadata events → stream-to-triples → Triples

- Literals on top level:
  - Start record id
  - Literal name: Klaus
  - Start entity died
  - Literal when: 1401
  - Literal where: HH
  - End entity
  - End record

- Entities on top level:
  - record-id
  - name: Klaus
  - died
  - where: HH

Serialised with Formeta
Counting triples

count-triples (countBy="object")

| count | 4 |
| count | 2 |
| count | 3 |
Outputting triples

| red | count | 4 |

```
template ("${o} times ${s}"")
```

"4 times red"

Use ${s}, ${p} and ${o} as placeholders for subject, predicate and object
Counting data values

- **morph**
  - Extracts data items that should be counted and outputs them as top level literals

- **stream-to-triples**
  - Converts the literals to triples

- **count-triples**
  - Counts the different object values of the triples

- **template**
  - Optionally: converts triples into formatted text

---

| 42 | Analysis of library metadata with Metafacture | SWIB 2013 | 25 November 2013 |
Counting data values: flow of data

- Start record id
- Start entity 033A
- Literal p: Hamburg
- End entity
- End record

Morph

- Start record id
- Literal loc: Hamburg
- End record

id loc Hamburg

Count

Hamburg count 1
**Metamorph: choosing data**

```xml
...<rules>
  <choose name="Location">
    <data source="033A.p">
      <regexp match="^Ffm$" format="Frankfurt a. M." />
    </data>
    <data source="033A.p" />
  </choose>
</rules>
...```

Only the value of the topmost data-statement that generates output is returned by the choose-statement.
Metamorph: generating constant values

...<rules>
  <data source="021A.a" name="Title">
    <constant value="All books have the same name" />
  </data>
</rules>
...

No matter what the value of literal 021A.a is, always output the defined value.
Exercises part 2

Triples and counting
Part 3

Joining data sets and analysing them
Joining streams of data

How is this done?
Converting triples into records

Triples → collect-triples → Metadata events

Sequences of triples with the same subject are merged.

Record 2
- PA: OA
- PB: OB

Record 1
- PC: OC
- PD: OD

Record 2
- PE: OE
- PF { ... }

Serialised entities are deserialised into entities.
Sorting triples
Linking streams in Flux with wormholes

"file1"
|open-file
// ...
|stream-to-triples
|@x;

"file2"
|open-file
// ...
|stream-to-triples
|@x;

@x
|wait-for-inputs("2")
|sort-triples
|collect-triples
|encode-formeta
|write("stdout");

Sends the triples into a “wormhole”

Receives triples from a “wormhole”

These three flows must be defined in the same Flux script
Advanced triplification: ID redirection

Metadata events → stream-to-triples (redirect="true") → Triples

- Start record id
- Literal _id: new id
- Literal name: Klaus

Replaces record id in subjects

...
Using _id-redirection

X

id: gnd1
id: gnd2
id: gnd3
id: HH
gnd: 3
id: Ffm
gnd: 1
id: Ks
gnd: 2

Start record HH
Literal gnd: 3
Literal country: D
End record

Morph

Start record HH
Literal _id: gnd3
Literal country: D
Literal wiki-id: HH
End record

X

Start record HH
Literal gnd: 3
Literal country: D
Literal wiki-id: HH
End record
Using `{to:ID}`-redirection

Finding the backlinks: Who links to this record?

```
- Start record gnd1
- Literal loc: HH
- End record

Morph

- Start record gnd1
- Literal `{to:HH}ref: gnd1`
- End record

HH
```

```
HH
ref
```
Putting the pieces together

1. **morph**
   - Prepares records to generate the right subject values

2. **stream-to-triples**
   - Conversion with id redirection
   - Joins the two streams of triples using a “wormhole”
   - Ensures that triples with the same subject form a sequence

3. **wait-for-inputs("2")**
   - Waits for all flows writing triples into the “wormhole”

4. **sort-triples**
   - Turns the triples back into metadata events

5. **collect-triples**
Metamorph: what else?

...<br/> <rules><br/> <data source="_id" name="recordId" />&gt;<br/> <data source="033A.n" name="Publisher" />&gt;<br/> <data source="_else" />&gt;<br/> </rules> ...
Exercises part 3

Joining data sets and analysing them
Wrapping up
What did we learn today?

- Foundations of processing metadata with Flux and Metamorph
- Exploring data sets by quantifying data values
- Joining data sets and analysing their relations
- Typical patterns for analysing data with Metafacture

These patterns are similar to the way Hadoop operates: This makes migration from your desktop to a Hadoop cluster easy
Metafacture

- Not only designed for data analysis but for metadata processing in general
- Software tool and library: It can easily be integrated into other applications
- Flux and Metamorph are extendable
- It is open source at [http://culturegraph.github.io/](http://culturegraph.github.io/)
Job advert

We are looking for a software developer for our solr-based search engine infrastructure

For more information please visit:
http://www.dnb.de/stellen
Thank you very much!

Further questions?
Contact me at c.boehme@dnb.de
or join the mailing list:
http://lists.dnb.de/mailman/listinfo/metafacture