

Datentransformation mit Metafacture – Aktuelle Entwicklungen

Adrian Pohl & Fabian Steeg

Offene Infrastruktur, Hochschulbibliothekszentrum NRW (hbz)

KIM Workshop,
WWW, 2. Mai 2022

<https://slides.lobid.org/2022-05-02-kim-ws/>



Agenda

1. Kontext: Datentransformation
2. Was ist Metafacture?
3. Metafacture Fix
4. Metafacture Playground
5. Ausblick
6. Fragen & Diskussion

1. Kontext: Datentransformation

Anwendungsszenarien

Anwendungsszenarien

Datenanalyse, z.B. Feldabdeckung im Katalog

Anwendungsszenarien

Datenanalyse, z.B. Feldabdeckung im Katalog

Datenaufbereitung, z.B. zur Visualisierung mit Kibana

Anwendungsszenarien

Datenanalyse, z.B. Feldabdeckung im Katalog

Datenaufbereitung, z.B. zur Visualisierung mit Kibana

Datenanreicherung, z.B. Ergänzung von Daten aus Wikidata

Anwendungsszenarien

Datenanalyse, z.B. Feldabdeckung im Katalog

Datenaufbereitung, z.B. zur Visualisierung mit Kibana

Datenanreicherung, z.B. Ergänzung von Daten aus Wikidata

Datenaggregation aus unterschiedlichen Quellen, z.B. **OERSI**

Anwendungsszenarien

Datenanalyse, z.B. Feldabdeckung im Katalog

Datenaufbereitung, z.B. zur Visualisierung mit Kibana

Datenanreicherung, z.B. Ergänzung von Daten aus Wikidata

Datenaggregation aus unterschiedlichen Quellen, z.B. **OERSI**

Systemmigration, z.B. nach Alma oder Folio

Annahmen

Annahmen

Transformationen von Metadaten gehören zum täglichen Geschäft wissenschaftlicher Bibliotheken.

Annahmen

Transformationen von Metadaten gehören zum täglichen Geschäft wissenschaftlicher Bibliotheken.

Es gibt viele unterschiedliche Methoden, die meist Programmierkenntnisse voraussetzen.

Annahmen

Transformationen von Metadaten gehören zum täglichen Geschäft wissenschaftlicher Bibliotheken.

Es gibt viele unterschiedliche Methoden, die meist Programmierkenntnisse voraussetzen.

Datentransformationen werden meist im Zusammenspiel von Fachabteilungen & IT umgesetzt, verbunden mit größerem Kommunikationsaufwand.

Annahmen

Transformationen von Metadaten gehören zum täglichen Geschäft wissenschaftlicher Bibliotheken.

Es gibt viele unterschiedliche Methoden, die meist Programmierkenntnisse voraussetzen.

Datentransformationen werden meist im Zusammenspiel von Fachabteilungen & IT umgesetzt, verbunden mit größerem Kommunikationsaufwand.

Bereits existierende, von anderen entwickelte Transformationsprozesse können nur bedingt entdeckt und nachgenutzt werden.

Das heißt:

Das heißt:

Es gibt großes Potential, eine immer wiederkehrende Arbeit zugänglicher, kollaborativer und effizienter zu gestalten.

Übergeordnete Ziele

Übergeordnete Ziele

Ermächtigung der Fachebene zur Konfiguration von
Datentransformationen

Übergeordnete Ziele

Ermächtigung der Fachebene zur Konfiguration von
Datentransformationen

Förderung von Praktiken zum Teilen und Auffinden von
Transformationsprozessen

2. Was ist Metafacture?

Was ist Metafacture?

Was ist Metafacture?

[...] zur Verarbeitung von semi-strukturierten Daten mit dem Fokus auf Bibliotheksdaten

Was ist Metafacture?

[...] zur Verarbeitung von semi-strukturierten Daten mit dem Fokus auf Bibliotheksdaten

Ein vielseitiges Werkzeug: nutzbar als Kommandozeilentool, als Java/JVM library, für Batch-Verarbeitung oder on-the-fly

Was ist Metafacture?

[...] zur Verarbeitung von semi-strukturierten Daten mit dem Fokus auf Bibliotheksdaten

Ein vielseitiges Werkzeug: nutzbar als Kommandozeilentool, als Java/JVM library, für Batch-Verarbeitung oder on-the-fly

Ein offenes Framework: Wiederverwendung und Austausch (von einzelnen Modulen und ganzen Workflows)

Metafactory-Historie

Metafactory-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

Metafactory-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

2013: Umzug auf GitHub, Open-Source-Projekt geworden

Metafactory-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

2013: Umzug auf GitHub, Open-Source-Projekt geworden

2019: Mit der Zeit immer weniger DNB-Ressourcen für Metafactory, hbz wird Maintainer

Metafactory-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

2013: Umzug auf GitHub, Open-Source-Projekt geworden

2019: Mit der Zeit immer weniger DNB-Ressourcen für Metafactory, hbz wird Maintainer

2019: Start von Metafactory Fix

Metafactory-Historie

2011: Start der Entwicklung durch DNB im Rahmen von Culturegraph; damals schon Austausch mit dem hbz

2013: Umzug auf GitHub, Open-Source-Projekt geworden

2019: Mit der Zeit immer weniger DNB-Ressourcen für Metafactory, hbz wird Maintainer

2019: Start von Metafactory Fix

2021: Start von Metafactory Playground

Wie Metafactory funktioniert

Wie Metafactory funktioniert

Grundidee: Daten fließen durch mehrere Module:
→ read → decode → transform → encode → write →

Wie Metafactory funktioniert

Grundidee: Daten fließen durch mehrere Module:
→ read → decode → transform → encode → write →

Jedes Modul erwartet Input eines bestimmten Typs und erzeugt Output eines bestimmten Typs

Wie Metafacture funktioniert

Grundidee: Daten fließen durch mehrere Module:

→ read → decode → transform → encode → write →

Jedes Modul erwartet Input eines bestimmten Typs und erzeugt Output eines bestimmten Typs

Verschiedene Formate werden unterstützt (z.B. PICA, MARC), erweiterbares Framework für eigene Formate

Wie Metafacture funktioniert

Grundidee: Daten fließen durch mehrere Module:

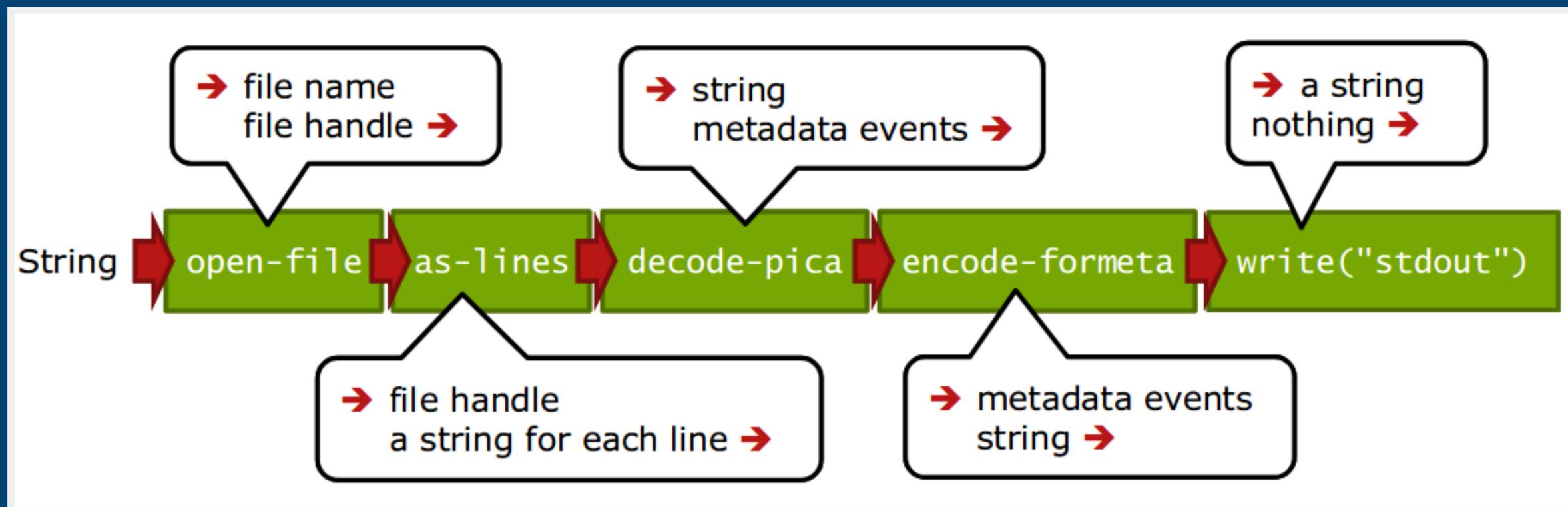
→ read → decode → transform → encode → write →

Jedes Modul erwartet Input eines bestimmten Typs und erzeugt Output eines bestimmten Typs

Verschiedene Formate werden unterstützt (z.B. PICA, MARC), erweiterbares Framework für eigene Formate

Durch Kombination einzelner Module, durch die unsere Daten fließen, bauen wir den Workflow

Ein Workflow



Aus: Christoph Böhme, http://swib.org/swib13/slides/boehme_swib13_131.pdf

Workflows konfigurieren und ausführen

Workflows konfigurieren und ausführen

Workflows können in Flux (einer speziellen Konfigurationssprache) oder mit Java (typischer über Java Generics) bearbeitet werden

Workflows konfigurieren und ausführen

Workflows können in Flux (einer speziellen Konfigurationssprache) oder mit Java (typischer über Java Generics) bearbeitet werden

Flux-Workflows können in einem Texteditor editiert und auf der Kommandozeile ausgeführt werden, Java-Workflows funktionieren wie andere Java-Komponenten

Workflow-Beispiel

Ausgabe des Inhalts von `bib-data.pica.gz` als Formeta

```
$ gedit sample1.flux &
```

```
"bib-data.pica.gz"  
|open-file|as-lines|decode-pica|encode-formeta|write("stdout");
```

```
$ flux.sh sample1.flux
```

<https://github.com/hbz/metafactory-flux-examples/tree/master/sample1>

Workflow-Beispiel

Ausgabe des Inhalts von `bib-data.pica.gz` als Formeta

```
$ gedit sample1.flux &
```

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

```
$ flux.sh sample1.flux
```

<https://github.com/hbz/metafactory-flux-examples/tree/master/sample1>

Workflow-Beispiel: Ausgabe

```
'482147350' {  
  '001@' {  
    'a': '5',  
    '0': '1-2'  
  },  
  '001A' {  
    '0': '1100:15-10-94'  
  },  
  '001B' {  
    '0': '9999:12-06-06',  
    't': '16:10:17.000'  
  },  
  '001D' {  
    '0': '9999:99-99-99'  
  },  
  '001H' {
```

<https://github.com/hbz/metafactory-flux-examples/tree/master/sample1>

Transformationsmodul

Transformationsmodul

Manipulation von Feldnamen und -werten; filtern, kombinieren, normalisieren etc.

Transformationsmodul

Manipulation von Feldnamen und -werten; filtern,
kombinieren, normalisieren etc.

Feldwerte aus Lookup-Tabellen in externen Dateien (z.B.
Freitextfelder -> kontrollierte Vokabulare)

Transformationsmodul

Manipulation von Feldnamen und -werten; filtern,
kombinieren, normalisieren etc.

Feldwerte aus Lookup-Tabellen in externen Dateien (z.B.
Freitextfelder -> kontrollierte Vokabulare)

Morph: XML-basiert, Feld- / Metadaten-Event-Ebene

Transformationsmodul

Manipulation von Feldnamen und -werten; filtern, kombinieren, normalisieren etc.

Feldwerte aus Lookup-Tabellen in externen Dateien (z.B. Freitextfelder -> kontrollierte Vokabulare)

Morph: XML-basiert, Feld- / Metadaten-Event-Ebene

Fix: eigene, Catmandu-Fix-artige Sprache, Record-basiert

Morph

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

Morph

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<metamorph xmlns="http://www.culturegraph.org/metamorph" xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">  
  <rules>  
    <data source="_id" name="id"/>  
    <data source="a" name="title"/>  
    <combine value="{first} {last} aus {place}" name="author">  
      <data source="b.v" name="first" />  
      <data source="b.n" name="last" />  
      <data source="c" name="place" />  
    </combine>  
  </rules>  
</metamorph>
```

Morph

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<metamorph xmlns="http://www.culturegraph.org/metamorph" xmlns:xs="http://www.w3.org/2001/XMLSchema-1.0" >  
  <rules>  
    <data source="_id" name="id" />  
    <data source="a" name="title" />  
    <combine value="{first} {last} aus {place}" name="author">  
      <data source="b.v" name="first" />  
      <data source="b.n" name="last" />  
      <data source="c" name="place" />  
    </combine>  
  </rules>  
</metamorph>
```

```
{"id": "1", "title": "Faust", "author": "JW Goethe aus Weimar"}  
{"id": "2", "title": "Räuber", "author": "F Schiller aus Weimar"}
```

Beispiel im Playground

Zwischenfazit

Zwischenfazit

Erfahrung hat gezeigt: Metafactory ist durch seine Modularität und Erweiterbarkeit sehr vielseitig einsetzbar, aber die Transformation mit Morph ist sehr komplex

Zwischenfazit

Erfahrung hat gezeigt: Metafactory ist durch seine Modularität und Erweiterbarkeit sehr vielseitig einsetzbar, aber die Transformation mit Morph ist sehr komplex

Kernproblem: Transformationsregeln beziehen sich konzeptuell auf ganze Records (z.B. wenn in Feld x das und das steht, dann mach mit Feld y das und das)

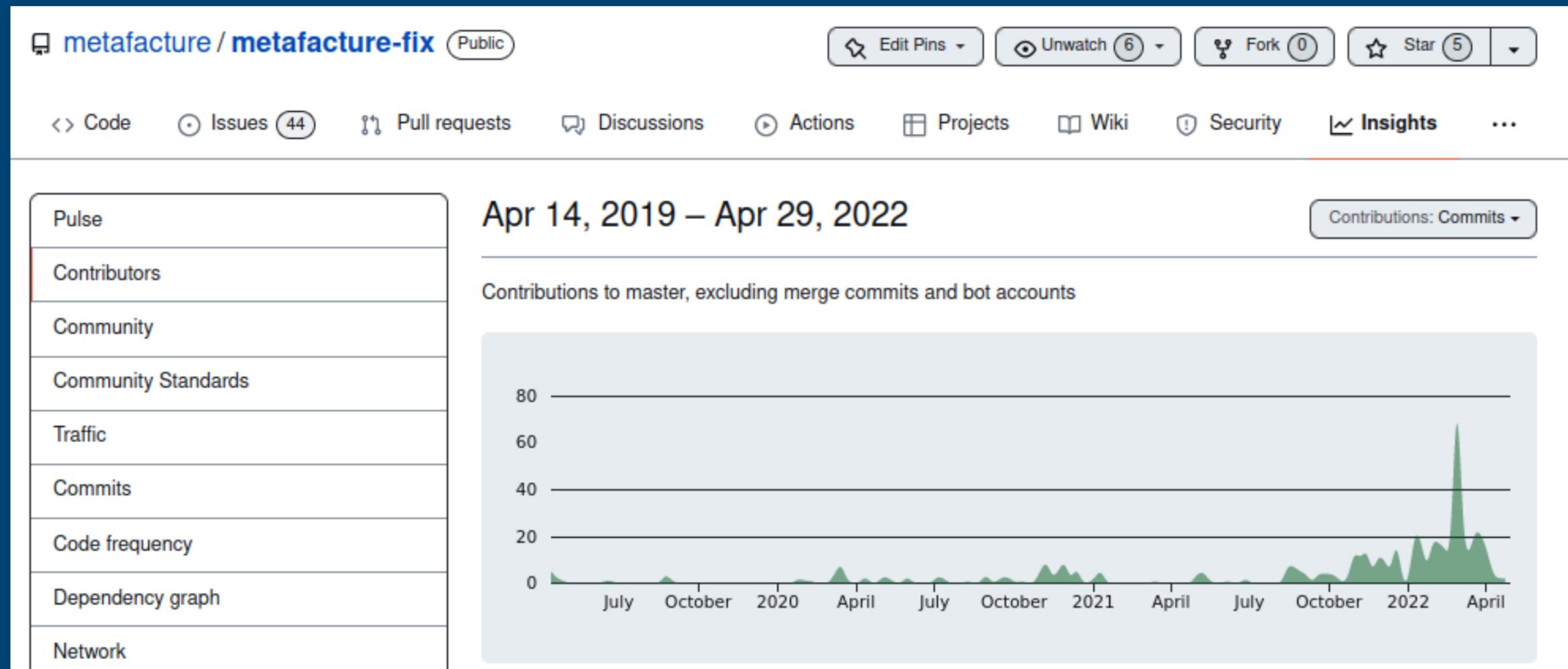
Zwischenfazit

Erfahrung hat gezeigt: Metafactory ist durch seine Modularität und Erweiterbarkeit sehr vielseitig einsetzbar, aber die Transformation mit Morph ist sehr komplex

Kernproblem: Transformationsregeln beziehen sich konzeptuell auf ganze Records (z.B. wenn in Feld x das und das steht, dann mach mit Feld y das und das)

Grundsätzlich behandelt man in Morph aber wie vorhin beschrieben quasi Feld-Events, wird sehr schnell unintuitiv

3. Metafactory Fix



Metafactory Fix: Ziele

Metafactory Fix: Ziele

Wechsel von einem Feld-basierten zu einem Record-basierten Verarbeitungsansatz bei der Transformation

Metafactory Fix: Ziele

Wechsel von einem Feld-basierten zu einem Record-basierten Verarbeitungsansatz bei der Transformation

Anknüpfung an existierende Konfigurationssprache aus

Catmandu (mittelfristiges Ziel: Standardisierung, s.
<https://github.com/elag/FIG>)

Metafactory Fix: Ziele

Wechsel von einem Feld-basierten zu einem Record-basierten Verarbeitungsansatz bei der Transformation

Anknüpfung an existierende Konfigurationssprache aus

Catmandu (mittelfristiges Ziel: Standardisierung, s. <https://github.com/elag/FIG>)

Erleichterung der Transformationskonfiguration

Metafactory Fix: Ziele

Wechsel von einem Feld-basierten zu einem Record-basierten Verarbeitungsansatz bei der Transformation

Anknüpfung an existierende Konfigurationssprache aus

Catmandu (mittelfristiges Ziel: Standardisierung, s. <https://github.com/elag/FIG>)

Erleichterung der Transformationskonfiguration

Vergrößerung der Zielgruppe um Bibliothekar:innen und andere Metadatenfachleute (bei uns z.B. in OERSI, erster Anwendungsfall und Entwicklungsbegleitung zu Fix)

Fix

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

Fix

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

```
move_field(_id, id)  
move_field(a, title)  
paste(author, b.v, b.n, '~aus', c)  
retain(id, title, author)
```

Fix

```
1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}  
2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

```
move_field(_id, id)  
move_field(a, title)  
paste(author, b.v, b.n, '~aus', c)  
retain(id, title, author)
```

```
{"id": "1", "title": "Faust", "author": "JW Goethe aus Weimar"}  
{"id": "2", "title": "Räuber", "author": "F Schiller aus Weimar"}
```

Beispiel im Playground

Aktueller Stand Fix

Aktueller Stand Fix

Beispiele: produktiv in **OERSI** (diverse Web-Quellen → JSON),
im Aufbau für **hbz-Verbundkatalog** (MARC → JSON) und
Rheinland-Pfälzische Bibliographie (Allegro → JSON)

Aktueller Stand Fix

Beispiele: produktiv in **OERSI** (diverse Web-Quellen → JSON),
im Aufbau für **hbz-Verbundkatalog** (MARC → JSON) und
Rheinland-Pfälzische Bibliographie (Allegro → JSON)

Tests: **unit** (Simulation von Input-Output-Events und Fix auf
Code-Ebene; aktuell 436 pass / 26 skip), **integration** (Input,
Flux, Fix, Output als Dateien wie bei Real-World-Setup;
aktuell 161 pass / 67 skip, d.h. aktuell 70% "conformance")

Aktueller Stand Fix

Beispiele: produktiv in **OERSI** (diverse Web-Quellen → JSON),
im Aufbau für **hbz-Verbundkatalog** (MARC → JSON) und
Rheinland-Pfälzische Bibliographie (Allegro → JSON)

Tests: **unit** (Simulation von Input-Output-Events und Fix auf
Code-Ebene; aktuell 436 pass / 26 skip), **integration** (Input,
Flux, Fix, Output als Dateien wie bei Real-World-Setup;
aktuell 161 pass / 67 skip, d.h. aktuell 70% "conformance")

d.h. vieles geht schon, aber es gibt auch noch einiges zu tun

Aktueller Stand Fix

Beispiele: produktiv in **OERSI** (diverse Web-Quellen → JSON),
im Aufbau für **hbz-Verbundkatalog** (MARC → JSON) und
Rheinland-Pfälzische Bibliographie (Allegro → JSON)

Tests: **unit** (Simulation von Input-Output-Events und Fix auf
Code-Ebene; aktuell 436 pass / 26 skip), **integration** (Input,
Flux, Fix, Output als Dateien wie bei Real-World-Setup;
aktuell 161 pass / 67 skip, d.h. aktuell 70% "conformance")

d.h. vieles geht schon, aber es gibt auch noch einiges zu tun

Aktuelle Dokumentation, GitHub-Repo, Playground

4. Metafacture Playground

Metafactory Playground

Metafactory Playground

Webbasierte Oberfläche zum Ausprobieren und
Austauschen von Workflows

Metafactory Playground

Webbasierte Oberfläche zum Ausprobieren und
Austauschen von Workflows

Ziel: Einstiegshürde für Metafactory senken, neben Morph-
Komplexität unserer Erfahrung nach ein weiteres zentrales
Problem bei der Metafactory-Nutzung

Metafactory Playground

Webbasierte Oberfläche zum Ausprobieren und
Austauschen von Workflows

Ziel: Einstiegshürde für Metafactory senken, neben Morph-
Komplexität unserer Erfahrung nach ein weiteres zentrales
Problem bei der Metafactory-Nutzung

Für Entwicklung, Dokumentation, Tutorials, Workshops

Metafactory Playground

Webbasierte Oberfläche zum Ausprobieren und
Austauschen von Workflows

Ziel: Einstiegshürde für Metafactory senken, neben Morph-
Komplexität unserer Erfahrung nach ein weiteres zentrales
Problem bei der Metafactory-Nutzung

Für Entwicklung, Dokumentation, Tutorials, Workshops

<https://metafactory.org/playground>



Metafacture Playground

beta

metafacture-framework master-SNAPSHOT

metafix 0.2.0-SNAPSHOT

Load Examples

Clear

Process

Share

Import Workflow

Export Workflow

Data

```
1 1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}
2 2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

Flux

```
1 PG_DATA
2 |as-lines
3 |decode-formeta
4 |fix
5 |encode-json
6 |print
7 ;
```

Fix

Morph

```
1 move_field(_id, id)
2 move_field(a, title)
3 paste(author, b.v, b.n, '~aus', c)
4 retain(id, title, author)
```

Result

```
{"id":"1","title":"Faust","author":"JW Goethe aus Weimar"}
{"id":"2","title":"Räuber","author":"F Schiller aus Weimar"}
```



Load Examples Clear Process Share Import Workflow Export Workflow

Data

```
1 1{a: Faust, b {n: Goethe, v: JW}, c: Weimar}
2 2{a: Räuber, b {n: Schiller, v: F}, c: Weimar}
```

Flux

```
1 PG_DATA
2 |as-lines
3 |decode-formeta
4 |fix
5 |encode-json
6 |print
7 ;
```

Fix

Morph

```
1 move_field(_id, id)
2 move_field(a, title)
3 paste(author, b.v, b.n, '~aus', c)
4 retain(id, title, author)
```

Result

No result

5. Ausblick

Community vergrößern

Community vergrößern

Metafactory hbz-intern als zentrales Tool etablieren

Community vergrößern

Metafactory hbz-intern als zentrales Tool etablieren

Weitere Nutzer:innen und Mitentwickler:innen innerhalb
und außerhalb des hbz-Verbunds gewinnen

Standards nutzen und aufbauen

Standards nutzen und aufbauen

SKOS Lookups (zum Andocken an [SkoHub Vocab](#)s)

Standards nutzen und aufbauen

SKOS Lookups (zum Andocken an [SkoHub Vocab](#)s)

Entity Reconciliation mit [OpenRefine](#)-kompatiblen Diensten

Standards nutzen und aufbauen

SKOS Lookups (zum Andocken an [SkoHub Vocab](#)s)

Entity Reconciliation mit [OpenRefine](#)-kompatiblen Diensten

Fix-Standardisierung, s. <https://github.com/elag/FIG>

ETL Hub

ETL Hub

ETL: Extract, Transform, Load

ETL Hub

ETL: Extract, Transform, Load

mehr Kollaboration, Teilen & Auffinden von Workflows
ermöglichen (nicht nur für Metafactory)

ETL Hub

ETL: Extract, Transform, Load

mehr Kollaboration, Teilen & Auffinden von Workflows
ermöglichen (nicht nur für Metafactory)

Entwicklung von Best Practices zur Paketierung und
Beschreibung von ETL-Konfigurationen

ETL Hub

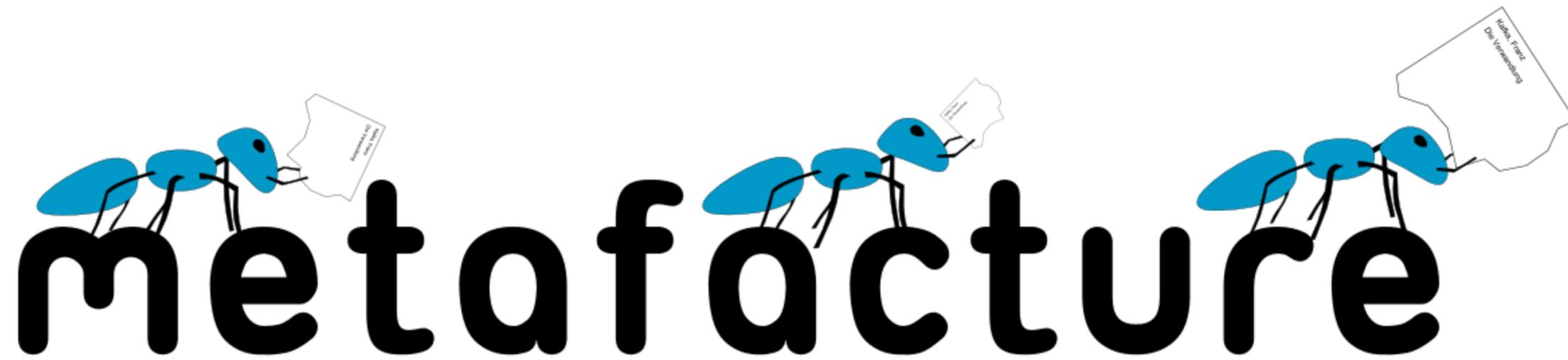
ETL: Extract, Transform, Load

mehr Kollaboration, Teilen & Auffinden von Workflows ermöglichen (nicht nur für Metafactory)

Entwicklung von Best Practices zur Paketierung und Beschreibung von ETL-Konfigurationen

Aufbau eines ETL Hubs zum Entdecken existierender ETL-Prozesse für die einfache Nachnutzung und Anpassung

6. Fragen und Diskussion



<https://metafactory.org>