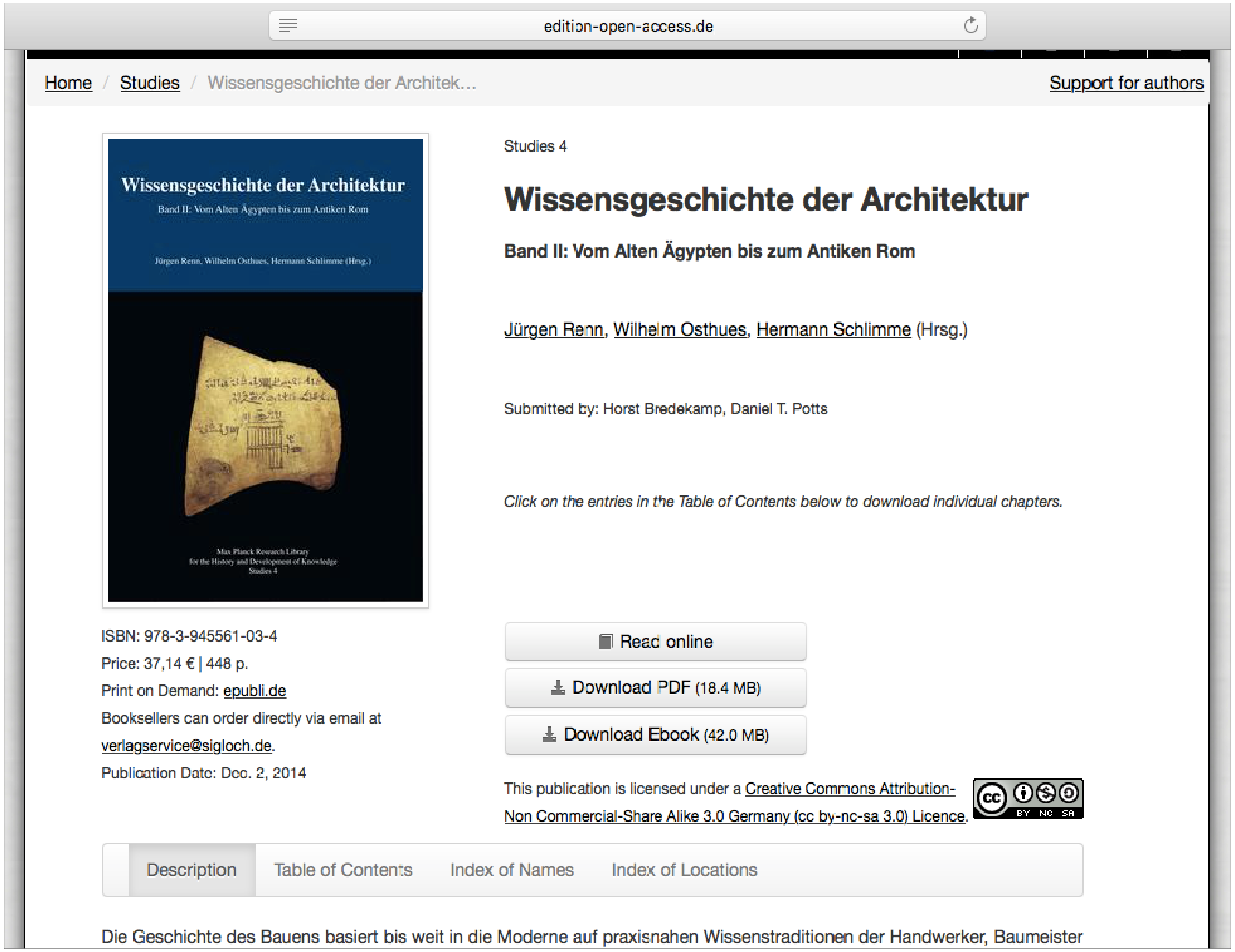
# Case Study TEI Customisation: a Restricted TEI Format for Edition Open Access

Samuel Gfrörer and Klaus Thoden

Max Planck Institute for the History of Science

TEI 2019, Graz, September 20, 2019

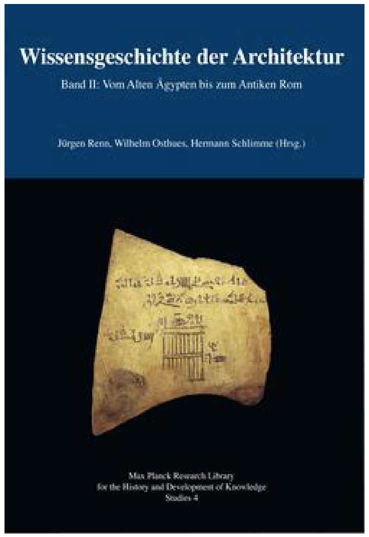
# Introduction to EOA



edition-open-access.de

Home / Studies / Wissensgeschichte der Architek…

Support for authors



Studies 4

## Wissensgeschichte der Architektur

**Band II: Vom Alten Ägypten bis zum Antiken Rom**

Jürgen Renn, Wilhelm Osthues, Hermann Schlimme (Hrsg.)

Submitted by: Horst Bredekamp, Daniel T. Potts

*Click on the entries in the Table of Contents below to download individual chapters.*

ISBN: 978-3-945561-03-4
Price: 37,14 € | 448 p.
Print on Demand: epubli.de
Booksellers can order directly via email at verlagservice@sigloch.de.
Publication Date: Dec. 2, 2014

Read online

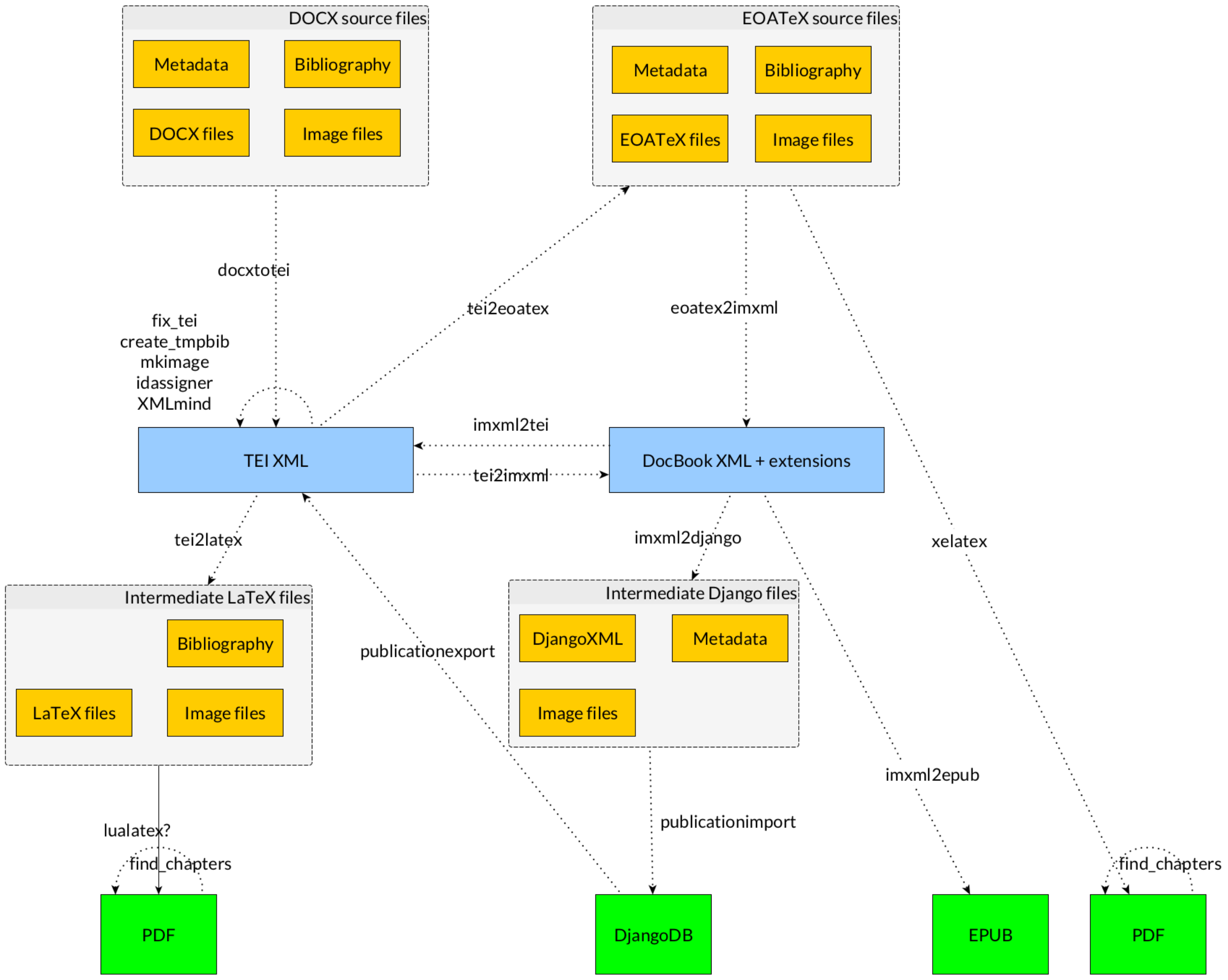
Download PDF (18.4 MB)

Download Ebook (42.0 MB)

Description | Table of Contents | Index of Names | Index of Locations

Die Geschichte des Bauens basiert bis weit in die Moderne auf praxisnahen Wissenstraditionen der Handwerker, Baumeister

# EOA – Document Processing



# 1. Approach: Relax NG Compact Schema for EOA

```
<?xml version="1.0" encoding="UTF-
8"?>
<TEI xmlns="
http://www.tei-c.org/ns/1.0">
...
  <body>
    <div n="1" type="part">
      ...(part specific)...
      <div type="chapter">
        ...(chapter specific)...
        <div type="section">
          ...(section specific)...
          <div type="subsection">
            ...
          </div>
          ...
        </div>
        ...
      </div>
      ...
    </div>
    ...
  </body>
</TEI>
```

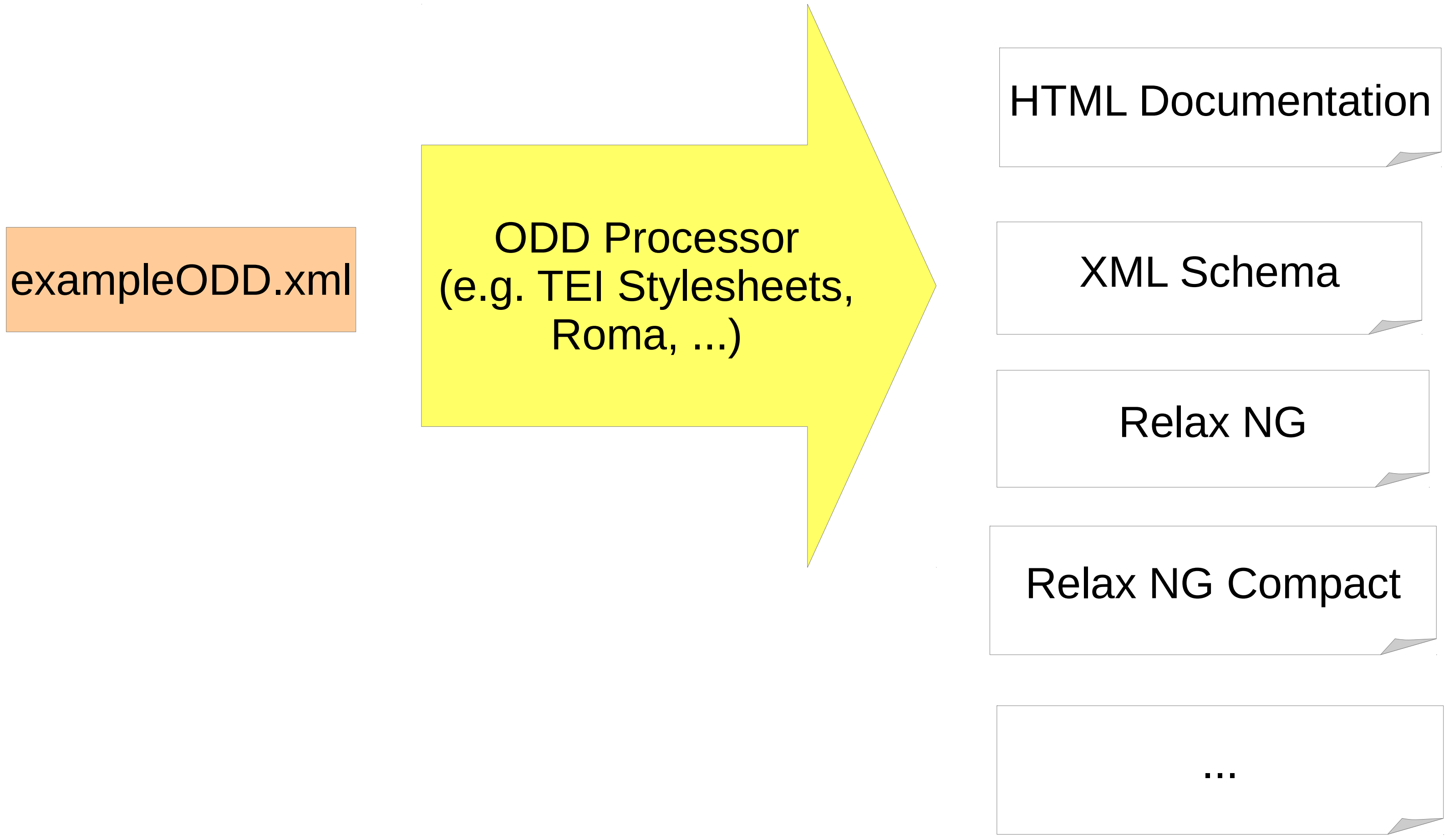
```
start =
   element TEI {
       ...
       element body { part* |
chapter* },
       ...
   }
part =
  element div {
    attribute type { "part" },
    ...(part specific)...
    ( eoaelement | section |
sourcesfeatures)*
  }
chapter =
  element div {
    attribute type { "chapter" },
    ...(chapter specific)...
    ( eoaelement | section |
sourcesfeatures)*
  }
section =
  element div {
    attribute type { "section" },
    ...(chapter specific)...
    ( eoaelement | subsection |
sourcesfeatures)*
  }
subsection = ...
...
```

## Introduction to ODD

- A TEI Document, making use of the `tagdocs` module
- Describes a customisation of an existing ODD
- Elements are grouped into modules

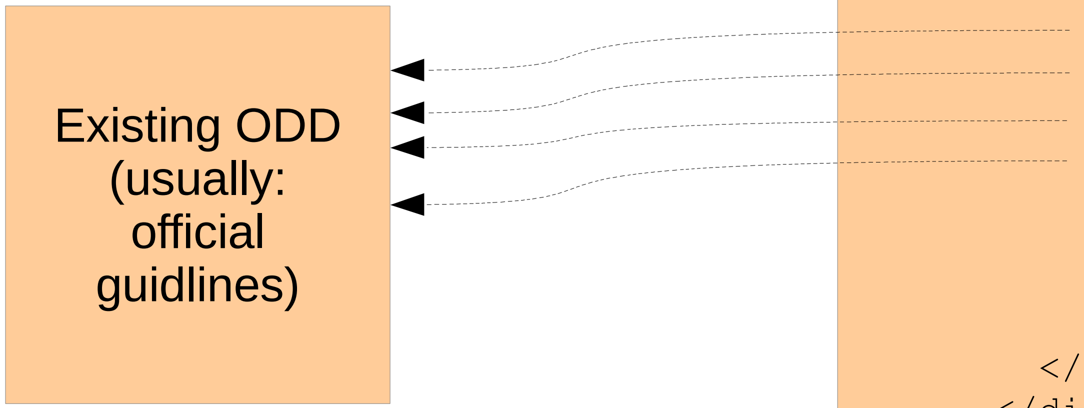
## ODD

- ODD = „One Document Does it All“:



# Introduction to ODD

- Pull in/reference Elements via `moduleRef`, `elementRef`
- add/change/delete elements via `elementSpec`



```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    ...
  </teiHeader>
  <text>
    <body>
      <div>
        <schemaSpec ident="exampleODD">
          <moduleRef key="core"/>
          <moduleRef key="tei"/>
          <moduleRef key="header"/>
          <moduleRef key="namesdates"/>
          ...
          <elementSpec ident="div" mode="change">
            ...
          </elementSpec>
        </schemaSpec>
      </div>
    </body>
  </text>
</TEI>
```

# Introduction to ODD

- Changing elements
  - Allowed content

allowed child elements
(„content model“)

allowed attributes

```
...
<elementSpec ident="div" mode="change">
  <classes>
    ...
  </classes>
  <content>
    ...
  </content>
  <attList>
    ...
  </attList>
</elementSpec>
```

# Introduction to ODD

- Classes
  - Membership in an **attribute class** the element has (at least) all the attributes in the class
  - Membership in a **model class** means the element can appear everywhere the class is mentioned

Classes this element is a member of

```
...
<elementSpec ident="div" mode="change">
  <classes>
    ...
  </classes>
  <content>
    ...
  </content>
  <attList>
    ...
  </attList>
</elementSpec>
```

# Introduction to ODD

- Literate Programming: Specification & Documentation in the same file

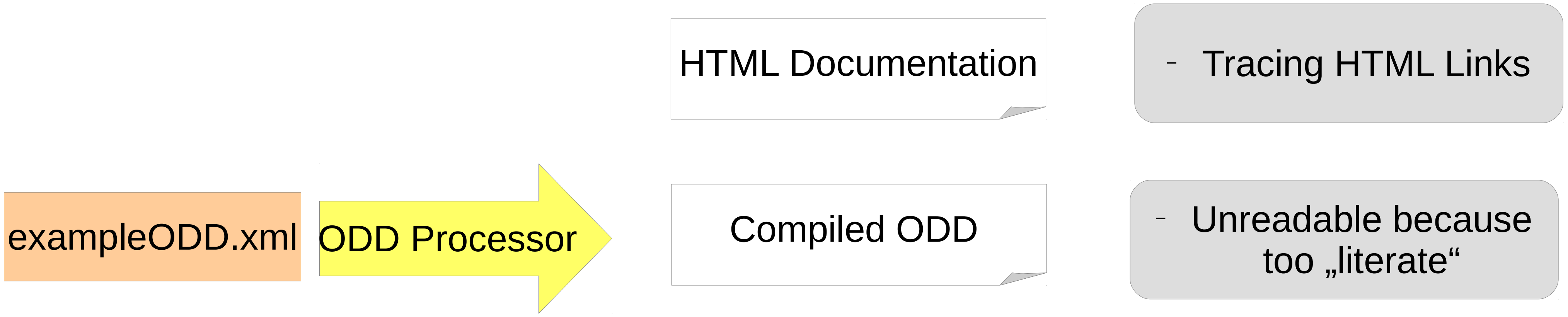
```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    ...
  </teiHeader>
  <text>
    <body>
      <div>
        <schemaSpec ident="exampleODD">
          ...
          <elementSpec ident="div" mode="change">
            ...
            <gloss>bla bla</gloss>
            <desc>detailed description</desc>
          </elementSpec>
        </schemaSpec>
      </div>
    </body>
  </text>
</TEI>
```

## ODD – Challenges

- Class system
  - Class inheritance works differently between model- and attribute classes
  - Documentation sometimes vague
- Different ways of changing an Element:
  - By changing class memberships/deleting classes
  - Simply by overwriting the content model
- How do we know, what our ODD defines?

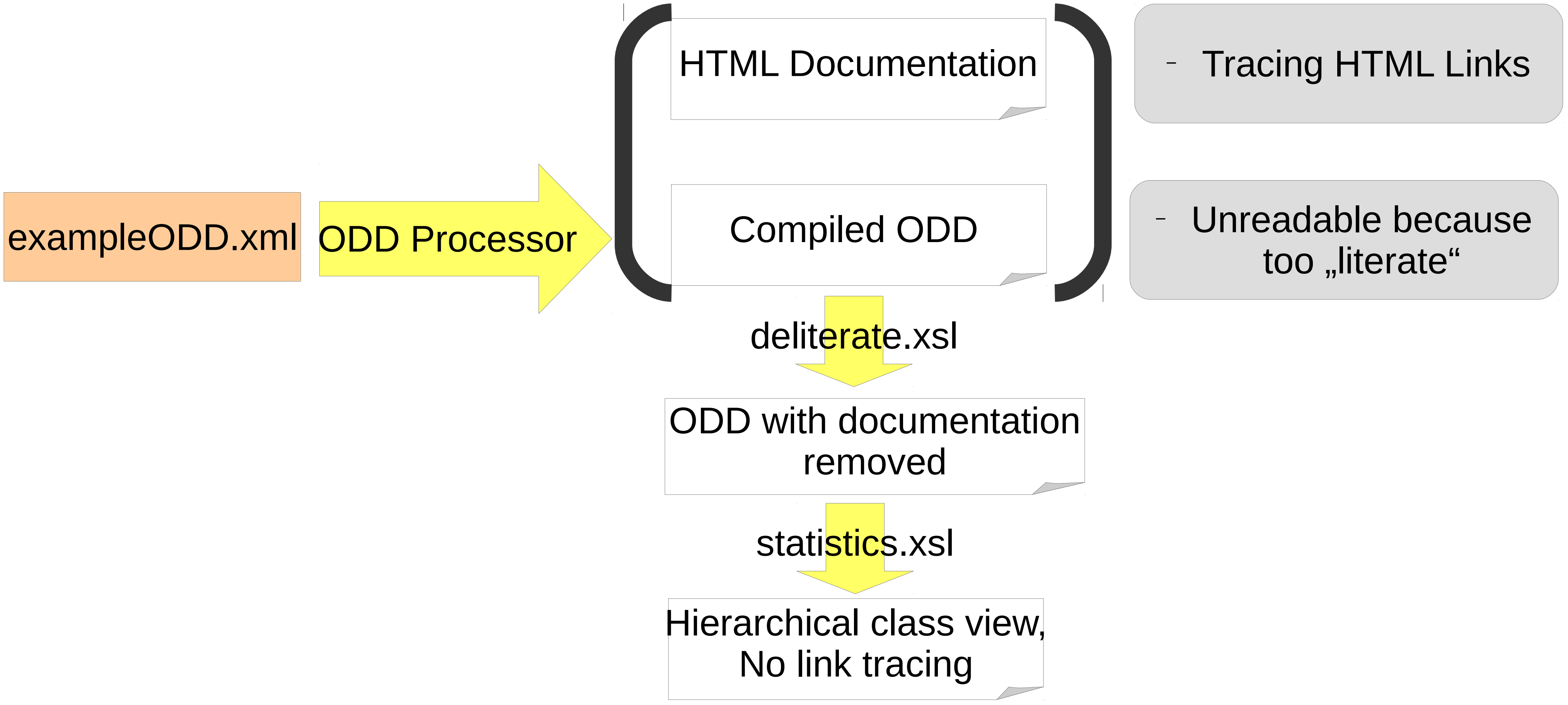
# ODD – Challenges

- How do we know, what our ODD defines?



# ODD – Challenges

- How do we know, what our ODD defines?
- Our Solution:


exampleODD.xml
ODD Processor
HTML Documentation
Compiled ODD
– Tracing HTML Links
– Unreadable because too „literate“
deliterate.xsl
ODD with documentation removed
statistics.xsl
Hierarchical class view, No link tracing

# ODD – statistics stylesheet

statistics.xsl

```
## structured view

modules:

- tei

  elements: (none)

  model classes:

  - model.nameLike.agent (core::name, namesdates::orgName, namesdates::persName)
    - model.nameLike (header::idno) [included by namesdates::model.persNamePart]
      - model.pPart.data ()
        - model.phrase () [included by dictionaries::model.ptrLike.form]
        - model.limitedPhrase ()
  - model.segLike ()
    - model.phrase () [included by dictionaries::model.ptrLike.form]
…
- core

  elements:

  - p
    - attributes: DELETE:@facs, DELETE:@change, DELETE:@resp, tei::att.written (@hand), ...
    - subelements:
    - member of: tei::model.pLike
...
```

## 2. Approach: handcrafted ODD for EOA

- Idea:
  - Look at the Relax NG Schema, and create an ODD for it
- Requirements:
  - The ODD schould be exactly equivalent to the legacy Relax NG Schema
  - The ODD should be a subset of the TEI Guidelines

# 2. Approach: handcrafted ODD for EOA

```
start =
  element TEI {
    ...
    element body { part* | chapter* },
    ...
  }
chapter =                                   ?
  element div {
    attribute type { "chapter" },
    ...
    head-ex-abbr,
    chapterabstract?,
    ( eoaelement | section |
sourcesfeatures)*
  }
...
section =                                   ?
  element div {
    attribute type { "section" },
    head,
    ( eoaelement | subsection |
sourcesfeatures)*
  }
…
subsection =                                ?
  element div {
    attribute type { "subsection" },
    head,
    (eoaelement|subsubsection)*
  }
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    ...
  </teiHeader>
  <text>
    <body>
      <div>
        <schemaSpec ident="ODDfromRnc">
          <moduleRef key="core"/>
          <moduleRef key="tei"/>
          <moduleRef key="header"/>
          <moduleRef key="namesdates"/>
          ...
          <elementSpec ident="div"
mode="change">
            <content>
              ???
            </content>
            <attList>
              ...
            </attList>
          </elementSpec>
        </schemaSpec>
      </div>
    </body>
  </text>
</TEI>
```

## 2. Approach: handcrafted ODD for EOA

- Idea:
  - Look at the Relax NG Schema, and create an ODD for it
- Problem: ⚡
  - Exactly one „content model“ for every element. No way to distinguish between `<div type="chapter"/>` and `<div type="section">` etc.…
- Solution: ☀
  - Schematron!

schematron

A language for making assertions about patterns found in XML documents

- Context specific rules based on XPath
- Can be embedded into ODD

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Schematron Example</title>
  <ns prefix="tei" uri="http://www.tei-c.org/ns/1.0
"/>
  <pattern name="check content of part">
    <rule context="tei:div[@type = 'part']">
      <sch:assert test="if self::tei:div then @type
= ‵chapter‵ else 1">
        invalid content for tei:div[@type = 'part'].
        Expected: only chapter elements
      </sch:assert>
    </rule>
  </pattern>
</schema>
```

≈

If current **context** contains „`tei:div[@type = ‵part‵`“:
  **test** if „`if self::tei:div then @type=‵chapter‵ else 1`“
  otherwise:
    print „invalid content...“

## 2.1 Approach: handcrafted ODD with Schematron

```
chapter =
  element div {
    attribute type { "chapter" },
   ...
    head-ex-abbr,
    element epigraph { p+ }?,
    authorbio?,
    chapterabstract?,
    ( eoaelement | section | sourcesfeatures)*
  }
...
section =
  element div {
    attribute type { "section" },
    head,
    ( eoaelement | subsection | sourcesfeatures)*
  }
…
subsection =
  element div {
    attribute type { "subsection" },
    head,
    (eoaelement|subsubsection)*
  }
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0
">

...
<elementSpec ident="div" mode="change">
  <constraintSpec ident="part"
scheme="schematron">
    <constraint>
<sch:rule xmlns:sch="
http://purl.oclc.org/dsdl/schematron"
context="tei:div[@type = 'part']">
  <sch:assert test="...“>invalid
content for tei:div[@type = 'part']
  </sch:assert>
</sch:rule>
    </constraint>
  </constraintSpec>
  <content>
    ...
  </content>
</elementSpec>
...
</TEI>
```

## Problems defining Schematron rules

- Writing correct `context=` expressions is hard
- Writing correct `test=` expressions is hard
  - we abandoned customising the „content model“

    => we have to check the content with Schematron/XPath
- Writing the rules is cumbersome and error prone

# Writing context expressions is hard:

```
…


ab =
  element ab {
    text
  }
chapterabstract =
   element ab {
     (text | markup | foreign | ref |
bibref)*
   }

...
divX = element div { ab }
divY = element div { chapterabstract }
...
```

- A Schematron rule for every `tei:ab` in the grammar:

```
<elementSpec ident="ab" mode="change">
  <constraintSpec ...>
    <constraint>
 <sch:rule xmlns:sch="
 http://purl.oclc.org/dsdl/schematron"
 context="???">
   <sch:assert test="...">...</sch:assert>
 </sch:rule>
    </constraint>
  </constraintSpec>
  <constraintSpec ...>
    <constraint>
 <sch:rule xmlns:sch="
 http://purl.oclc.org/dsdl/schematron"
 context="???">
   <sch:assert test="...">...</sch:assert>
 </sch:rule>
    </constraint>
  </constraintSpec>
</elementSpec>
```

## Writing context expressions is hard:

- Problem: 2 Definitions for `tei:ab` element with different „content models“. Not distinguishable by their `@type` Attribute values
- Idea „context tracing“: distinguish them by their context element (XPath parent)

# Writing context expressions is hard:

```
…


ab =
  element ab {
    text
  }
chapterabstract =
   element ab {
      (text | markup | foreign | ref |
bibref)*
   }

...
divX = element div { ab }
divY = element div { chapterabstract }
...
```

- context tracing, 1 step:

```
<elementSpec ident="ab" mode="change">
  <constraintSpec ...>
    <constraint>
 <sch:rule xmlns:sch="
 http://purl.oclc.org/dsdl/schematron"
 context="tei:ab[parent::tei:div]">
   <sch:assert test="...">...</sch:assert>
 </sch:rule>
    </constraint>
  </constraintSpec>
  <constraintSpec ...>
    <constraint>
 <sch:rule xmlns:sch="
 http://purl.oclc.org/dsdl/schematron"
 context="tei:ab[parent::tei:div]">
   <sch:assert test="...">...</sch:assert>
 </sch:rule>
    </constraint>
  </constraintSpec>
</elementSpec>
```

## Writing context expressions is hard:

- Still, the elements are not distinguishable by their parents
- Solution: Apply context tracing recursively

## Writing correct tests is hard

```
…
chapterabstract =
   element ab {
     (text | markup | foreign
| ref | bibref)*
   }
```

???

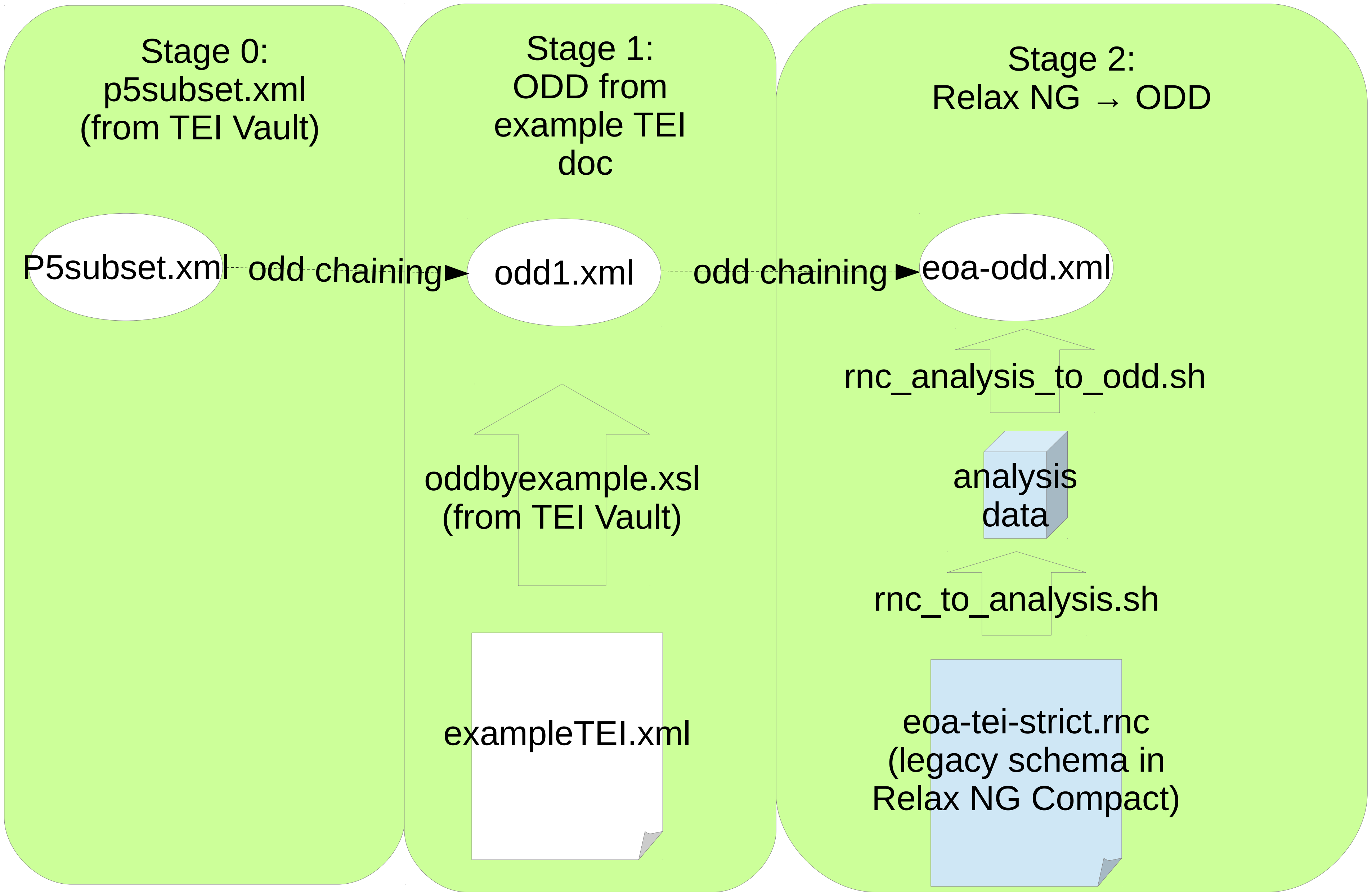
```
<elementSpec ident="ab" mode="change">
  <constraintSpec ...>
    <constraint>

 <sch:rule xmlns:sch="
 http://purl.oclc.org/dsdl/schematron"
 context="...">
   <sch:assert test="
     ???
   ">...</sch:assert>
 </sch:rule>

    </constraint>
  </constraintSpec>
  ...
</elementSpec>
```

## Writing correct tests is hard

- Problem:
  - Need to (forward) trace Grammer rules
  - How do we check advanced Expressions, like:
    - A B
    - A | B
    - A?
    - A+
    - A*

    ?
- Solution: A Parser based on XPath that
  - succeeds exactly for the  correct content model of an element

# Approach 3: Automatic ODD Generation



## Approach 3: Automatic ODD Generation

### Example output:

```
...
<elementSpec ident="ab" mode="change">
  <constraintSpec ident="ab-idm44957540789760" scheme="schematron">
    <constraint>
      <sch:rule xmlns:sch="http://purl.oclc.org/dsdl/schematron"
         context="tei:ab[not(self::tei:ab[@type = 'authorbio']) and not(self::tei:ab[@type =
'chapterabstract']) and not(self::tei:ab[@type = 'equation']) and not(self::tei:ab[@type =
'subequations']) and not(self::tei:ab[@type = 'equationarray']) and not(self::tei:ab[@type =
'theoremdeclaration']) and not(self::tei:ab[@type = 'suggestedcitation']) and not(self::tei:ab[@type =
'bibdatabase'])][parent::tei:div[@type = 'dedication']]">
        <sch:assert test="./ (: check eof :) self::*[ if ( self::tei:ab[not(self::tei:ab[@type =
'authorbio']) and not(self::tei:ab[@type = 'chapterabstract']) and not(self::tei:ab[@type =
'equation']) and not(self::tei:ab[@type = 'subequations']) and not(self::tei:ab[@type =
'equationarray']) and not(self::tei:ab[@type = 'theoremdeclaration']) and not(self::tei:ab[@type =
'suggestedcitation']) and not(self::tei:ab[@type = 'bibdatabase'])][parent::tei:div[@type =
'dedication']] ) then ( not(child::*) ) else ( not(following-sibling::*[1]) ) ]">
          invalid content for tei:ab. expected:ab-idm44957540789760 =
  element ab{ text }
        </sch:assert>
      </sch:rule>
    </constraint>
  </constraintSpec>
  <constraintSpec> ...(some other definition for „tei:ab“)... </constraintSpec>
  <constraintSpec> ...(some other definition for „tei:ab“)... </constraintSpec>
  ...
</elementSpec>
...
```

## Summary: EOA Schema Evolution

**1. Relax NG (Compact)**

- TEI conformance
- User Docu?

+ Nice structure
+ Very expressive

**2. handcrafted ODD**

+ TEI conformance
+ User Docu

- Only on Def. Per El. => Schematron needed
  - ctxt.-free logic as ctxt.-dependent rules
  - not declarative => less support by tools (e.g. autocompletion)

- Challenging to learn:
  - Class system
  - Semantics slightly vague
  - missing tools
- Schematron is hard:
  - Ctxt. Tracing
  - Cumbersome
  - Error prone
- Not feasable by hand

**3. Autogenerated ODD**

- Complex
- Generated Schematron rules are complex and unreadable
- Needs more testing
- Feels like a hack

# Thank you!

- EOA: http://edition-open-access.de
- Repository: https://github.molgen.mpg.de/EditionOpenAccess (MIT License)

Funded by the German Ministry for Education
and Research (Grant Nr. 16OA061)