From XML to UDL

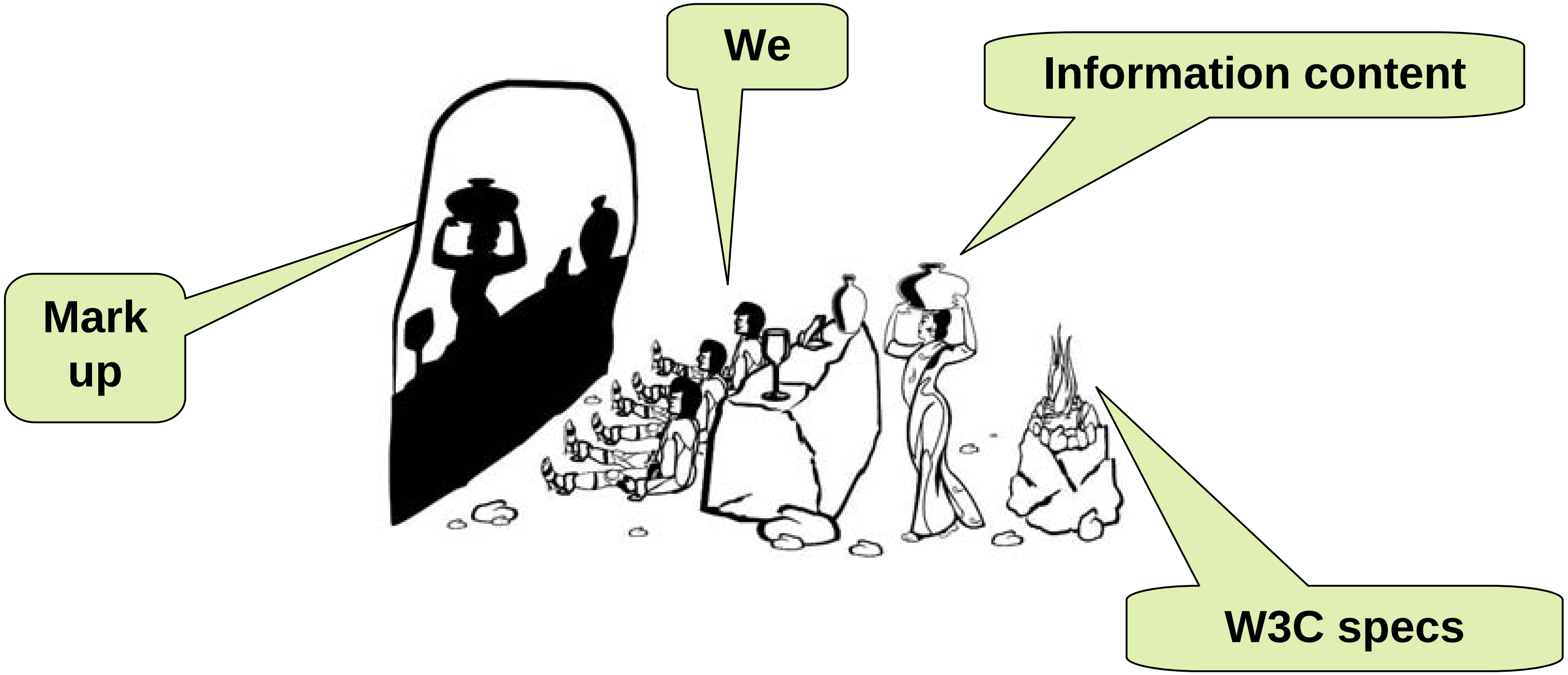
A unified document language, supporting multiple markup languages

Hans-Jürgen Rennau, Traveltainment GmbH

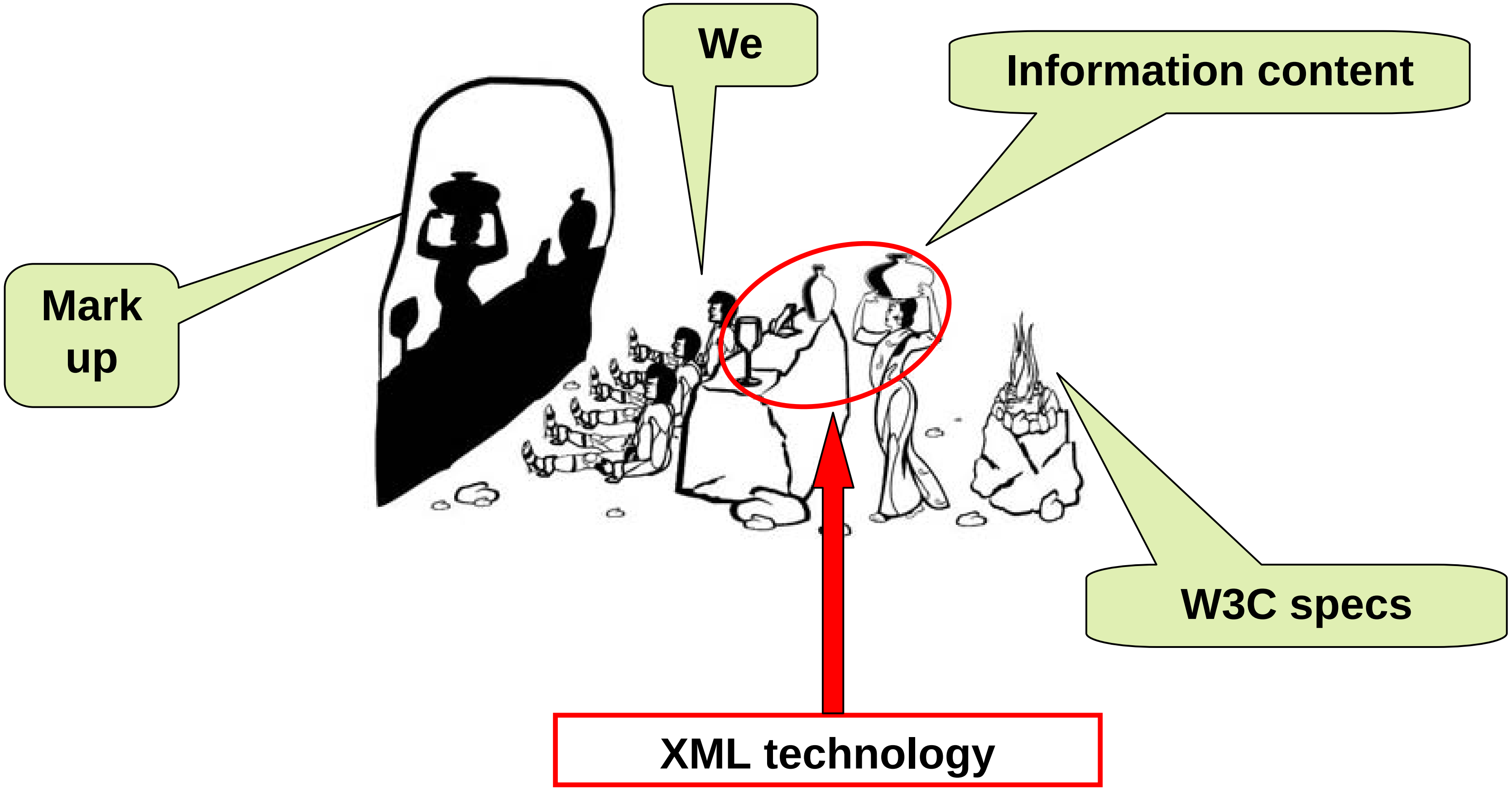
Presented at Balisage 2012, August 7

Plato’s allegory of the cave

XML’s platonic nature



XML technology sees content



We

Information content

Mark up

W3C specs

XML technology

XML technology sees nodes

Layer: *shadows* **(markup)**	**<foo>…</foo>**
Layer: *things* **(XDM items)**	**node-items**
Layer: *technology* **(code)**	**XPath, XQuery, XSLT**

Jonathan Robie:
“The dream is past…”

The dream of one universal markup language
is now past.

JSON is clearly here to stay, and it is becoming the format of choice for data interchange.

What to do about JSON?

Layer: ***shadows*** **(markup)**	**<foo>…</foo>**
Layer: ***things*** **(XDM items)**	**node-items**
Layer: ***technology*** **(code)**	**XPath, XQuery, XSLT**

Integration approach #1: JSONiq

Layer: ***shadows*** **(markup)**	***XML document*** **<foo>…</foo>**	***JSON document*** **{ … }**
Layer: ***things*** **(XDM items)**	**node-items**	***json-items*** ***(array, object)***
Layer: ***technology*** **(code)**	**XPath, XQuery, XSLT**	

Integration approach #2: XSL Working Group

Layer: *shadows* (markup)	***XML document*** **<foo>…</foo>**	***JSON document*** **{ … }**
Layer: *things* (XDM items)	**node-items**	***map-items***
Layer: *technology* (code)	**XPath, XQuery, XSLT**	

Proposal: UDL = Unified Document Language

Layer: ***shadows*** ***XML document*** ***JSON document***
(markup) **<foo>…</foo>** **{ … }**

--

Layer: ***things***
(XDM items) **node-items**

--

Layer: ***technology***
(code) **XPath, XQuery, XSLT**

UDL – main idea

XPath navigation is based on nodes

➔

JSON data should be modeled as node tree!

We need a model which

- “Redefines” JSON to represent a node tree
- Defines JSON parsing: markup => tree
- Defines JSON serialization: markup <= tree

To model JSON as node tree – a naïve approach (1)

- Content
  - Objects: elements and their child elements
  - Arrays: elements and their child elements
  - Simple values: elements with simple content
  - Null values: nilled elements
- Names
  - JSON names: element names
  - Array members: use standard names (e.g. “item”)
- J-Structure (object/array)
  - Object/array distinction ad hoc attribute (e.g. “is-array”)

To model JSON as node tree – a naïve approach (2)

```
[
 {"code" : "AAL", "airport" : "Aalborg, Denmark"},
 {"code" : "AES", "airport" : "Aalesund, Norway"},
 {"code" : "ZID", "airport" : "Aarhus, Denmark"}
]
```

➔

```
<j:array j:is-array="true">
   <j:item>
      <code>AAL</code>
      <airport>Aalborg, Denmark</airport>
   </j:item>
   <j:item>
      <code>AES</code>
      <airport>Aalesund, Norway</airport>
   </j:item>
   <j:item>
      <code>ZID</code>
      <airport>Aarhus, Denmark</airport>
   </j:item>
</j:array>
```

The name problem: JSON names are *strings*

```
[
 {"code" : "AAL", "name info" : "Aalborg, Denmark"},
 {"code" : "AES", "name info" : "Aalesund, Norway"},
 {"code" : "ZID", "name info" : "Aarhus, Denmark"}
]
```

➔

```
<j:array isArray="true">
   <j:item>
      <code>AAL</code>
      <name_0020info>Aalborg, Denmark</name_0020info>
   </j:item>
   <j:item>
      <code>AES</code>
      <name_0020info>Aalesund, Norway</name_0020info >
   </j:item>
   <j:item>
      <code>ZID</code>
      <name_0020info>Aarhus, Denmark</name_0020info>
   </j:item>
</j:array>
```

UDL: JSON <=> node tree

- Content
  - Objects: elements and their child elements
  - Arrays: elements and their child elements
  - Simple values: elements with simple content
  - Null values: nilled elements
- Names
  - JSON names: new node property [key]
  - Element names: standard names
- J-Structure (object/array)
  - Object/array distinction new node property [model]

UDL – JSON and node tree

```
[
 {"code" : "AAL", "name info" : "Aalborg, Denmark"},
 {"code" : "AES", "name info" : "Aalesund, Norway"},
 {"code" : "ZID", "name info" : "Aarhus, Denmark"}
]
```

➔

```
<udl:array>
  <udl:map udl:model="map">
    <udl:value udl:key="code">AAL</udl:value>
    <udl:value udl:key="name info">Aalborg, Denmark</udl:value>
  </udl:map>
  <udl:map udl:model="map">
    <udl:value udl:key="code">AES</udl:value>
    <udl:value udl:key="name info">Aalesund, Norway</udl:value>
  </udl:map>
  <udl:map udl:model="map">
    <udl:value udl:key="code">ZID</udl:value>
    <udl:value udl:key="name info">Aarhus, Denmark</udl:value>
  </udl:map>
</udl:array>
```

UDL extends the node model

- New node property: [key]
  - Like a second “name”, string-based, optional
  - Present if – and only if – the parent has [model] = map

- New node property: [model]
  - Value **`sequence`**
    - Child elements MUST NOT have a key
    - Content = ordered collection of child nodes

  - Value **`map`**
    - Child elements MUST have a key
    - Content = unordered collection of child elements

The [model] property

The new [model] property is a tribute to the fact that the conventional XML content model is *not as universal as it looked*.

Value “**sequence**” is the conventional XML model: content is a sequence of child nodes:

structure is order-based.

Value “**map**” is the big alternative: content is a map of child nodes: structure is key-based.

Extension of XPath: key test

- Key test – a third node test
  (besides *name test*, *kind test*)

- Checks the value of the **[key]** property

- Syntax

#foo	possible if only name chars
#"last name"	always possible

- Freely combinable with axes
  `descendant::#a/parent::#b/#c`

The key test in action

```
[
 {"code" : "AAL", "name info" : "Aalborg, Denmark"},
 {"code" : "AES", "name info" : "Aalesund, Norway"},
 {"code" : "ZID", "name info" : "Aarhus, Denmark"}
]
```

Queries

```
/*/*/#code/string()
➔ "AAL", "AAL", "AAL„

/*/*[#code eq "ZID"]/#"name info"/string()
➔ "Aarhus, Denmark"

/descendent::#code[. eq "ZID"]/../#"name info"/string()
➔ "Aarhus, Denmark“
```

Extension of XQuery: JSON constructors

```
let $country := "Denmark"
let $codes := //#airport[. contains $country]/../#code
return
   {"country" : $country, "codes" : [ $codes ]}
=

…
return
<udl:map udl:model="map">
  <udl:value udl:key="country">{$country}</udl:value>
  <udl:array udl:key="codes">{
     for $code in $codes
     return <udl:value>{$code}</udl:value>
  </udl:array>
</udl:map>
```

➔ `{ "country" : "Denmark", "codes" : ["AAL", "ZID"] }`

JSON constructors: overview

- Key-oriented constructor: Expr1 : Expr2

  Example: $label : $src/addInfo

- Map constructor { Expr }

  Example: { “code” : $code, “title” : $title }

- Array constructor [Expr]

  Example: [$codes]

Comprehensive example (1)

- Task
  - Transform a JSON document
  - Use XQuery

- Demonstrates:
  - Querying JSON data
  - Constructing JSON data

Comprehensive example: JSON input

```
[
   {
      "year" : 2011,
      "title" : "JSON",
      "author" : [
         {"last" : "Legoux", "first" : "C."}
      ],
      "price" : 35.95,
      "sigs" : ["LL1002"]
   },
   {
      "year" : 2012,
      "title" : "XML",
      "author" : [
         {"last" : "Legoux", "first" : "C."},
         {"last" : "Berlin", "first" : "D."}
      ],
      "price" : 29.95,
      "sigs" : []
   }, …
]
```

Comprehensive example: JSON output

```
[
   {
      "name" : "Legoux, C.",
      "books" : [
         {"title" : "JSON", "year" : "2011"},
         {"title" : "UDL",  "year" : "2012"},
         {"title" : "XML", "year" : "2012"}
      ]
   },
   {
      "name" : "Okuda, J.",
      "books" : [
         {"title" : "UDL", "year" : "2012"}
      ]
   },
   …
]
```

Comprehensive example: query

```
[
  for $author in distinct-values(
    //#author/*/concat(#last , ', ', #first))
  let $books :=
    //#author[*/concat( #last , ', ', #first ) = $author]/..
  order by $author
  return
    {
      "name" : $author,
      "books" : [
        for $book in $books
        order by $book/#title
        return
        {
          "title" : $book/#title/string(),
          "year" : $book/#year/string()
        }
      ]
    }
]
```

Extension of XML markup

- No new syntactical constructs
  - **pseudo attributes** (and one pseudo tag)
- Goal #1: express new node properties
  - udl:key, udl:model, udl:defaultModel
- Goal #2: enable insertion of non-XML markup
  - udl:markup

JSON within XML: example

```
<codes xmlns="http://example.com">
  <iata udl:markup="json">
  [ { "code" : "AAL", "airport" : "Aalborg, Denmark" },
    { "code" : "AES", "airport" : "Aalesund, Norway" } ]
  </iata>
</code>
==============================================================
<codes xmlns="http://example.com">
 <iata>
   <udl:map udl:model="map">
     <udl:value udl:key="code">AAL</udl:value>
     <udl:value udl:key="airport">Aalborg, Denmark</udl:value>
   </udl:map>
   <udl:map udl:model="map">
     <udl:value udl:key="code">AES</udl:value>
     <udl:value udl:key="airport">Aalesund, Norway</udl:value>
   </udl:map>
 </iata>
</codes>
```

JSON *within* XML

- <foo udl:markup="json">**{ ... }**</foo>

- <foo udl:markup="json">**[...]**</foo>

- Content of <foo>:

  child nodes of the map (or array) element represented by the JSON markup

JSON *instead of* XML

- Document **with** markup declaration

```
<?udl markup=”json”>
[
 { "mtype" : 23, "from" : "C12", "to" : "D02" },
 { "mtype" : 11, "from" : "C22", "to" : "E01" },
 { "mtype" : 41, "from" : "C31", "to" : "V02" },
 { "mtype" : 50, "from" : "C01", "to" : "V02" },
]
```

- Document **without** markup declaration

```
[
 { "mtype" : 23, "from" : "C12", "to" : "D02" },
 { "mtype" : 11, "from" : "C22", "to" : "E01" },
 { "mtype" : 41, "from" : "C31", "to" : "V02" },
 { "mtype" : 50, "from" : "C01", "to" : "V02" },
]
```

Serialization model

- Serialization param method: new value json

- New serialization parameter info-loss:
  - `json.strict` – any information loss causes error
  - `json.ignore-names` – element names are ignored
  - `json.projection` – any information loss is ignored
    - Element names are ignored
    - Attributes are ignored
    - Mixed content is projected to element children

The scope of the UDL proposal

Extension of…	Goal
1) XDM node model	express XML documents *and* JSON documents
2) XML markup	a) complete representation of the node model b) combination of XML and non-XML markup
3) XPath	support navigation by new node property [key]
4) XQuery	support elegant construction of JSON structures
5) Serialization model	control the handling of information loss

Limitations & future research – standard mappings

- UDL achievements (= UDL core)
  - Translation: JSON markup ⇔ node tree
  - Processing: JSON data with XML technologies
  - Markup integration: JSON within XML

- UDL – not yet addressed (= UDL extensions)
  - Mapping: JSON => equivalent, *readable* XML
  - Mapping: any XML => JSON (lossless)
  - Round-tripping: X-J-X

UDL and mapping solutions

Many XML/JSON mapping solutions have been proposed. However, UDL provides a new conceptual framework which might facilitate the definition of mapping standards.

Mappings are now node tree transformations. The definition and evaluation of mappings can be built on a firm basis.

First step towards mapping support

nJSON document:

```
{
   "date" : "2012-08-06",
   "place" : "London",
   "temperatures" : ["12", "21"]
}
```

nnJSON document:

```
<getWeatherRS xmlns="http://example.com" udl:model="map">
   <date>2012-08-06</date>
   <place>London</place>
   <temperatures>
      <t>12</t>
      <t>21</t>
   </temperatures>
</getWeatherRS>
```

Evaluating UDL proposal – two distinctions

- Distinction #1
  - Central idea: JSON markup = node tree
  - Translation into technical detail:

    node properties [key] and [model]

- Distinction #2
  - What *is* achieved (UDL core)
  - What *should be* achieved (UDL extensions):

    mapping support, …?

Thank you!