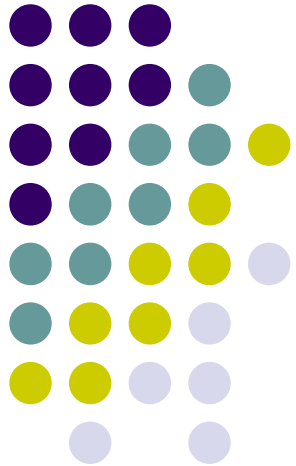
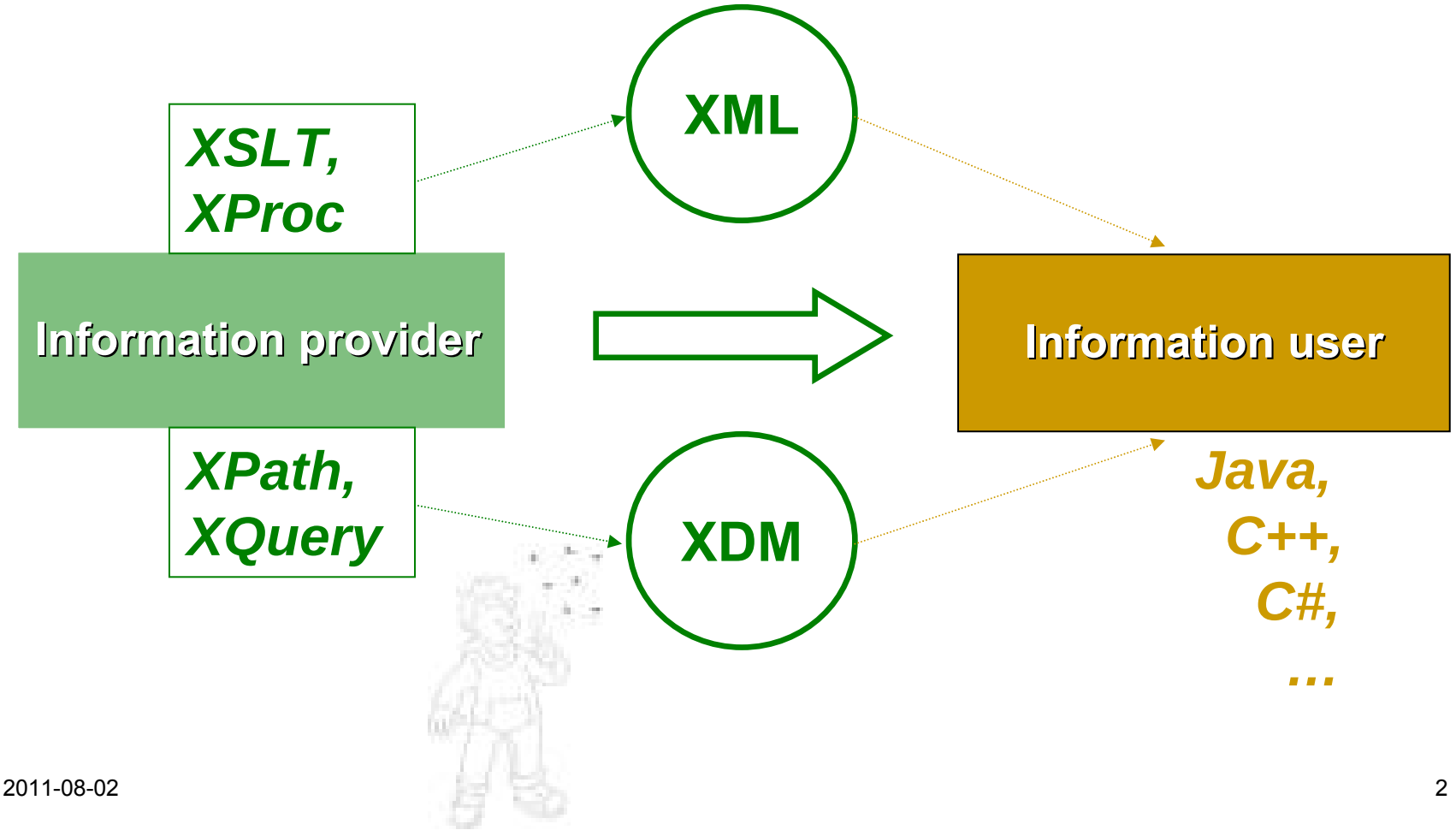
XDML

an extensible markup language for XDM

Hans-Jürgen Rennau, bits GmbH
David Lee, Epocrates Inc.



Information *delivery*



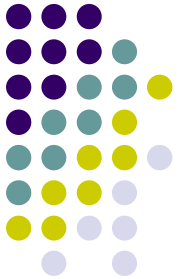
XML is great but…

- Sometimes we want **atomic values**, not nodes…

  "Mike, <quantity>1</quantity> more beer, please!"

- Sometimes we want a **collection** of independent entities, not a monolithic, single tree

  Would you like file directories to contain a single "content document", rather than a collection of files?

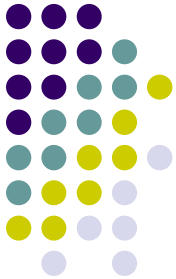


… XDM is larger

- **Collection** of independent items
- Supports XML data *and* **atomic values**

- Model summary
  - Every value is a sequence of items
  - An item is either a node or an atomic value
  - Seven node kinds *(document, element, text, …)*
  - 45 atomic types *(string, integer, date, …)*

XDM 3.0 (working draft)



- XDM 1.0 + further item types:
  - function items; map items; *array items?*

- **Map items**
  - Keys – atomic value (e.g. string or QName …)
  - Values – XDM item sequences

- Implications …
  - Support for complex values composed of named parts
  - Parts can be nested

- *Example*: a **file system** can be represented by an XDM value:
  directory = map item; XML file = node item; other file = atomic item

Map item – an example

```
{
    "state"          => "FL"
    "projects"       => "I04", "S40", "S48"
    "report"         => <waterReport>…
    "schema"         => <xsd:schema …>…
    "interface"      =>
        {
          "getSummary"    => 'xquery 1.0 …; declare … "
          "getLocations" => "//loc/@name/string(.)"
          "toHTML"        => <xsl:transform…>…
          "broadcast"     => <c:xproc …> …
        }
}
```

XDM 3.0 & integration

- **Retrieval** of the parts: convenient (named-based)
- **Use** of the parts: unassisted by metadata

Usage Category	***Example***
Transformation	Apply XSLT
Extraction	Apply XPath
Evaluation	Apply XQuery
Execution	Execute as a Perl script or stylesheet
Storage	Store in a file or database
Delivery	Pass on to another component
Dispatchal	Use as a SOAP request
… more categories	*… and more examples*

XDM for integration: XDM*L*

- XDML = extensible markup language for XDM
- Goal: creation of more useful XDM values
  Perspective: ***XDM user point of view***
- Idea: add control items (“markup”) which …
  - Create structure
  - Attach metadata
- Compatible with XDM 1.0 and XDM 3.0

XDML – a first example

```
<xm:part name="projects" type="strings"/>,
"I04", "S40", "S48" ,

<xm:part name="report" type="node"/>
<waterReport>…,

<xm:part name="toHTML" type="node"/>,
<xsl:transform …>…,
```

Named parts

XDML value = data items + *control items*

XDML – adding metadata

```
<xm:part name="projects" type="strings"/>,
"I04", "S40", "S48" ,

<xm:part name="report" type="node">
   <xm:interface>
      <xm:method name="getHTML">
         <submitToXSLT serialize="true">
            <stylesheet>$part{toHTML}</stylesheet>
         </submitToXSLT>
      </xm:method>
   </xm:interface>
</xm:part>
<waterReport>…,

<xm:part name="toHTML" type="node" private="true">,
<xsl:transform …>…,
```

Metadata = attributes and children of *control items*

XDML – (Java) user code

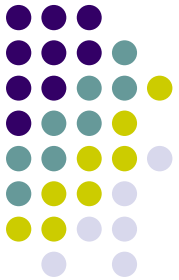
```
// *** obtain value
XDML xdml = …;

// *** retrieve parts
String[] pro  = xdml.getStrings("projects");
Node     rep  = xdml.getNode("report");

// *** invoke part methods
String   repH = (String)xdml.invoke("report","getHTML");
```

Retrieval of *part data*, invocation of *part methods*

XDML - definition

- **Syntax rules** guide the creation of XDML values
  - Adding structure (named, nestable parts)
  - Adding metadata (each part has its metadata)
- An **information model** defines the information content of an XDML value
- A **processing model** specifies a processing …
  - Defined by metadata
  - Triggered by XDML user actions (API calls)
- An **API** enables the XDML user to retrieve and to process data

XDM*L* – syntax rules



- XDML value = data items + control items

- **Schema** for control items

  (`xm:part, xm:complexPart`*)*

- **Composition rules** - how to associate control items with data items
  - Present encoding: XDM 1.0 (no map items)
  - Later: alternative based on map items

Simple and complex parts

```
<xm:part name= "projects" type="strings"/>,
"I04", "S40", "S48" ,
```

```
<xm:part name="report" type="node"/>,
<waterReport>…,
```

```
<xm:part name="schema" type="node" />,
<xsd:schema …>…,
```

```
<xm:complexPart name="codelib">,

<xm:part name="getSummary" type="string">,
"xquery version 1.0 …; declare … ",

<xm:part name="toHTML" type="node">,
<xsl:transform …>…,

<xm:complexPartEnd/>
```

XDML – information model

- XDML value = collection of **information units**
- Information unit = name + value + metadata
- **Simple** information unit
  - Value = sequence of data items
  - No nested units
- **Complex** information unit
  - Value = set of nested units
  - No data items outside of nested units
- **Metadata** = descriptive metadata + control metadata

XDML – processing model

- Principle: processing is …
  - Defined by metadata
  - Triggered by user actions (API calls)
- Control metadata = definition of **post-processing**
- Definition of post-processing = design and implementation of a unit **interface**:
  - Names and implementation of methods
  - Implementation of certain built-in methods (“finalize” and “execute”)

Information with an interface

```
<xm:part name="logdata" type="node">
  <xm:interface>
    <xm:method name="getResources" dependsOn" c_gr">
      <submitToXSLT>…
      <submitToXQuery>…
    </xm:method>
    <xm:method name="getWarnings">
      <submitToXQuery>…
    </xm:method>
    <xm:method name="getErrors">
      <submitToXQuery>…
     </xm:method>
  </xm:interface>
</xm:part>,
<hhla:log …>…
```

Method

Operations

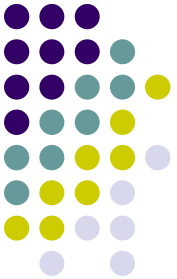
Data context

Information with an interface

```
<xm:part name= "logdata" type="node">
  <xm:interface>
    <xm:method name="getResources" dependsOn="c_gr">
      <submitToXSLT>
        <stylesheet>$part{c_gr}</stylesheet>
      </submitToXSLT>

      <submitToXQuery>
        <query>
for $r in //resource order by $r/@id
return $r/@id/string()
        </query>
      </submitToXQuery>
    </xm:method>
    …
  </xm:interface>
</xm:part>,
<hhla:log …>…
```

Method

Data context



Operations

- Basic unit of available functionality
  - Consumes input
  - Produces a result (optional)
  - May have side effects
- Output = any value, not necessarily an XDM value
- Input = **data context** + **request message**
- Data context is …
  - The value of the containing information unit …
  - … or the result of a preceding operation

Operation request message

- Element providing the values of operation parameters
- Statically known and dynamic parameters

- Parameter values may …
  - Reference a named **invocation argument**
  - Reference the **value of an information unit**

```
<xm:method name="getSummary" params="select">
    <submitToXSLT serialize="false">
        <stylesheet>$part{getSummary}</stylesheet>
        <xm:params selectedLocs="$arg{select}"/>
    </submitToXSLT>
</xm:method>
```

Invocation arguments

```
<xm:part name="emails" type=„node“>
   <xm:interface>
      <xm:method name="getSummary" params="date select">
         <execAsPerl args="-m sum -d $arg{date}"/>
         <submitToXSLT serialize="false">
            <stylesheet>$part{getSummary}</stylesheet>
            <xm:params selectedLocs="$arg{select}"/>
         </submitToXSLT>
      </xm:method>
   </xm:interface>
</xm:part>,
#!/usr/bin/perl -w …
================================================
XDML xdml = …;
Args args = new Args();
args.set("date", "2011-03-21").set("select", "FL");
Node summary = xdml.invoke("emails", "getSummary", args);
```

Available operations

- **User-defined operations** *(runtime registration)*
- **Standard operations**

createMapFromStrings
createPropertiesFromString

execAsSQL
execAsPerl
execAsSQL
execAsSystemCmd
execAsXProc
execAsXQuery
execAsXSLT

readDocument
readTextFile

sendFTP
sendSOAP

submitToPerl
submitToSystemCommand
submitToXQuery
submitToXSLT
submitToXProc

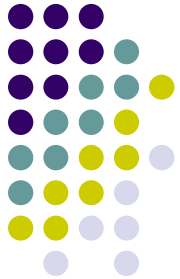
validate

writeDocument
writeTextFile

Methods

- Unit of defined processing
- Execution = sequential execution of one or several operations
- Definition = sequence of operation request messages
- The definition is bound to an “**execution context**”:
  - When to execute
  - What to do with the return value

Execution context



- When to execute a method, what to do with the return value.
- Context “**enable**”:
  - Return value is delivered to the caller
  - Triggered by API call `invoke(partName, methodName)`
- Context “**execute**”:
  - No return value
  - Triggered by API call `execute()`
- Context “**finalize**”:
  - The return value replaces the initial value of the information unit
  - Triggered by API call `finalize()`

Execution context “enable”

XDML value:

```
<xm:part name="log" type="node">
   <xm:interface>
      <xm:method name="getWarnings">
         <submitToXQuery>
            <query>//*[etype eq "warn"]</query>
         </submitToXQuery>
      <xm:method>
   </xm:interface>
</xm:part>
<log>…</log>
```

==========

XDML user code:

```
XDML xdml = …;
Node[] w = (Node[]) xdml.invoke("log","getWarnings");
```

Execution context “execute”

XDML value:

```
<xm:part name= "backupWizard" type="string">
   <xm:execute>
      <execAsPerl/>
   </xm:execute>
</xm:part>,
#!/usr/bin/perl -w
Use strict;
Use Time::Local;
…

==========================================================
```

XDML user code:

```
XDML xdml = …;
xdml.execute();
```

Execution context “finalize”

XDML value:

```
<xm:part name="curTemperature" type="string">
   <xm:finalize>
      <sendSOAP href="http://meteo.org/ws“/>
      <submitToXQuery>
         <query>//*:temperature/xs:string</query>
      </submitToXQuery>
   </xm:finalize>
</xm:part>,
<getTemperature><site>abc</site></getTemperature>

==================================================
```

XDML user code:

```
XDML xdml = …;
xdml.finalize();
String temp = xdml.getString("curTemperature");
```

Execution context “finalize”

XDML value:

```
<xm:part name= "logdata" type="node">
  <fm:finalize dependsOn="logx">
    <submitToPerl>
      <perl>$part{logx}</perl>
    </submitToPerl>
  </fm:finalize>
  <xm:interface>
    <xm:method name="getResources" …>…
  </xm:interface>
</xm:part>, "log-ops-20110401.txt"
================================================
```

XDML user code:

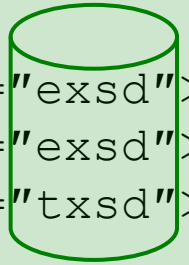
```
XDML xdml = …;
xdml.finalize();
String[] r = xdml.invoke("logdata", "getResources");
```

Implicit finalization

```
<xm:part name="xsd" type="node">
   <xm:interface>
      <xm:method name="getElemNames" dependsOn="exsd">
         <submitToXQuery>…
      <xm:method>
   </xm:interface>
</xm:part>,
<xsd:schema…>…,

<xm:part name="exsd" type="node" private="true">
   <xm:finalize dependsOn="xsd">
      <execAsXQuery>
         <contextItem>$part{xsd}</contextItem>
      </execAsXQuery>
   </xm:finalize>
</xm:part>,
xquery version 1.0;
…
```

Example – “schema-reporter”

```
<xm:part name="xsd" type="node">
  <xm:finalize params="href">
    <loadDocument href="$arg{href}"/>
  </xm:finalize>
  <xm:interface>
    <xm:method name="getTargetNS">…
    <xm:method name="getElemNames"dependsOn="exsd">…
    <xm:method name="getTypeNames"dependsOn="exsd">…
    <xm:method name="getDataPaths"dependsOn="txsd">…
  </xm:interface>,
</xm:part>,   "TO-BE-REPLACED-BY-XSD",
```



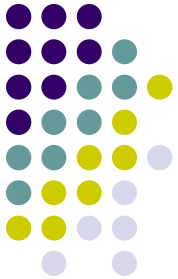
```
<xm:part name="exsd" type="node" private="true">
  <xm:finalize dependsOn="xsd">…
</xm:part>,   "TO-BE-REPLACE-BY-EXPANDED-XSD",
<xm:part name="txsd" type="node" private="true" >
  <xm:finalize dependsOn="exsd">…
</xm:part>,   "TO-BE-REPLACED-BY-TREE"
```

Example - user perspective

```
// *** loading the XDML value
XDML xdml = …;
Args args = new Args().set("href", "weather.xsd");
xdml.finalize(args);

// *** use the information interface
String   tns    = xdml.invoke("xsd", "getTargetNS");
String[] enames = xdml.invoke("xsd", "getElemNames");
String[] tNames = xdml.invoke("xsd", "getTypeNames");
String[] dpaths = xdml.invoke("xsd", "getDataPaths");
```

All data are hidden behind the interface;
intermediaries are generated just in time and cached.

XDML benefits (SMEB)



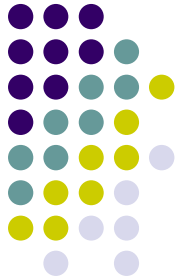
- **S**tructure – XDML creates an intuitive structure which
  - Integrates XML nodes and atomic values
  - Treats its parts as independent, self-contained entities
- **M**etadata – XDML attaches metadata externally, non-invasively.
- **E**xtended functionality – XDML provides access to XDML operations, often offering otherwise unavailable functionality.
- **B**ehavior – XDML may associate data with an interface.

XDML issues (CTI)

- Complexity – often overhead
  - We don't need named parts if there is only one part
  - We don't need methods if the processing is trivial
- Type system – still immature
  - Uneasy integration of XDM data types and arbitrary types
  - Missing formal definition of type matching
- Interoperability of operations - not yet formally defined
- *Next steps*:
  - Simplified creation/usage of simple results
  - Interface inheritance, XDML value import, …

Imagine…

XSLT,
XProc
XML
Information provider
XDML
Info. user
XPath,
XQuery
XDM
Java,
C++,
C#,
…
Technology
Encoding
Information



Thank you!