XProc: Beyond application/xml

Vojtěch Toman
EMC Corporation
vojtech.toman@emc.com

XML Prague 2012

EMC²

Motivation

“[XProc is] a language for describing operations to be performed on XML documents.”

“…what flows between steps through input ports and output ports are exclusively XML documents or sequences of XML documents.”

vs.

- Real-life pipelines often have to deal with non-XML data
  - Read from external sources
  - Produced by the pipeline itself

EMC²

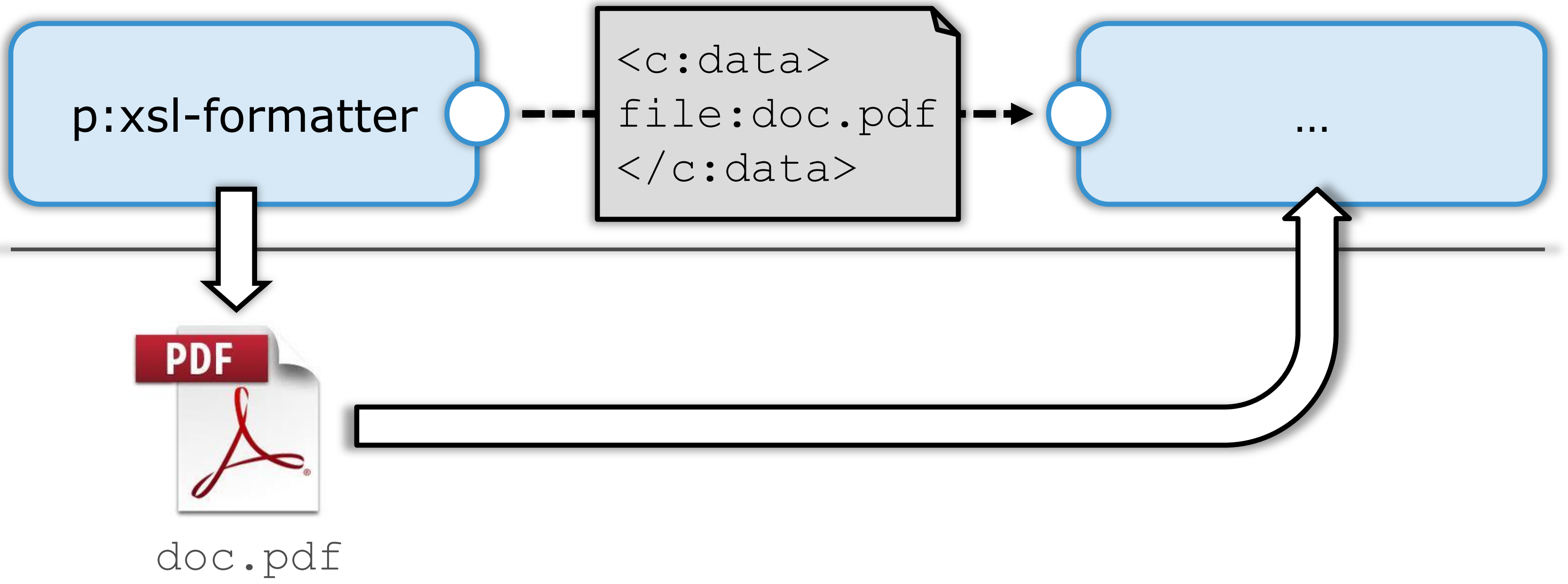
BaSE64enCoDINg==

```
<c:data content-type="application/octet-stream"
        encoding="base64">
  QUwsQWxhYmFtYQpBSyxBbGFza2EKQVosQXJpem9uYQo...
</c:data>
```

- Not much we can do with such content
  - Sending it over HTTP using `p:http-request`
  - Unescaping it with `p:unescape-markup`
- Cannot use `p:store` to store the raw octet stream
- Need for extensions

Using an External Channel

- Steps use an external channel for non-XML data
  - File system
- Steps pass URI references to the external data



Introducing Non-XML Media Types

- XProc is built from the ground up on XML Infoset
  - Steps expect XML Infoset instances on the input ports and produce XML Infoset instances on the output ports.
- Option 1
  - XProc processor provides some kind of a (synthetic) XML Infoset view
- Option 2
  - The steps can operate on non-XML data as well
  - `p:identity`, `p:store`, `p:sink`, ...

Introducing Non-XML Media Types

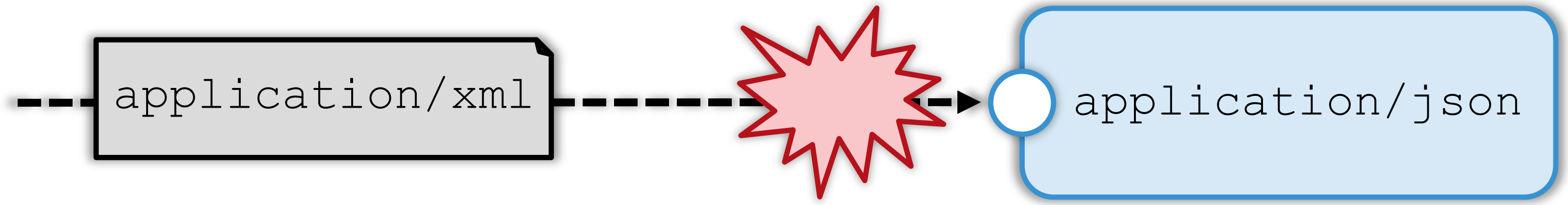
- XProc uses XPath as the expression language
- What does querying over non-XML data actually mean?
- Does it correspond to querying some kind of metadata gleaned from the original data?
  - Dimensions of an image
- Or is it the ability to inspect the raw octet stream?
  - Querying text or semi-binary formats

Proposed Extension at Glance

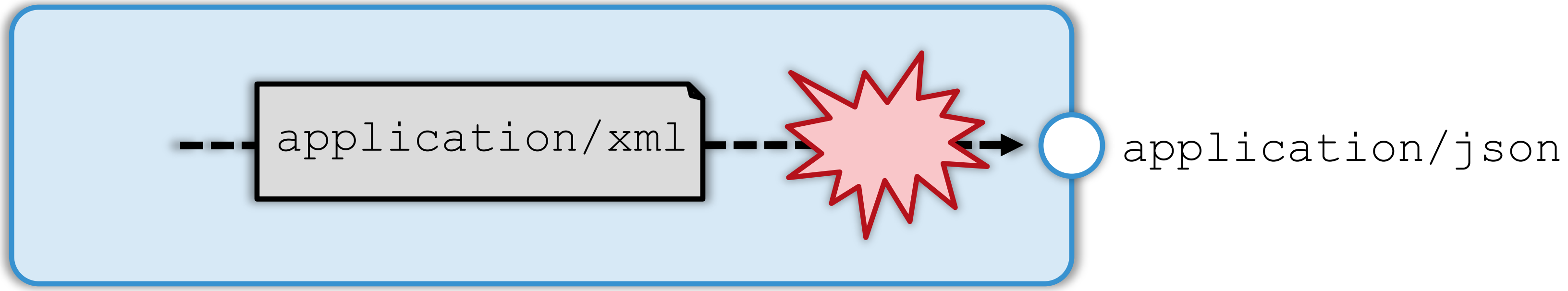
- Both XML and non-XML data can flow through the pipeline
  - XML data flows as XML Infoset instances
  - Non-XML data flows as "raw" octet streams
- The data is annotated with media type information
  - `application/xml`, `image/png`, ...
- Steps declare what media types they consume and produce
  - Specified on the `p:input`/`p:output` level
  - Specific (`application/xml`) or wildcard (`*`)
  - XProc processor converts between media types if necessary
- XPath data model extensions

Input and Output Conversion

- Input port conversion

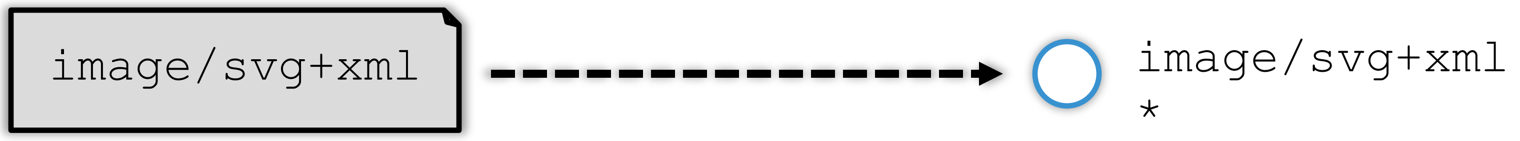


- Output port conversion

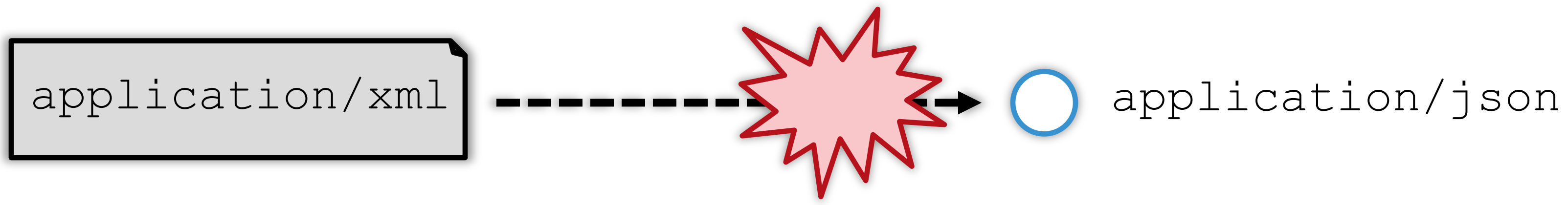


Media Type Conversion Algorithm

- The data media type matches the port media type



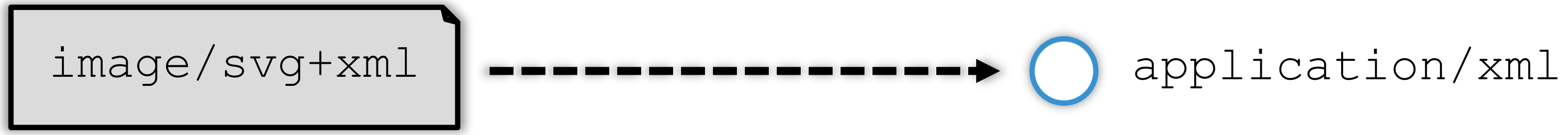
- Otherwise, if the XProc processor knows how to map from the data media type to the port media type



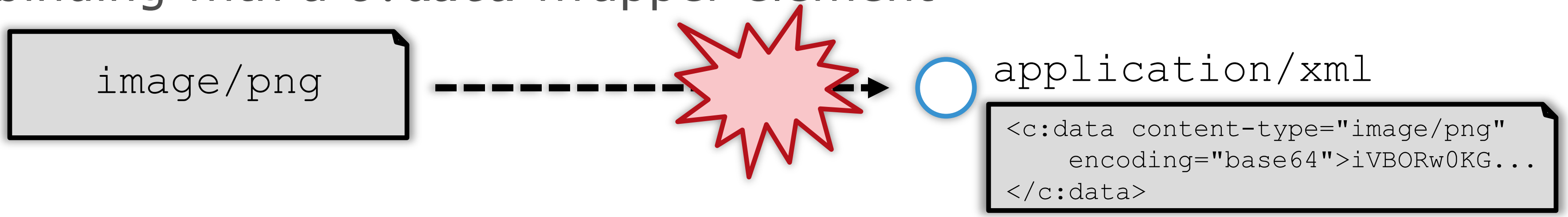
- Otherwise, fall-back

Media Type Conversion Algorithm

- Both the data and the port media types XML media types

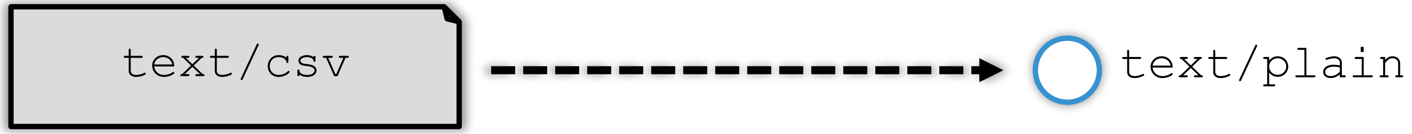


- The port media type is `application/xml` – apply `p:data` binding with a `c:data` wrapper element



```
<c:data content-type="image/png"
    encoding="base64">iVBORw0KG...
</c:data>
```

- Both the data and the port media types are text media types



- Any other combination of media types results in an error

Supported Media Types Mappings

- …implementation-defined
- Undermines interoperability
- Difficult to agree on “one size fits all” mappings that would satisfy all users or use cases
  - XML/JSON

XPath Extensions

- XPath 2.0 only
- A new property on the XDM Document Node
  - `content-type`, possibly empty
- New node type: *Binary Data Node*
  - `base-uri`, possibly empty
  - `content-type`, possibly empty
- XPath extension function
  - `m:content-type() as xs:string?`
  - `m:content-type($arg as node()?) as xs:string?`

Language Modifications: Step Declaration

```
<p:declare-step>
  <p:input port="source"
           m:content-type="application/xml"/>
  <p:output port="result" m:content-type="*"/>
  ...
</p:declare-step>
```

- Parameter input ports always accept the media type `application/xml`

Language Modifications: Bindings

- The `p:data` binding does not wrap/base64-encode unless requested
- The `m:as-content-type` attribute
  - All bindings
  - No conversion

```
<p:xquery>
  <p:input port="query">
    <p:data href="searchquery.xq"
            m:as-content-type="application/xquery"/>
  </p:input>
</p:xquery>
```

EMC²

Language Modifications: Built-in Steps

- `p:pipeline` is equivalent to:

```
<p:declare-step>
  <p:input port="source" primary="true"
           sequence="false" m:content-type="*"/>
  <p:input port="parameters" primary="true"
           kind="parameter"/>
  <p:output port="result" primary="true"
            sequence="false" m:content-type="*"/>
  ...
</p:declare-step>
```

- `p:group, p:for-each, p:choose, p:try`
  - Can be used to process any media type
- `p:viewport`
  - XML-specific

Language Modifications: Atomic Steps

- Standard XProc steps
  - `p:count`, `p:http-request`, `p:identity`, `p:sink`, `p:split-sequence`, `p:store`, `p:exec`, `p:xquery`
- `m:as-content-type`
  - A dynamic version of the `m:as-content-type` attribute

```
<p:declare-step type="m:as-content-type">
  <p:input port="source" sequence="true"
           m:content-type="*"/>
  <p:output port="result" sequence="true"
            m:content-type="*"/>
  <p:option name="content-type" required="true"/>
</p:declare-step>
```

Conclusion

- A pragmatic approach
  - Extensions to the XProc processing model as well as to the language
  - Reliance on the capabilities of the XProc processor as to what kinds of media type conversions it supports
- Too open/non-interoperable or providing just the right level of flexibility?
  - The most practical solution most likely lies somewhere in-between
- Starting point for further discussions

EMC²®