

XQuery as a *data integration language*

Assessing the potential.

Hans-Jürgen Rennau, Traveltainment GmbH
Christian Grün, BaseX GmbH
Presented at Balisage 2015, August 13, 2015




2015-08-13

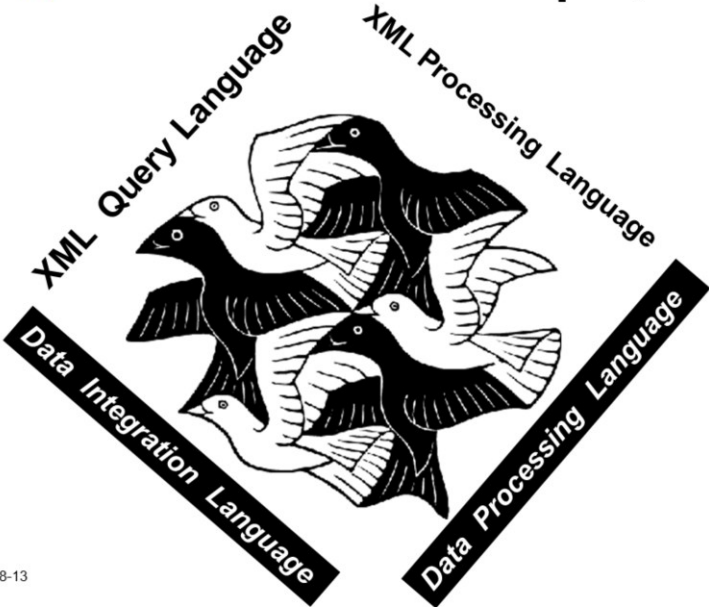
XQuery data integration

1

In spite of its name, XQuery is certainly more than a query language. But what is it? Recent extensions of the language, giving access to non-XML resources, viewed in conjunction with the typical strengths of the language, seem to open a new perspective: XQuery as a data integration language. Let us explore.



xquerY is WHAT is Xquery



2015-08-13

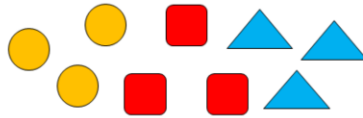
2

Well, the language is a set of facts, provided by the language specification, which is clear and precise. But the facts themselves are not yet the picture we see. The picture results from how we view the facts, how we restructure and connect them, and last not least - our imagination. Here, you might see white doves, or black ones, or even both. Similary, when looking at XQuery, it is possible to see very different things.

Data integration language ?



- Integration – facing multiplicity and heterogeneity



- Data integration language:
making integration *simpler*



2015-08-13

XQuery data integration

3

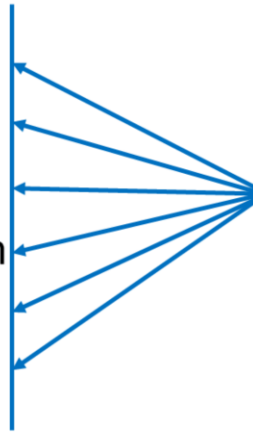
Data integration means dealing with information distributed over several, usually heterogeneous resources. Any general purpose programming language has APIs for accessing different types of resources - so every general purpose language can be used for data integration. Then why might a language be called a data integration language? The key aspect is simplicity. A data integration language enables simple solutions for typical operations of data integration.

Integration – core operations



Data ...

- Selection
- Construction
- Modification
- Transformation
- Exploration
- Validation



Data from
* multiple
* heterogeneous
resources

Integration

Some typical operations required during data integration. A data integration language should support them well especially in a context of multiple, heterogeneous resources.

Divide and conquer



How to assess the integration capabilities?

Investigation strategy

- XML integration capabilities
(capabilities in an XML-only environment)
- Access to non-XML resources
- XML integration capabilities applicable to non-XML?

We shall first investigate the integration capabilities of XQuery in a pure XML environment. Then we evaluate XQuery access to non-XML resources. Finally we ask whether the integration capabilities observed in an XML-only environment can be applied to these non-XML resources as well.

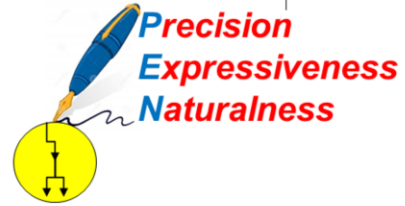
Data navigation



```
//@flightNumber
```

```
//flights  
/flight  
/@flightNumber
```

```
//flights  
/flight [arrival /@airport = 'IAD']  
      [departure/@airport = doc('/usr/data/airports.xml')  
      //airport[@cty!='US']/@code]  
/@flightNumber
```



Any understanding of XQuery and its potential uses is based on awareness of XQuery's concept of data navigation – XPath.

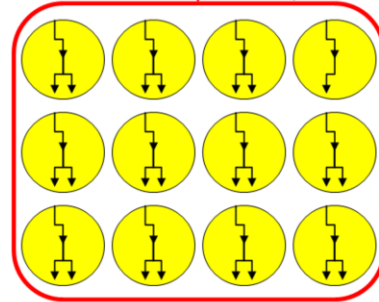
Bulk navigation



```
//@flightNumber
```

```
//flights  
/flight  
/@flightNumber
```

```
//flights  
/flight [arrival /@airport = 'IAD']  
        [departure/@airport = doc('/usr/data/airports.xml')  
          //airport[@cty!='US']/@code]  
/@flightNumber
```



2015-08-13

XQuery data integration

7

Data integration is not dealing with a single resource, but many resources. Hence our interest in bulk navigation – navigation applied to many resources simultaneously, producing an integrated result.

Bulk navigation

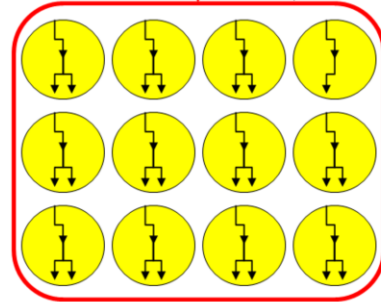
```
(doc('f1.xml'), doc('f2.xml'))  
//@flightNumber
```

document pump

```
doc('airports.xml') //href /doc(.)  
//flights  
/flight  
/@flightNumber
```

document pump

```
file:list($dir, true(), 'f*.xml') !concat($dir, '/', .) !doc(.)  
//flights  
/flight [arrival /@airport = 'IAD']  
[departure/@airport = doc('/usr/data/airports.xml')  
//airport[@cty!='US']/@code]  
/@flightNumber
```



2015-08-13

XQuery data integration

8

And here we encounter a first fact which might have an impact on our picture of XQuery. The transition from single document navigation to bulk navigation does not add any complexity to the code. You only have to add an initial step to your path expression, producing the documents to be navigated. This extended path applies the original navigation to all those documents and delivers a merged result.

Document pump – a pattern



Original: PATH

Bulk version: UFACTORY ! doc(.) ! PATH

Some expression emitting URIs

UFACTORY „name list“

`unparsed-text-lines('doclist.txt')`

UFACTORY „file list“

`file:list($dir, true(), '*.xml')!concat($dir, '/', .)`

UFACTORY „document catalog“

`doc('mycatalog.xml')//@href`

2015-08-13

XQuery data integration

9

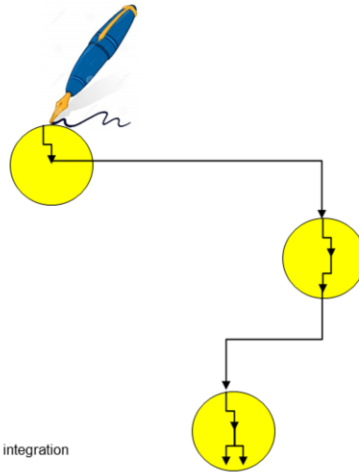
So here we have a simple design pattern how to transform single document navigation into bulk navigation: prefix the path by an expression producing URIs, which are piped into document construction, whose result is then piped into the proper navigation.

Cross-document navigation



Navigation can cross document boundaries
any number of times.

```
//flights
/flight
/termsAndConditions
/@URL
/doc(.)
//payments
/ccProvider
/@href
/doc(.)
//contact
```



2015-08-13

XQuery data integration

10

Another fact deserving attention when thinking of data integration is cross-document navigation: link traversal. A path step reaches items containing document URIs, the subsequent step leaps to the roots of the referenced documents, whence the navigation continues, drilling down into those documents.

Boundless navigation



In summary, XPath navigation is *boundless*:

- Can have starting / intermediate / end points distributed over several documents
- Can cross document boundaries at any time



The navigation model fuses all accessible documents into a **single space of information**

=

info space

2015-08-13

11

Joining these insights into a single whole – the precision and simplicity, the bulk friendliness, the irrelevance of document boundaries – a picture emerges which shows XPath navigation as boundless. And the space of this navigation – the sum total of accessible XML documents – is in turn fused by XPath into a single, homogeneous substrate, which I like to call the info space. By definition, this space is an ideal stage for data integration.

Construction ...



Data integration requires data construction –
rearranging bits of distributed information

Data construction **powered by navigation:**

- Embedded expressions
- Update facility – combines navigation / modification

The immediate effect of data navigation is data selection. But data integration needs more than data selection – distributed information must not only be selected, but reassembled into new entities. Construction is as essential a part of integration as selection.

... powered by navigation



```
<routes>

  <national>{
    //flight[not(addInfo/@international)]
    /<route>{@*, * except addInfo}</route>
  }</national>

  <international>{
    //flight[addInfo/@international]
    /<route>{@*, * except addInfo}</route>
  }</international>

</routes>
```

NAV#1 => data source

NAV#2 => new contents

In this example, different navigations are used to select data sources (flight elements) and new element contents, which are a filtered copy of the data source contents. The effort of construction is minimal, thanks to navigation.



Modification

- Special case of construction: modification
- XQuery Update Facility
- Modification **powered by navigation**

Update Expressions:

- Sub expression #1: *selects* the target node
- Sub expression #2: *changes* the target node

It is worthwhile to consider a special case of construction – modification. The XQuery Update Facility enables an elegant combination of navigation and change: an independent sub expression selects the target node, another expression provides the new value to be inserted, which more often than not is in turn expressed by some navigation - for example into a configuration file.

Bulk modification



document pump

```
doc('airlines.xml') //flights/@href /doc(.)
```

navigator

```
//flights  
/flight [arrival /@airport = 'IAD']  
      [departure/@airport = doc('/usr/data/airports.xml')  
        //airport[@cty!='US']/@code]
```

modifier

```
/(insert node attribute international {true()}) into addInfo)
```

Let us look at an example of bulk modification. The first line pumps documents into the navigation to flight elements which describe a flight from a non-US airport to Washington. The last line modifies the contents of these flight elements, inserting an „international“ attribute into its addInfo child element.

age

. 10

. 17

. 18

. 19

. 20

antigenSequence

. NSFKNNYEKALKQYNSTGDYRSHAVDKIQNTLHCCGVTDYRDWTD ...

cellLine

. A-431

. A549

. AN3-CA

. BEWO

. CACO-2

cellType

. Langerhans


. Leydig cells

. Purkinje cells

. adipocytes

. bile duct cells

Resource exploration



Example output:
names and values of simple-content elements

Resource exploration is an important activity when attempting data integration. It is about gaining insight about resource contents. XQuery is a great tool for such tasks. For example, when confronted with an unknown type of XML documents, you may want to learn the names and sampled contents of its data elements.

Resource exploration



```
declare variable $limit as xs:int external := 5;
```

Collect all data elements

```
for $elem in /**[not(*)][text()]  
group by $ename := local-name($elem)
```

... group them by name

```
let $values :=  
  sort(distinct-values($elem))  
  [position() le $limit]
```

... extract values

```
order by $ename  
return  
  ($ename, $values!concat(' ' , .), '')  
)
```

Example code:
names and values of simple-content elements

A few lines of XQuery code give you the answer. It goes without saying that transition to bulk exploration is just a couple of code lines away.

Validation



- Data integration often requires data validation:
 - Grammar-based – e.g. XSD validation
 - Rule-based – e.g. Schematron
- Presently, only **extension functions** available
- XQuery assets:
 - **Bulk validation**
 - **Expression-based validation**

Validation is important, and it is very important in the context of data integration. It is a pity that there are not yet standard functions available providing XSD and other validation. Fortunately, popular XQuery processors offer extension functions for validation. What XQuery has to offer is excellent support for bulk validation and expression-based validation.



Bulk validation

```
let $xsds := UFACTORY1 ! doc(.)
let $docs := UFACTORY2 ! doc(.)
let $reports :=
  for $doc in $docs
  let $name := $doc/local-name(*)
  let $namespace := $doc/namespace-uri(*)
  let $xsd :=
    $xsds[$name = */xs:element/@name
          [$namespace = string(*/@targetNamespace)]
  let $msgs := validate:xsd-info($doc, $xsd)
  return
    <report doc="{document-uri($doc)}">{
      $msgs ! <msg>{.}</msg>
    }</report>
return
  <reports>{$reports}</reports>
```

Simple bulk navigation picks the appropriate XSD

19

Interesting in the context of data integration is bulk validation: validating a heterogeneous set of instance documents against a collection of XSDs, letting the bulk validator sort out for each instance document which schema to pick. This is a trivial task of bulk navigation – inspect the name and namespace of the instance document root, and pick the schema with matching target namespace and containing a top-level element declaration with a matching name.

Expression-based validation



```
<rules>
  <rule msg="No product found."
        expr="empty(//*:Product)"/>
  <rule msg="Travel start date > end date."
        expr="//travel[@start gt @end]"/>
</rules>
```

Rule descriptors
(expression + message)

```
<report>{
  let $docs := UFACTORY ! doc(.)
  for $doc in $docs return
    <errors uri="{document-uri($doc)}">{
      doc('rules.xml')
      //rule[xquery:eval(@expr, map{'':$doc})]
      /<error msg="{@msg}"/>
    }</errors>
}</report>
```

Error list produced by a
navigation expression !

20

Grammar-based validation cannot check business rules. XQuery expressions are an ideal way of both defining and checking a rule. This example shows a simple generic rule checker. The validation is essentially executed by navigation: visit the rule descriptors, filter them by applying their expression to the target document, and emit the associated message. Note that this code uses a vendor-defined extension function for the dynamic evaluation of XQuery expressions found in the configuration.

XML data integration



XQuery **excellently suited**

- Language foundation: boundless **navigation**
- Data **selection** – implemented by navigation
- Data **construction** - powered by navigation
- Data **modification** - powered by navigation
- **Transformation** – supported by navigation
- Resource **exploration** supported by navigation
- Resource **validation** supported by navigation

Ok – dealing with an XML-only environment, XQuery is excellently suited for data integration – thanks to a language structure laying a groundwork of boundless data navigation and using it to power construction and modification. Navigation also provides the programmer with strong support for transformation, exploration and validation.

XQuery - the Great Transition



- Proved: XQuery suitable for XML integration
- Q: suitable for data integration in general?
 - XQuery 1.0: only XML
 - XQuery 3.0: add plain text resources
 - XQuery 3.1: add JSON
 - EXPath, vendor-specific extension functions:
HTML, CSV, SQL, HTTP POST, archives, RDF

Take a deep breath – our first result is that XQuery is a great tool for XML integration. Recent versions of the language (3.0, 3.1) added access to non-XML resources – plain text and JSON. And in the mean time popular XQuery processors already provided access to numerous other resource types. So let us ask: is XQuery suitable for data integration in general?



Resource access

- Resource **representation** = XDM value
Examples: node, map, array, string
- Resource **access**: an XQuery function
returning a resource representation

Examples:

doc (URI)

unparsed-text (URI)

json-doc (URI)

2015-08-13

23

Let us first consider the concept of resource access. In XQuery, resource contents must be represented by an XDM value. Resource access is provided by an XQuery function returning a resource representation.

Resource access patterns



- **Input (resource identification)**

- Access by URI `doc (URI)`
- Access by text `parse-json (string)`
- Access by message `http:send-request (msg)`
- Access by query `sql:execute (query)`

- **Output (resource representation)**

- XML node tree `json-to-xml (string)`
- map/array tree `parse-json (string)`
- string `unparsed-text (URI)`

2015-08-13

XQuery data integration

24

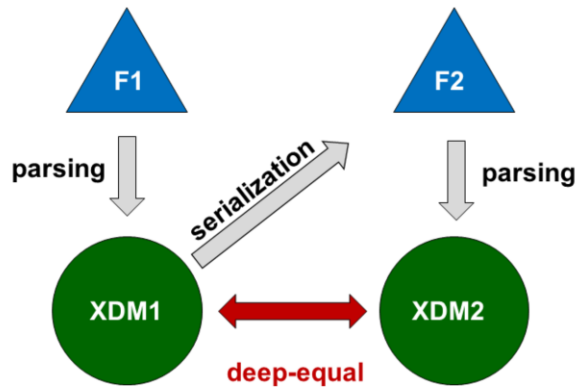
We should be aware of various access patterns. Resource contents are delivered in exchange to a resource URI, or to the resource text itself, or a request message, or a database query. The resource representation can be an XML node tree, a tree of nested maps/arrays, or simply a string.

Resource representation - formally



DEFINITION. XDM binding

- Parsing function P
- Serialization function S
- Constraint:

$$\text{deep-equal}(P(F), P(S(P(F))))$$


2015-08-13

XQuery data integration

25

If data integration becomes a strategic goal, we may need new concepts defining the relationships between XDM values and resource instances, expressing constraints on information loss and the effects of round-tripping. Here comes a proposal for a formal definition of resource representation.

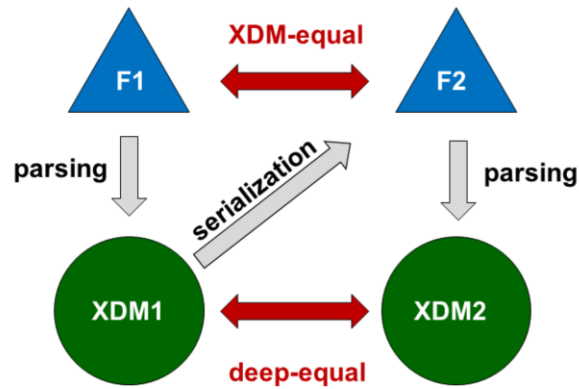
Resource representation - formally



DEFINITION. XDM equal

Two format instances
F1, F2 are XDM equal

\Leftrightarrow
 $\text{deep-equal}(P(F1), P(F2))$

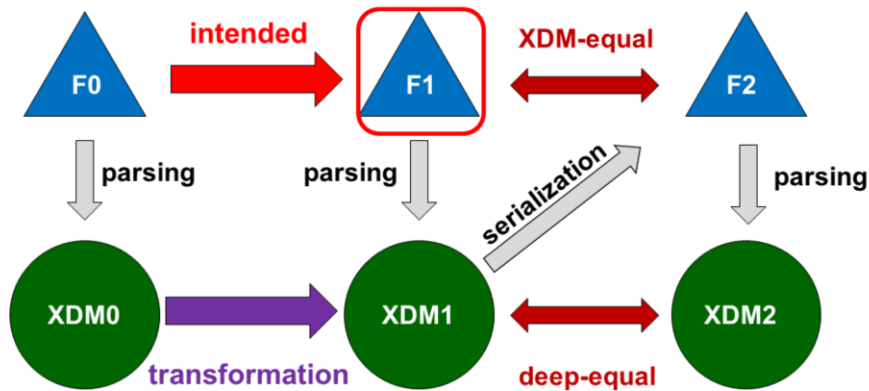


XDM bindings create a formal bridge between format instances and XDM values. Now we can compare format instances by comparing the associated XDM values

Non-XML transformation



format instances



XDM values

2015-08-13

XQuery data integration

27

And now we can define the transformation between format instances in terms of XDM manipulations. A transformation between two format instances can be accomplished as the transformation between the corresponding XDM values as defined by a particular XDM binding. The result of the XDM transformation can be serialized into a format instance which is guaranteed to be XDM equal to the intended end result.



XML binding

[DEFINITION. [XML binding](#)]

An XDM binding whose parsing function returns an **XML node tree**.

Nota bene.

XQuery's navigation capabilities apply to XML node trees – not to map/array trees.

XML binding is just a special case of XDM binding – the XDM representation is an XML node tree. This is an important aspect as XPath navigation can only be applied to node trees, not to map/array trees.



XDM binding - examples

- Format: JSON

- Three XDM bindings:

Bindings identified by a URI
(namespace#name of parsing function)

- (1) json#parse (XDM=XML) (BaseX)
- (2) fn#json-to-xml (XDM=XML) (standard)
- (3) fn#json-doc (XDM=map/array) (standard)

- Q: How to navigate à la: `//booking/bookingID`

2015-08-13

XQuery data integration

29

Let us consider examples of XDM binding. Dealing with JSON resources and using the BaseX processor, three XDM bindings are available. The first is given by a BaseX-defined parsing function, which produces a BaseX-defined XML-representation. The second is given by a standard function, which produces a W3C-defined XML representation. And the third is given by a standard function returning a map/array representation. Now let us ask how to perform a navigation which in an XML context would be expressed by `//booking/bookingID`.

Navigation - XML binding



- Case 1: json:parse (XDM=XML (BaseX))

```
//booking/(.,_)/bookingID
```



- Case 2: fn:json-to-xml (XDM=XML (W3C))

```
//*[@key eq 'booking']  
/descendant-or-self::j:map[1]  
//*[@key eq 'bookingID']
```



30

In the case of an XML representation the navigation is accomplished by a more or less simple path expression. The expression varies, dependent on which XML representation is used – the BaseX-defined representation, or the W3C-defined representation.

Navigation - non-XML binding



- Case 3: fn:json-doc (XDM=map/array)

```
declare function local:descendant($item as item(), $fieldName as xs:string) {  
  typeswitch($item)  
  case map(*) return (  
    let $field := $item($fieldName)  
    return  
    if ($field instance of array(*)) then $field?* else $field,  
    map:keys($item)[. ne $fieldName] !  
    local:descendant($item(.), $fieldName)  
  )  
  case array(*) return $item?* ! local:descendant(., $fieldName)  
  default return ()  
};  
local:descendant($data, 'booking')?bookingID
```



31

If the resource is represented by nested maps and arrays, the navigation can only be accomplished by writing complex code. This is not navigation any more – this is just a piece of tough programming work. Sorry - the XDM value obtained for the JSON document is not in scope of XQuery's outstanding navigation capabilities.

Map/array navigation issues



Primitive navigation: `a?b?c`

- Only child axis – no deep, no upward navigation
 - No deep navigation `no //foo`
 - No upward navigation `no ancestor::booking`
- Result heterogeneous `(map, array, string, ...)`
- Result anonymous `no $res/name()`
- Knowledge of array-steps required

`flights?*>departure?airport`

- No modification `try $j?a?b!map:put(., 'c', 'X')`

Map/array trees can be navigated in a primitive way, using the lookup operator (?). It is important to realize that this navigation simply cannot be compared with XPath navigation.



XML-binding ⇔ integration

- XML-binding **available**
 - => format within scope of XPath navigation
 - => within scope of integration capabilities
- XML-binding **not available**
 - => format not within scope of XPath navigation
 - => *out of scope of integration capabilities ?*
... or *conditionally within scope ?*

White doves ?

or both ?

Black doves ?

If for a given format F an XML binding exists, it is fully within scope of XQuery's navigation capabilities. This seems to imply that the format is within scope of XQuery's integration capabilities. But what if only a map/array binding is available? Certainly, the format is not within scope of XQuery's navigation capabilities, and as a rule the ease of integration is reduced. But to which degree? Does the format fall out of integration, or is the potential of integration by an large preserved? The answer also depends on the complexity of the format instances – the less complex, the smaller the disadvantage of having to do without strong navigation. The answer might be similar to the answer whether Escher's graphic shows white or black doves.

Wrapping up



- Everything within reach of XQuery's navigation capabilities is within reach of easy construction, modification, transformation and aggregation => within reach of integration.
- To consider: extend access to non-XML resources systematically (HTML, CSV, SQL, archives, ...). Make sure to support XML bindings whenever possible.

2015-08-13

XQuery data integration

34

Wrapping up, XQuery's capabilities of integration are tightly coupled to its capabilities of navigation. If the XQuery Working Group should identify data integration as a strategic goal of XQuery, non-XML resource access would need to be extended systematically, and a strong emphasis on XML bindings would be important.

XQuery spec – first sentences

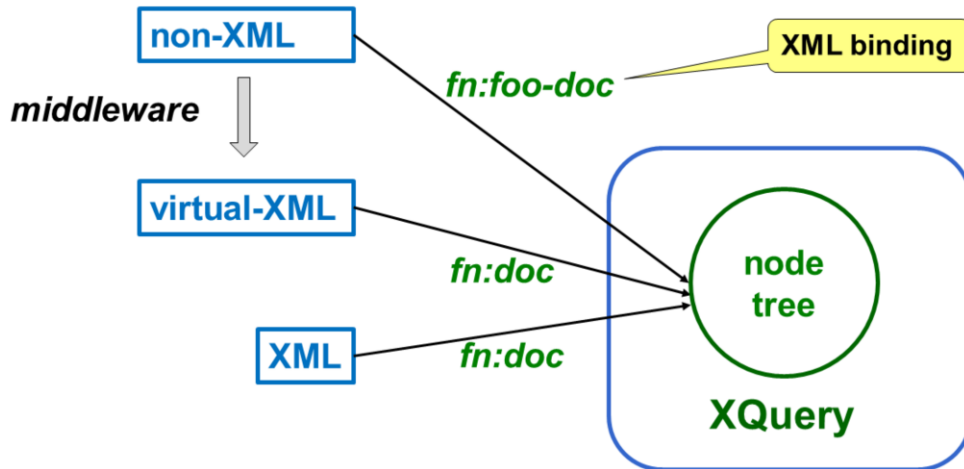


... A query language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware.

This specification describes a query language ... designed to be broadly applicable across many types of XML data sources.

Coming to an end, I return to the beginning of XQuery, the first sentences of the spec... These sentences are about data integration. They envision XML as a unified view on many kinds of data, a view made available by middleware services creating XML on the fly. This picture turns XQuery into a data integration language, which, however, depends on middleware – infrastructure outside of the language. But by now the acceptance of XML is decreasing and such middleware cannot be taken for granted.

Integration via XML binding



2015-08-13

XQuery data integration

36

But the vision of XQuery as a data integration language might be preserved by introducing a second way how non-XML data might become node trees: not only by middleware services, but by parsing functions, those of XML bindings.

A conceivable extension



... A query language that uses the structure of XML intelligently can express **queries across all these kinds of data**, whether physically stored in XML or **viewed as XML via middleware, or bound to a node tree representation within XQuery itself.**

This specification describes a query language ... designed to be broadly applicable across many types of **XML and XML-bindable data sources.**

So... Let's edit the text a little...

Thank you for looking!



... and call it a day (or night) – thank you for listening.