IBM Research

Phantom XML

- if you look too hard it isn't there

Kristoffer H. Rose
Lionel Villard

Overview

- **Motivation**
- **Phantomization**
- **XML Processing**
- **Experiments**
- **Conclusion & perspectives**

Motivation

- **A lot of heterogeneous data formats out there**
  - Relational, binary, semi-structured, structured, etc...
  - All the formats can be converted to XML
- **XML processing ubiquitous ...**
- **... but XML syntax by itself is "inefficient"**
  - Verbose
  - No indexes
- **Sometimes access to actual XML *data* is *only* through XPath**

Why XML (or a semi-structured format)?

- **Why not relational?**
  - Difficult to represent some data in unordered indexed tables
  - Shredding is complex and in practice lossy (loss of element order, loss of whitespace, etc..)
  - Reconstructing original XML involves complex joins (that the programmer needs to write!)
- **Why not graph?**
  - No good way to serialize graphs
  - No clear scoping
- **Why not ...**

Existing solutions for XML conversion

- **Batch conversion**
  - Good ...
  - but legacy applications depending on the old format must be rewritten...
  - ...and converted data can be big!
- **On the fly conversion**
  - Good too ...
  - but the standard XML format requires significant overhead for generating or parsing XML character sequences

Existing solutions for XML processing

- **Batch processing**
  - Really good solutions for small documents,
  - Highly optimized streaming processors exist,
  - but need batch conversion of the input
- **Embedded processing**
  - JXPath operates on custom in-memory object structures,
  - DOM3/JAXP compiles XPath for full in-memory XML data model instances,
  - but hard to minimize the memory use

Goals

- **Execute XML processing programs without XML materialization**
  - Convert “on-demand” to XML data model
  - Do not even access parts of data not needed for conversion
  - Processing with XPath but also XSLT, XQuery, ...
  - Allow creation and updates
- **Allow such processing for any kind of structured and semi-structured data**
  - Respect access/update pattern restrictions
  - Exploit optimal access/update patterns

Phantomization

“Just don't build the XML”

Virtualized XML Data Model: Focus

- **Our implementation of the XQuery/XPath data model**
  - XPath/XSLT 2.0 and XQuery defined on top of this (W3C CR)
  - One interface called Focus
- **Cursor-based model**
  - *The application never has a handle on the data!*
  - *Enable smart conversion and memory management*

Feature

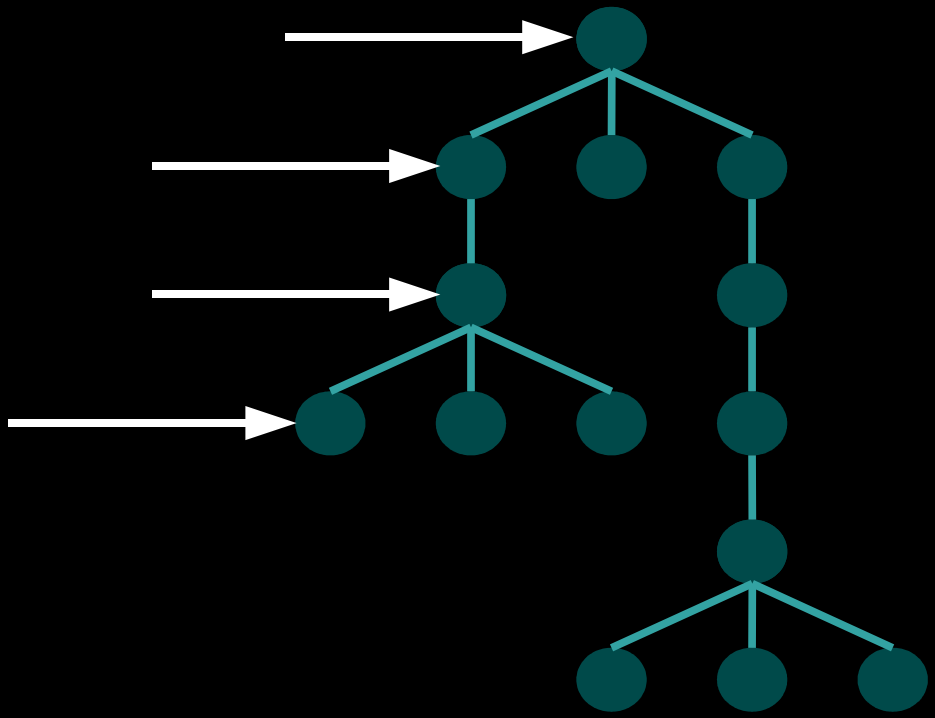
- A *feature* allows one particular access or update operation (or simple pattern) on the data
  - Access node property (*getName*(), *getValue*(), *getType*(), etc...)
  - Navigation (*toChildren*(), *toAttribute*(), *toParent*(), etc...)
  - Cursor management (*duplicate*(), *free*())
  - Mutation (*setValue*(), *addAttribute*(), *addElement*(), etc...)
- Features are dynamic
  - features()
  - duplicate(requestedFeatures)

Profile

- A *profile* identifies a specific usage pattern and is characterized by a set of features
  - Streaming input: depth-first tree traversal (grammar)
  - Forward-only: reserve axis (*toParent()*, ...) not allowed
  - Full: all features allowed
  - Streaming output: depth-first tree construction
  - Forward-only output: only append
  - ...

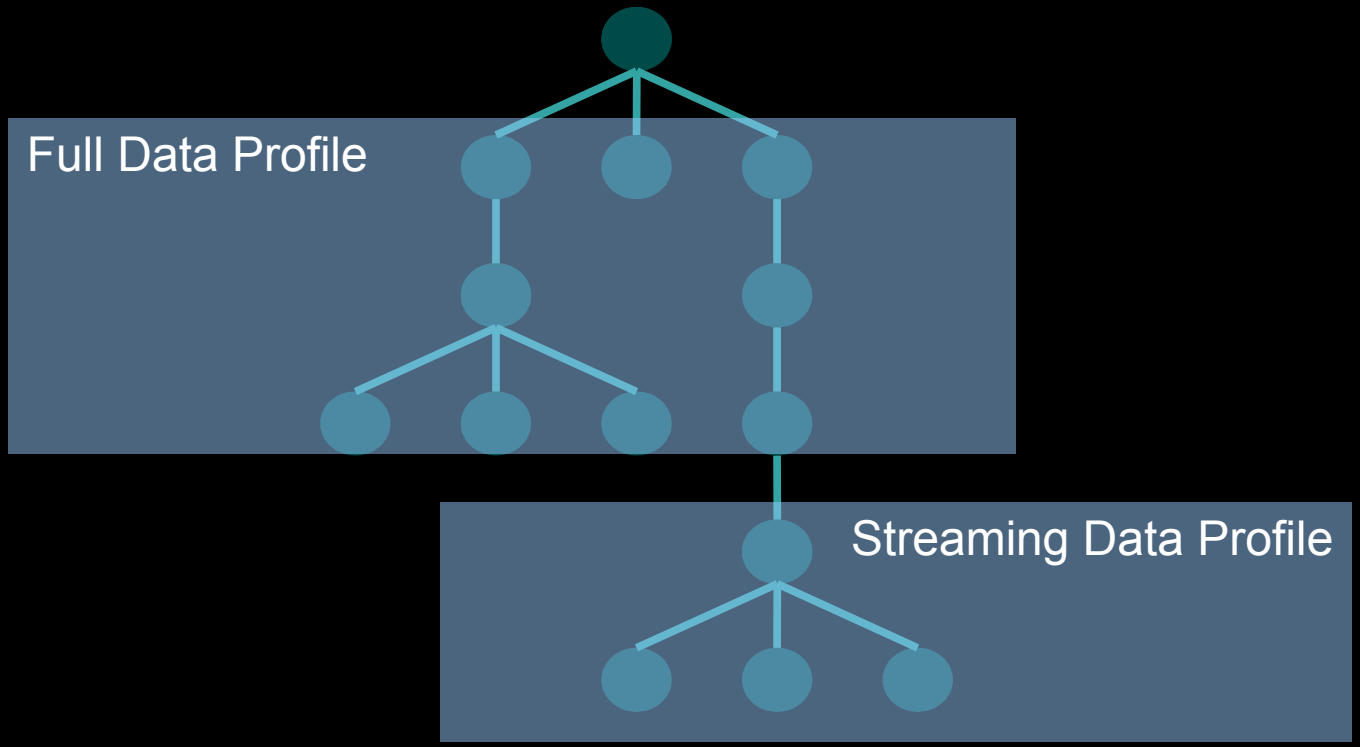
Forward only profile

- **Define a window on the data**



Dynamic profiles

- **Profiles can be changed at any time**



Processing

“Be lazy!”

XPath engine for virtual XML – Static time

- **"Classic" XPath 2.0/XQuery 1.0 compiler**
  - Parsing => Normalizing (Core XQuery) => Optimizing
- **In the context on Phantom XML**
  - Minimize the number of features needed (rewriting)
  - Determine which features are needed for processing a given XPath (static analysis)
  - Dynamically reduce the number of features needed

Assumptions: no cost model and huge (even infinite) input documents

Minimizing needed features

- **Based on rewriting techniques**
- **Example: Forward-only transformation**
  - Remove reverse axis => use the forward only profile
  - Make explicit fragments of document which need to be cached (variable)
  - See “Compiling XPath into a State-less Forward-only Subset”
- **Example: distinct-doc-order() function removal**
  - Avoid duplicates removal and sorting
  - See “Optimizing Sorting and Duplicate Elimination in XQuery Path Expressions”
- **Schema-based rewriting**

Schema rewriting

//(africa|europe)/@id

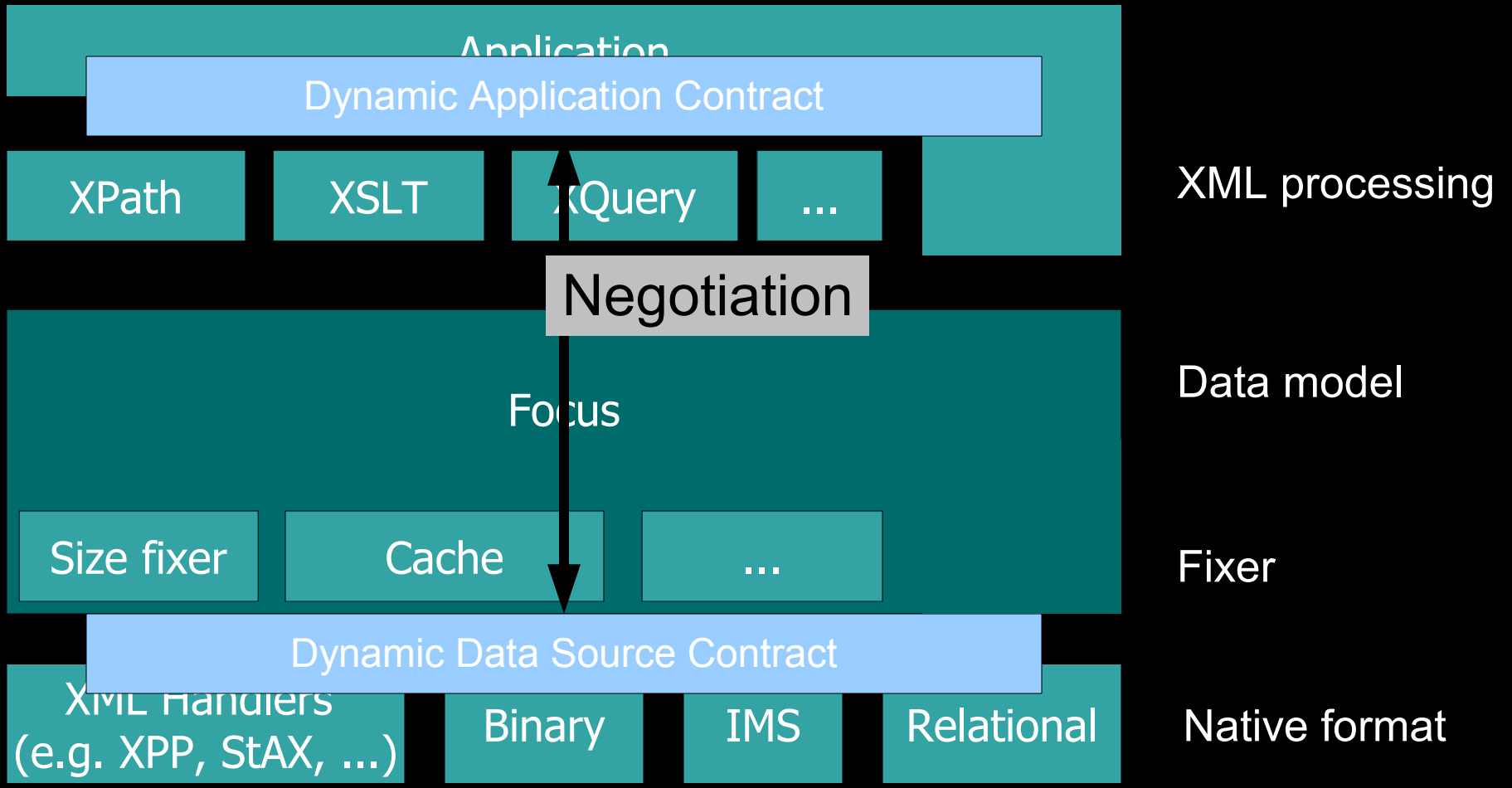
Schema: *africa* occurs only once and before *europe*

//*simple-union*(africa, europe)/@id

At runtime

- **Pull processing: otherwise entire conversion is done**
- **Lazy processing: minimize cursor duplications, caching**
- **Determine which profile to use for each duplicate**
  - The XPath itself (static feature analysis)
  - The requirements of the application

Runtime architecture

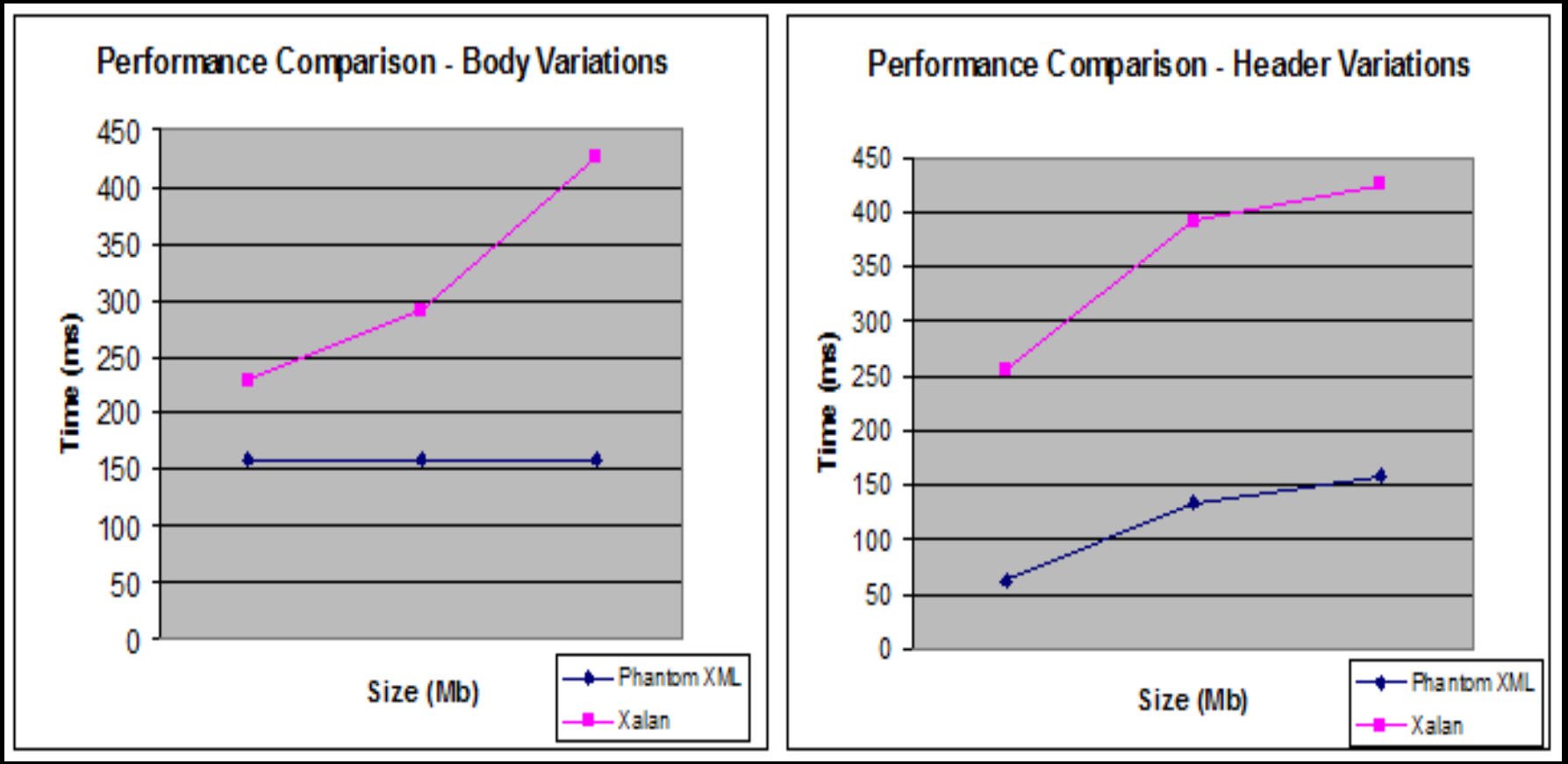


Experiment

```
v:parse(v:decode(
    v:unzip('examples.sxw')/
       zip/entry[@name eq 'content.xml']/bytes,
    'UTF-8'
   )
)/*/body[1]/h[3]/text()
```

```
<office:document-content
                xmlns:office="http://openoffice.org/2000/office" ...>
  ...
  <office:body>
         <office:h> My header </office:h>
    ...
```

Measurements



Conclusion

- XPath/XQuery data model implementation
  - Lightweight: cursor-based
  - Adaptive: feature-based
  - Easy integration of foreign data format: feature completion
- Efficient lazy XPath processor over any kind of data
- Allows dynamic optimizations ala JIT
- No inherent limit on document size
- Try it out on alphaworks: www.alphaworks.ibm.com/tech/virtualxml

Some perspectives

- **Richer feature set**
- **Identifying more used profiles**
- **More native data formats**
  - DFDL: generic Data Format Description Language
  - Relational database
  - EXIF (JPEG, MPEG, etc...)
  - ...
- **Cost-model**

www.alphaworks.ibm.com/tech/virtualxml

Backup

Streaming input profile

Document ::= toChildren Sibling free
Sibling ::= getName? (getValue | Attributes Children)
Attributes ::= duplicate toAttributes (Attribute-traversal)? free
Attribute-traversal ::= (getName? getValue? toNext)+
Children ::= duplicate toChildren (Sibling-traversal)? free
Sibling-traversal ::= (Sibling toNext)+

Streaming output

```
Document ::= add-first-child-element Content free
Content ::= (add-attribute)* (First to-children (Following to-next)* to-parent)?
First ::= add-first-child-text
        | add-first-child-comment
        | add-first-child-processing-instruction
        | add-first-child-element Content
Following ::= add-following-sibling-text
            | add-following-sibling-comment
            | add-following-sibling-processing-instruction
            | add-following-sibling-element Content
```