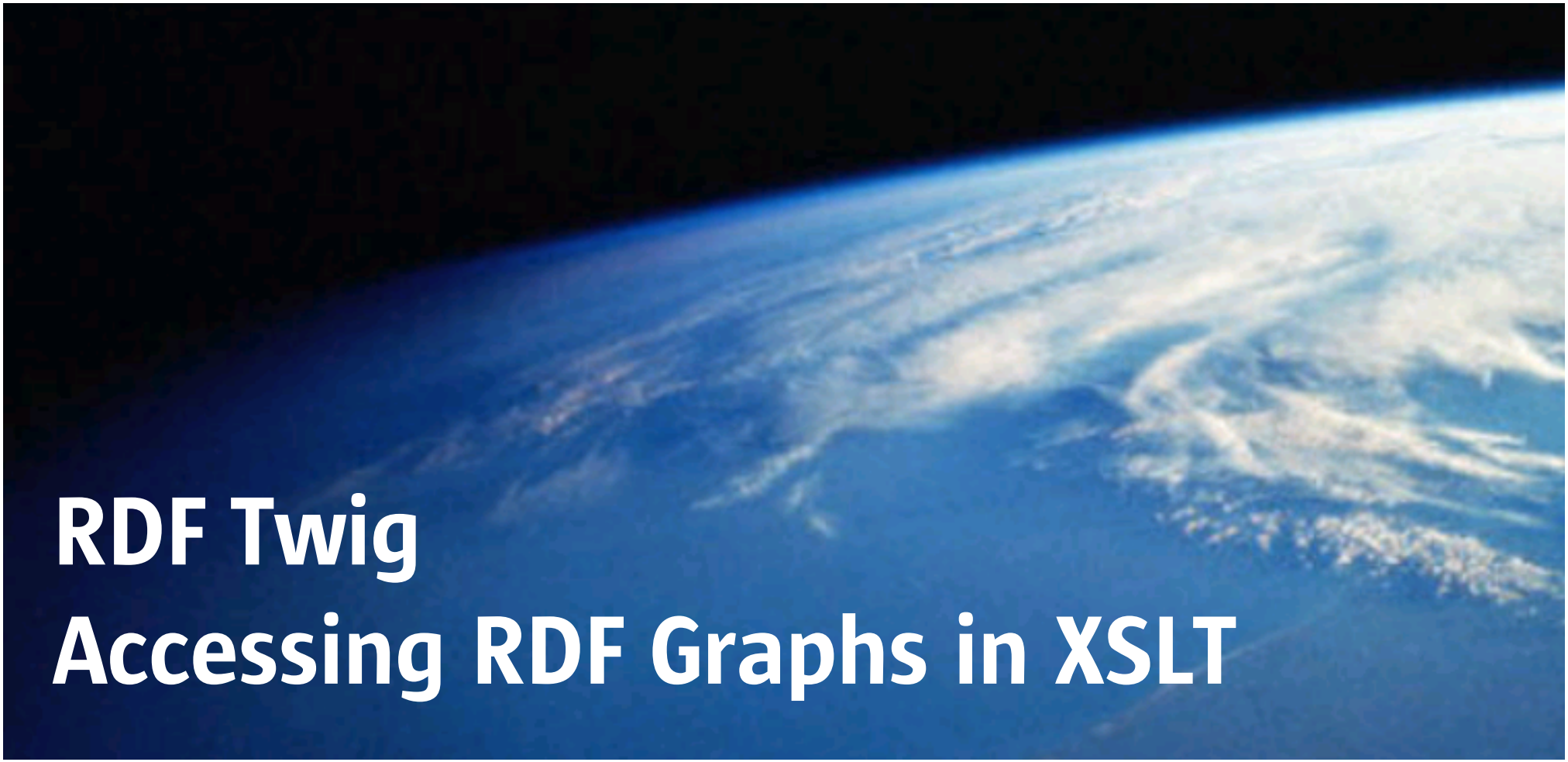


RDF Twig
Accessing RDF Graphs in XSLT

Norman Walsh

Extreme Markup Languages
04 - 08 August, 2003

http://www.sun.com/

Sun microsystems
We make the net work.

Version 1.0

Introduction

“There's nothing as practical as a good theory”

Introduction

“There's nothing as practical as a good theory.”

“There's nothing as theoretical as good practice.”

Introduction

“There's nothing as practical as a good theory.”

“There's nothing as theoretical as good practice.”

“Sometimes a practical solution is good enough.”

Observations

- RDF is a useful way to store and process information that fits into the RDF paradigm.
- Lots of information does fit into that paradigm.
- RDF can be serialized in XML.
- XSLT is a useful way to process XML.

But...

- Processing RDF with XSLT is difficult and tedious.

The Problem

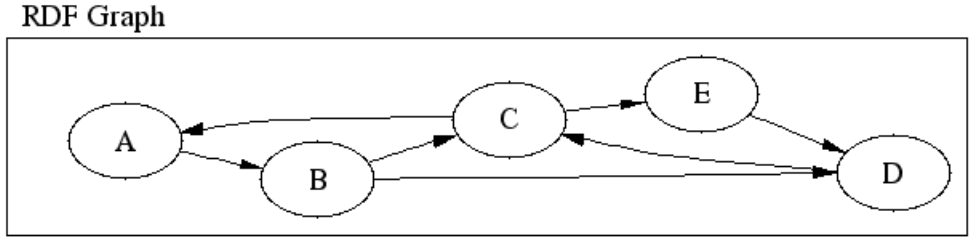
- XSLT (and XPath) are designed to operate on XML documents. XML documents are trees.
- A collection of RDF statements is a directed graph, but it is not generally a tree.
- Templates designed to transform RDF often stumble over this missmatch at the data model level.
- But RDF has an XML serialization, doesn't it?

What about RDF Serialization?

- Nodes in a tree have only one parent.
- Nodes in a graph may have several “parents”.
- If node identity is to be preserved:
  - Nodes must be treated in two different ways.
    - It boils down to: instantiate once.
    - Reference elsewhere.

Serialization Example

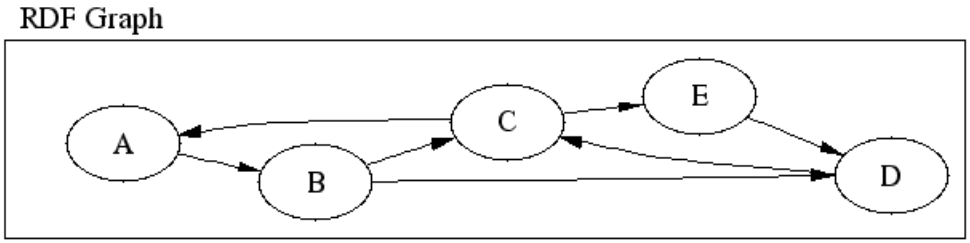
Consider this small graph:



And how it might be serialized:

```
<A>
   <B>
      <C>
         ??? What do you do about A?
```

Serialization Example (Continued)




```
<A node="n1">
  <B node="n2">
    <C node="n3">
      <A node="n1"/>
      <E node="n4">
        ...
```

- Graph: B/C/A/* = B
- Tree: B/C/A/* = empty node set

Working With Serialized RDF

- Know your serialization tool. There are several flavors and recent RDF changes introduce at least one more.
- Use keys and conditional logic in your templates to identify and correctly process nodes that are inline and nodes that are referenced.
- In the general case, you need a choose statement for each node, one to test for `@rdf:resource` and one to test for `@rdf:about`.

Difficult and tedious.

There's More Than One Way To Do It

There's no single, right way to do the serialization.

- **Any node could be the “root” of the tree.**
- **Nodes must be instantiated exactly once.**
- **Which nodes are “new” and which are “duplicates” depends on where you start and how you build the tree.**

RDF Twig

- Let's you start at any node in the RDF graph.
- Builds a serialized representation of that part of the graph (with a few user-tuneable parameters).
- Returns the tree as a document so that you can apply XSLT to it.

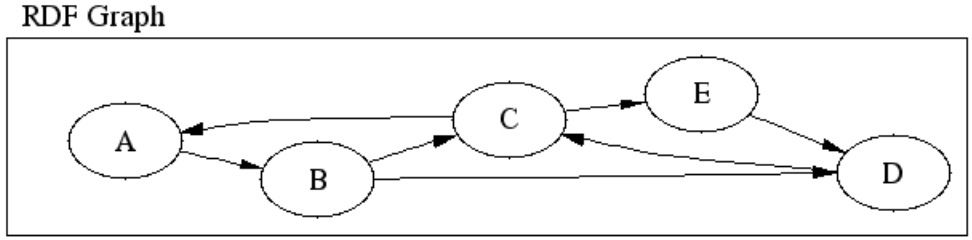
In short, RDF Twig lets you serialize interesting parts of the graph on the fly.

RDF Twig Implementation

- RDF Twig is implemented as a set of (Java) XSLT extension functions and elements.
- The current implementation is built on top of the Jena RDF toolkit.

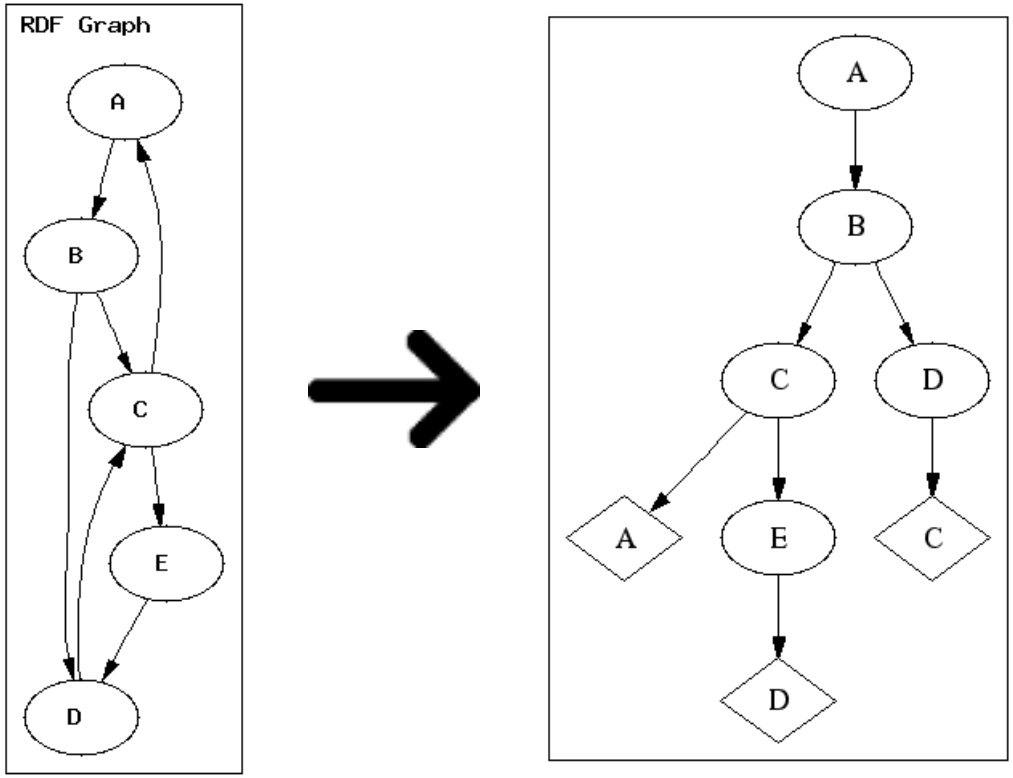
How to Serialize

Consider this graph:



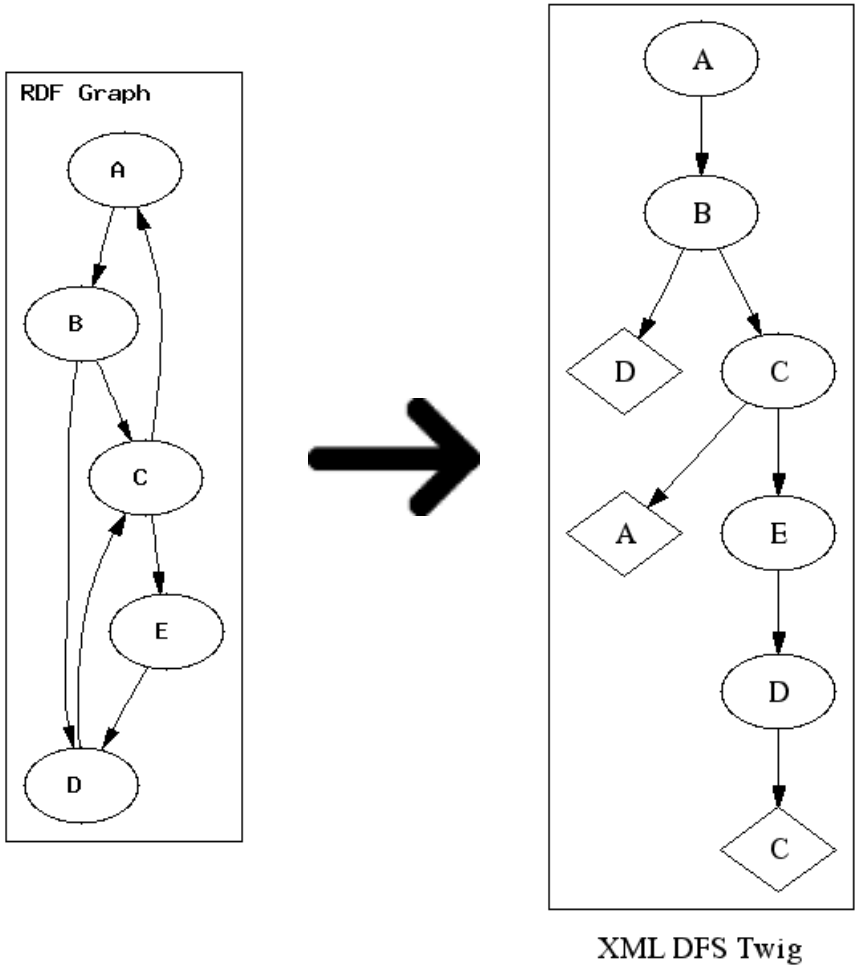
How can this be serialized (starting at A)?

Serialize Breadth First



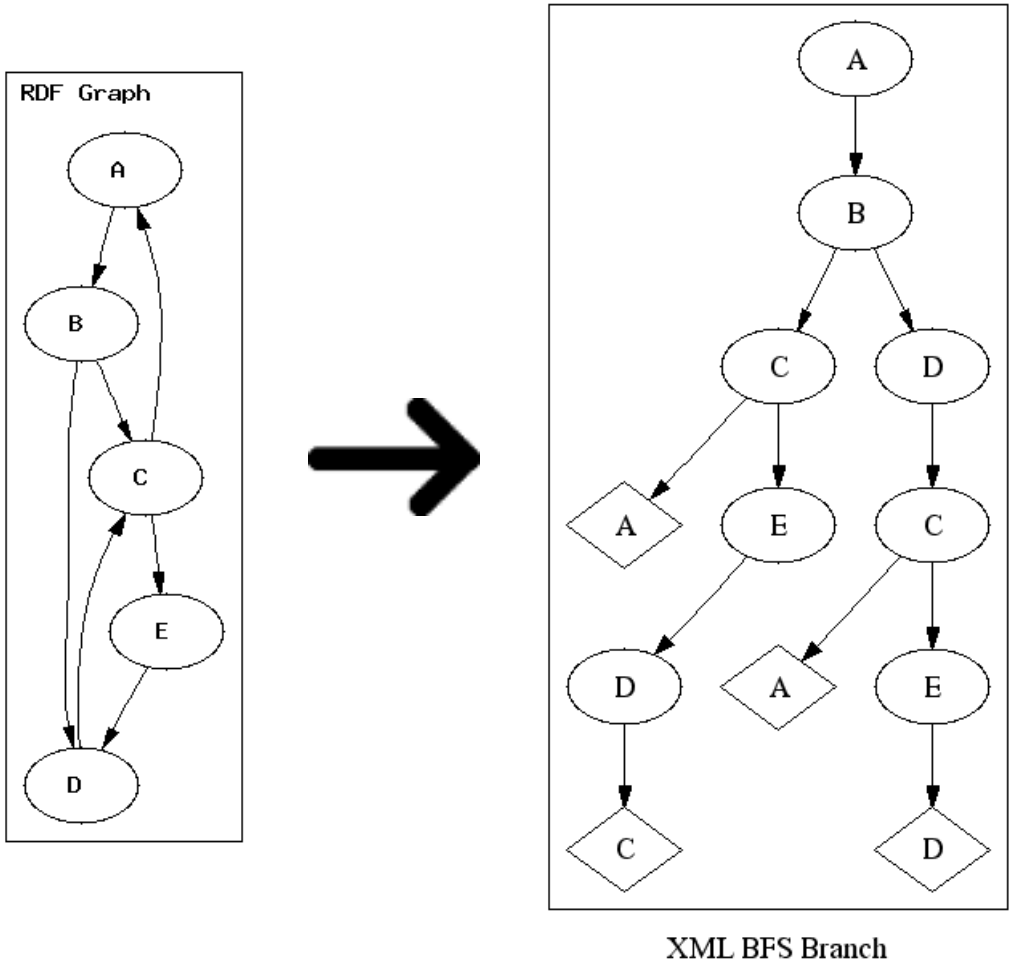
XML BFS Twig

Serialize Depth First



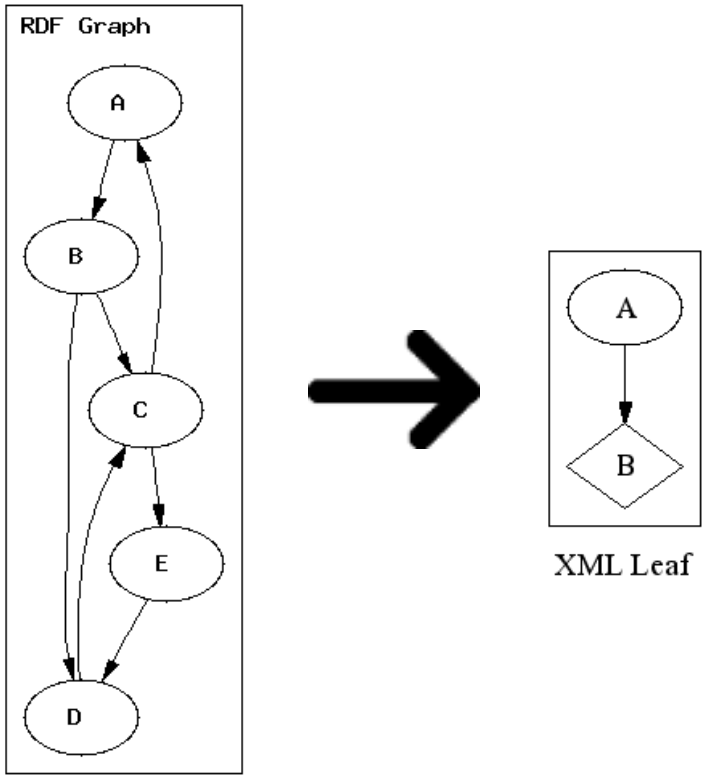
XML DFS Twig

Serialize Breadth First Deep



XML BFS Branch

Serialize a Leaf



RDF Twig Terminology

twig	A shallow breadth-first tree.
dftwig	A shallow depth-first tree.
branch	A deep tree.
leaf	A “tree” with no instantiated children.

RDF Twig in Action

Load the model:

```
<xsl:variable name="model"
  select="rt:load('diagrams/bgraph.rdf')"/>
```

Grab a node:

```
<xsl:variable name="A"
  select="rt:resource('http://uri/for/A')"/>
```

Turn the results into a tree:

```
<xsl:variable name="tree" select="rt:twig($A)"/
```

RDF Twig in Action (Continued)

At this point, `$tree` contains an XML document that can be queried and transformed with XSLT like any other input document.

RDF Twig in Action

Construct a property:

```
<xsl:variable name="label"
  select="rt:property('http://example.com/graph
```

Find some nodes:

```
<xsl:variable name="findResults"
  select="rt:find($label, 'D')"/>
```

Turn the results into a tree:

```
<xsl:variable name="tree"
  select="rt:twig($findResults)/twig:result"/>
```

RDF Twig Functions

- `load()` (and `store()`) RDF graphs.
- `resource()` gets (or creates) a single resource.
- `property()` gets (or creates) a property.
- `twig()`, `dftwig()`, `branch()`, `leaf()` get parts of a graph.

RDF Twig Functions (Continued)

- `find()` finds resources (that have a property).
- `get()` finds resources (that are a property).
- `filter()`, `filterNot()` trim a set of resources.
- `union()`, `intersection()`, `difference()` perform the obvious boolean operations on sets of resources.

RDQL Support

RDF Twig now supports RDQL:

```
<xsl:variable name="a">
  <rq:rdql return="a">
SELECT ?a, ?b
WHERE  (?a, &lt;http://somewhere/pred1&gt;, ?b)
AND    ?b < 5
  </rq:rdql>
</xsl:variable>
```

This is a result tree, so you need a `node-set` extension to access it.

Isn't There a Better Way?

Wouldn't it be better to extend XPath (XSLT?) to operate over graphs?

Yes, probably. But RDF Twig satisfies an immediate need: to access RDF graphs in XSLT stylesheets today.

A “Real” Example

```
<xsl:variable name="contactType"
     select="rt:resource('http://nwalsh.com/rdf/p

<xsl:variable name="allContacts"
     select="rt:twig(rt:find($rdf:type,
                            $contactType),1)/twig:result

  ...

<xsl:for-each select="$allContacts">
  <xsl:apply-templates
       select="rt:leaf(string(@rdf:about))" mode=
</xsl:for-each>
```

Warts

- Deep trees can be prohibitively large.
- “Serialize on the fly” is conceptually different.
- Trying to build trees that are “just big enough” sometimes introduces the inline/reference problem again.
- Function dispatch oddness in the current implementation.

References

- RDF Twig: http://rdftwig.sourceforge.net/
- Saxon: http://saxon.sourceforge.net/
- Xalan Java: http://xml.apache.org/xalan-j/
- Jena: http://www.hpl.hp.com/semweb/jena