by Philippe Poulard

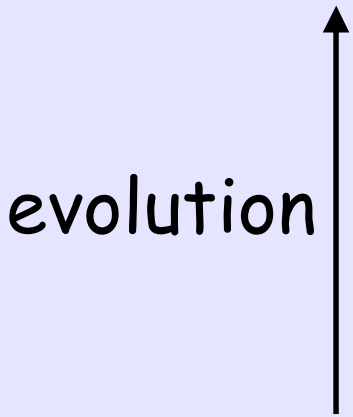
August 7-10, 2007 Montréal, Canada
Extreme Markup Languages®
a registered trademark of IDEAlliance

Active Tags

Mastering XML with XML

Philippe.Poulard@sophia.inria.fr

Darwin's point of view



Active Tags : everything is XML

OOP : everything is object

Unix : everything is a file

People still tend to map XML abstractions
toward the OO world

Processing-oriented XML languages

- **XSLT**
- **Ant**
- **XProc**
- **JSP/JSTL**/taglib
- **Jelly**
- many others…

each has been designed for a specific purpose

Scope and extensibility

- **XSLT** : extensions exists but…
  - can't handle blobs with SQL
  - a «Web» extension would be irrelevant
- **Ant** : for Java, strongly related to application packaging
- **XProc** : not so much control, only XML, not mature
- **JSP/JSTL**/taglib : for the Web only
- **Jelly** : can't design declarative libraries ?
- many others…

Standalone but alone

- What about putting together several instructions of each of them ?
  - if-then-else x N languages
  - design a new declarative language
- What about low-level considerations ?
  - variable handling (XPath)
  - unmarshalling

How new languages could take the benefits of other tag libraries ?

- A framework for native XML processing

Active Tags

http://ns.inria.fr/active-tags/

- Borrows the best to each of them
- Common rules for XOP
- Full power of integrated XML technologies
- and more…

What is the Active Tags suite ?

- A set of specifications (language/platform independant)
- A general-purpose framework
- Batch, Web applications, embedded in an application
- Looks like XSLT, but with several instructions set
- XPath-centric
- Can query various data sources
  (RDBMS, LDAP, XML native databases)
- Declarative / imperative / both

- Overall presentation of the system
  - Basic examples
- X-Operable objects
  - Browse non-XML objects with XPath
- Macro-tags
  - Binding an implementation to an active tag
- Deep mixity of grammar constructs
  - Mixing declarative languages with imperative sentences
  - The Active Schema Language
- The architecture explained
  - A self-defined engine
- Don't break your stream
  - XPath-based filtering for streaming pipelines

- Overall presentation of the system

Basic examples

- X-Operable objects

Browse non-XML objects with XPath

- Macro-tags

Binding an implementation to an active tag

- Deep mixity of grammar constructs

Mixing declarative languages with imperative sentences

The Active Schema Language

- The architecture explained

A self-defined engine

- Don't break your stream

XPath-based filtering for streaming pipelines

XCL : one of the core modules of Active Tags

Control structure instructions	<xcl:if><xcl:then><xcl:else> <xcl:for-each>
XML-oriented processing	<xcl:parse> <xcl:parse-stylesheet> <xcl:transform>
Document handling	<xcl:document> <xcl:element> <xcl:attribute> <xcl:append> <xcl:delete>
XPath-based filtering	<xcl:filter> <xcl:rule>

A simple « Active Sheet »

a convenient root

```
<xcl:active-sheet
    xmlns:xcl="http://ns.inria.org/active-tags/xcl">
  <xcl:parse name="myDoc"
      source="file:///path/to/document.xml"/>
  <xcl:parse-stylesheet name="myXslt"
      source="file:///path/to/stylesheet.xsl"/>
  <xcl:transform source="{$myDoc}"
      stylesheet="{$myXslt}"
      output="file:///path/to/output.html"/>
</xcl:active-sheet>
```

instruction
that creates
the property
named « myDoc »

XPath expression

Like « AVT » in XSLT
but :

- can occur in text content
- is not cast to string
- can refer to objects

Another simple « Active Sheet »

Get a system property

Create an XML document with some litterals

```
<xcl:active-sheet
    xmlns:sys=http://ns.inria.org/active-tags/sys
    xmlns:xcl="http://ns.inria.org/active-tags/xcl">
  <xcl:set name="who" value="{string($sys:env/who)}"/>
  <xcl:document name="xml">
    <example>
      <title>Hello {$who} !</title>
    </example>
  </xcl:document>
  <xcl:transform output="{$sys:out}" source="{$xml}"/>
</xcl:active-sheet>
```

Serialize the document to the standard output
As no stylesheet is involved, a copy is performed

An HTTP service

incoming URL :

http://www.eml2007.com/index.xml?who=John+Doe

match ?

```
<web:service
    xmlns:web="http://ns.inria.org/active-tags/web"
    xmlns:xcl="http://ns.inria.org/active-tags/xcl">
  <web:mapping match="^/index\.xml$">
    <xcl:set name="who"
             value="{string($web:request/who)}"/>
    <xcl:document name="xml">
      <example>
        <title>Hello {$who} !</title>
      </example>
    </xcl:document>
    <xcl:transform source="{$xml}"
        output="{value($web:response/@web:output)}"/>
  </web:mapping>
</web:service>
```

- Overall presentation of the system
  - Basic examples
- X-Operable objects
  - Browse non-XML objects with XPath
- Macro-tags
  - Binding an implementation to an active tag
- Deep mixity of grammar constructs
  - Mixing declarative languages with imperative sentences
  - The Active Schema Language
- The architecture explained
  - A self-defined engine
- Don't break your stream
  - XPath-based filtering for streaming pipelines

Cross-Operable objects


```
<xcl:transform source="{$xml}"
        output="{value($web:response/@web:output)}"/>
```

???

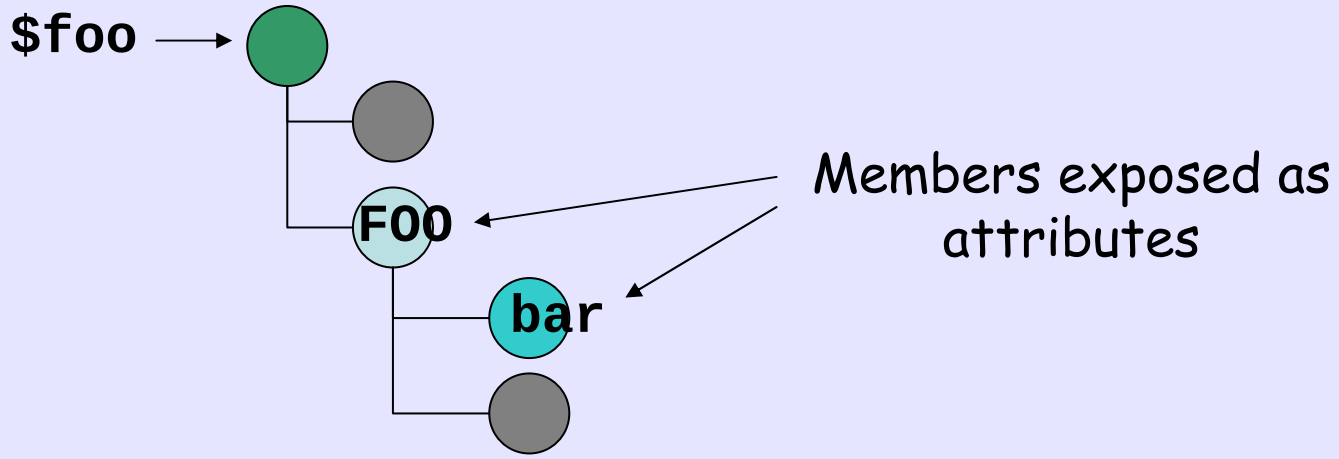
$web:response

- is not an XML node
- expose some of its properties as XML attributes
- that are not themselves XML nodes

An object that behaves like XML is a X-Operable object

- Browsable with XPath
- Operable with X-update

```
<xcl:attribute>
<xcl:append>
<xcl:delete>
```




```
$foo/@FOO/@bar
```

XPath syntax is correct

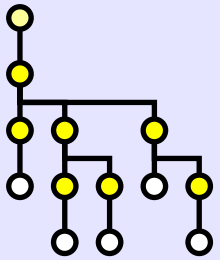
Not necessary representable with markup

→ this is an advantage

```
<web:response
   web:output="[io:output@189c036]"
   web:mime-type="text/html">
      <Cache-Control>no-cache</Cache-Control>
      <Date>Tue, 15 Nov 1994 08:12:31 GMT</Date>
</web:response>
```

Not necessary representable with markup

Avoid round-trip



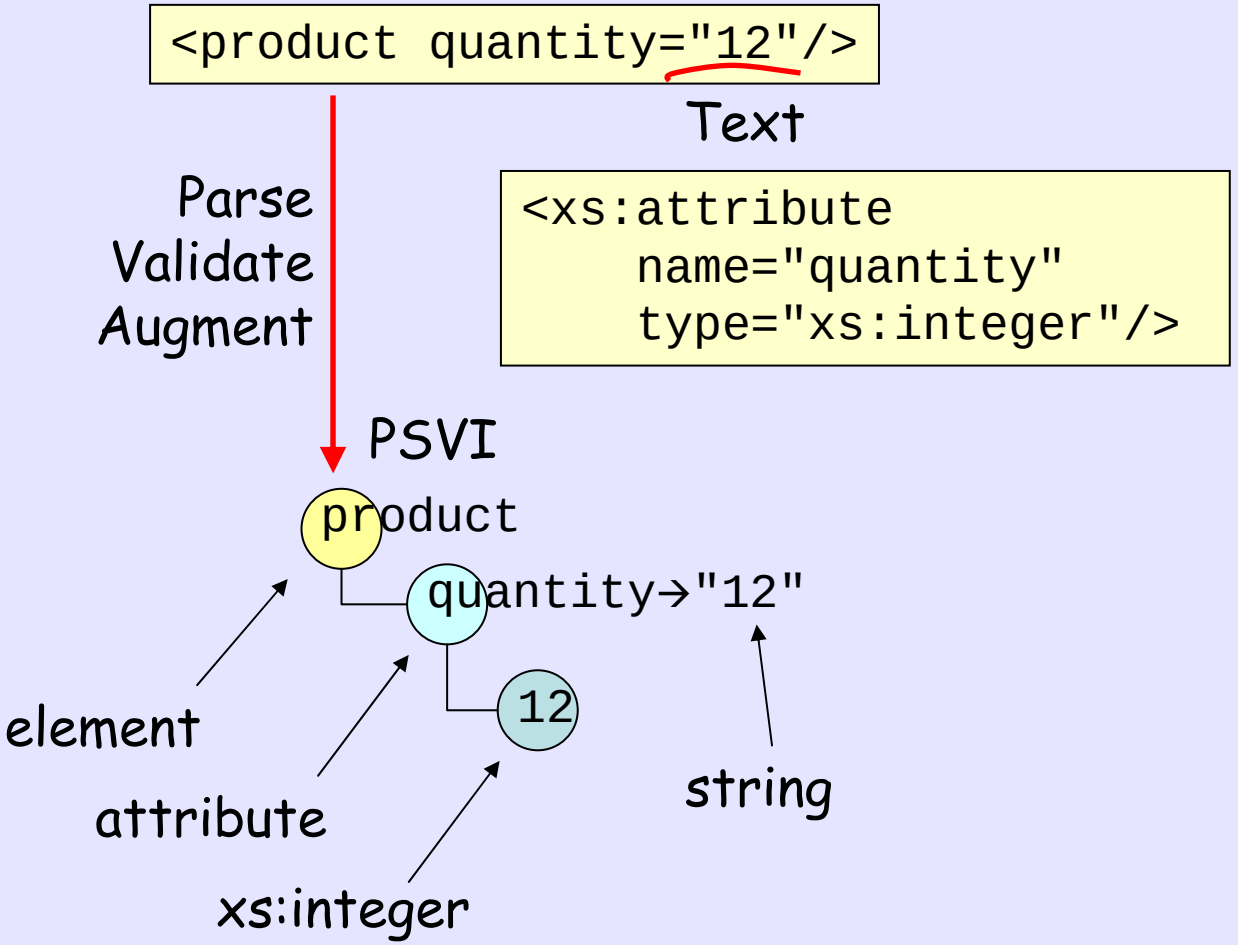
internal representation

An XML representation is sometimes irrelevant

Big objects, binary object : high-cost for XML encoding/decoding

Objects with lots of members : accessing them when necessary
(exemple : a file system)

Are they compatible ?

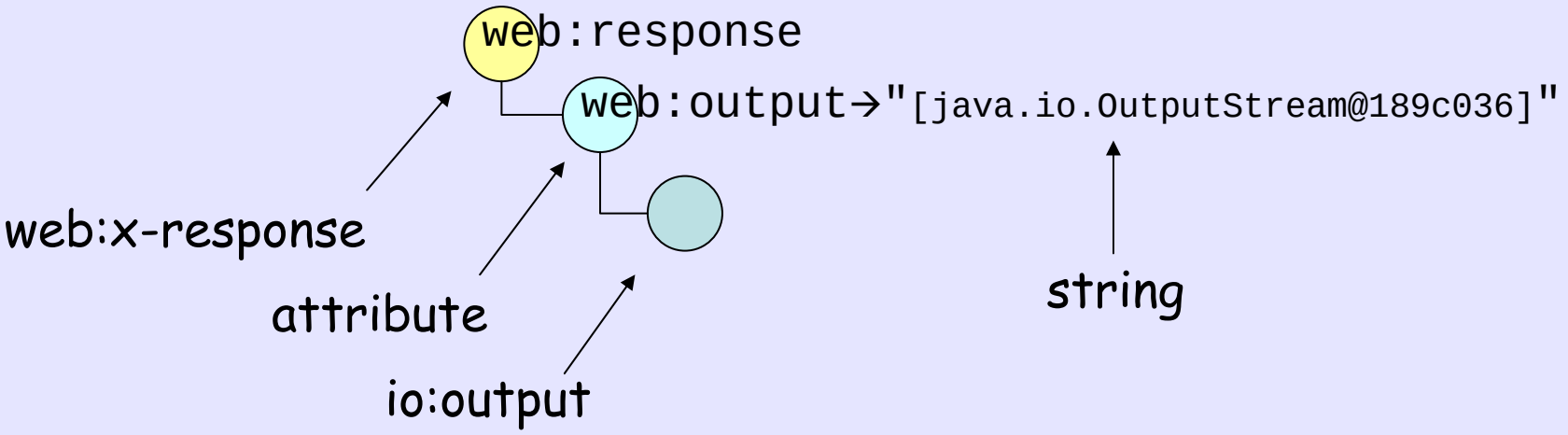


X-Operable objects and the XML data model


```
<web:response
   web:output="[io:output@189c036]"
   web:mime-type="text/html">
      <Cache-Control>no-cache</Cache-Control>
      <Date>Tue, 15 Nov 1994 08:12:31 GMT</Date>
</web:response>
```

~~Parse~~
~~Validate~~
~~Augment~~

This is an intrinsic characteristic of the typed data



X-Operable objects : exemple


```
<xcl:set name="baseDir"
  value="{io:file('file:///path/to/base/dir/')}"/>
```

Create an io:x-file

```
$baseDir/*
$baseDir//*
$baseDir//*[@io:extension='xml']
$baseDir//*[@io:size > 1024]
$baseDir/subpath/doc.xml
$baseDir/subpath/*[name()='123.xml']
```

No obscure syntax such as the linux `find` command

- it's XPath !
- works on all platform
- works on several URI schemes (file, http, ftp...)

- Overall presentation of the system
  Basic examples
- X-Operable objects
  Browse non-XML objects with XPath
- **Macro-tags**
  Binding an implementation to an active tag
- Deep mixity of grammar constructs
  Mixing declarative languages with imperative sentences
  The Active Schema Language
- The architecture explained
  A self-defined engine
- Don't break your stream
  XPath-based filtering for streaming pipelines

Materials that are "active"

active tags :	`<xcl:parse>`
XPath functions :	`io:file()`
predefined properties :	`$sys:out`
foreign attributes :	`@xcl:version` ← directives
data types :	`io:x-file` ← x- for « XML friendly »

EXP : Extensible XML Processor
One of the core modules of Active Tags

```
<exp:module
  xmlns:exp="http://ns.inria.org/active-tags/exp"
  xmlns:eml="http://www.extrememarkup.com/2007/active-tags/foo"
  version="1.0"
  target="eml"
>
  <exp:element name="eml:foo"
     source="res:com.extrememarkup.foo.FooAction"/>
  <exp:function name="eml:bar"
     source="res:com.extrememarkup.foo.BarFunction"/>
  <exp:attribute name="eml:version"
     source="res:org.inria.ns.reflex.processor.core.VersionAttr
</exp:module>
```

```
<eml:foo eml:version="1.0">
   {eml:bar()}
</eml:foo>
```

The implementation is not given by a class, but inline

In the IO module :

```
<exp:function name="io:exists">
  <xcl:set value="{value(io:file($exp:args[1])/@io:exists)}"/>
</exp:function>
```

```
{io:exists('file:///path/to/somewhere')}
```

An easy mean to define new active materials.

XSLT2 and XQuery can only define functions.

Where are the active tags ? In Active Tags !

Remember the first exemples...


```
<exp:module
    xmlns:exp="http://ns.inria.org/active-tags/exp"
    xmlns:xcl="http://ns.inria.org/active-tags/xcl"
    xmlns:eml="http://www.extrememarkup.com/2007/active-tags/he
    version="1.0"
    target="eml"
>
  <exp:element name="eml:say-hello">
    <xcl:document name="xml">
      <example>
        <title>Hello {value($exp:params/@who)} !</title>
      </example>
    </xcl:document>
    <exp:exports>
      <exp:export value="{$xml}"
        name="{string($exp:params/@variable-name)}"/>
    </exp:exports>
  </exp:element>
</exp:module>
```

This sequence was used twice

It can be replaced by our custom tag

```
<eml:say-hello
            who="[the name of the guy to say hello]"
 variable-name="[the name of the variable to export]"/>
```

- Overall presentation of the system
  Basic examples
- X-Operable objects
  Browse non-XML objects with XPath
- Macro-tags
  Binding an implementation to an active tag
- **Deep mixity of grammar constructs**
  Mixing declarative languages with imperative sentences
  The Active Schema Language
- The architecture explained
  A self-defined engine
- Don't break your stream
  XPath-based filtering for streaming pipelines

The power of declarative languages

- Specify "What" rather than "How-to"
- Processors operate them not necessary sequentially

→ Straigthforward, expressive, concise

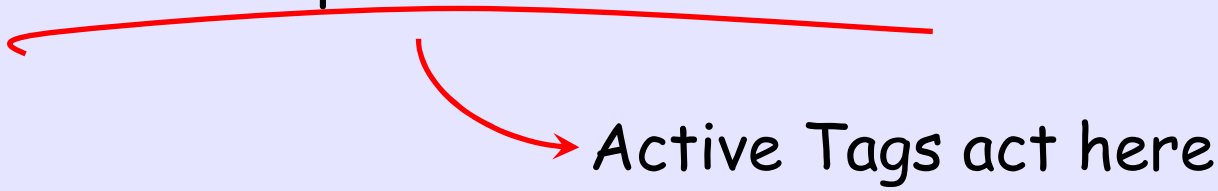
Exemples :

- W3C XML Schema
- XML Catalogs
- SCXML
- J2EE Web deployment descriptor
- ...

The limited power of declarative languages

- When the limit of the language is reached,
  there are no ways out

- Solutions :
  - Redesign (extension) ☹
  - Mix with imperative statements ☺



Exemple : the Active Schema Language

W3C XML Schema, Relax NG, DTD :

Instances of schemas are hard-coded

→ static abstract tree

Active Schema :

- similar constructs
  (elem. and attr. references, sequences, choices…)
- mixed with imperative constructs

→content models are computed while validating

→abstract trees become dynamic

→expressiveness is boosted

A purchase order


```
<purchase-order xmlns="http://www.extrememarkup.com
                          /2007/active-tags/po">
  <items total="188.93">
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
    <free-item partNum="261-ZZ">
      <productName>Kamasutra for dummies</productName>
      <quantity>1</quantity>
    </free-item>
  </items>
</purchase-order>
```

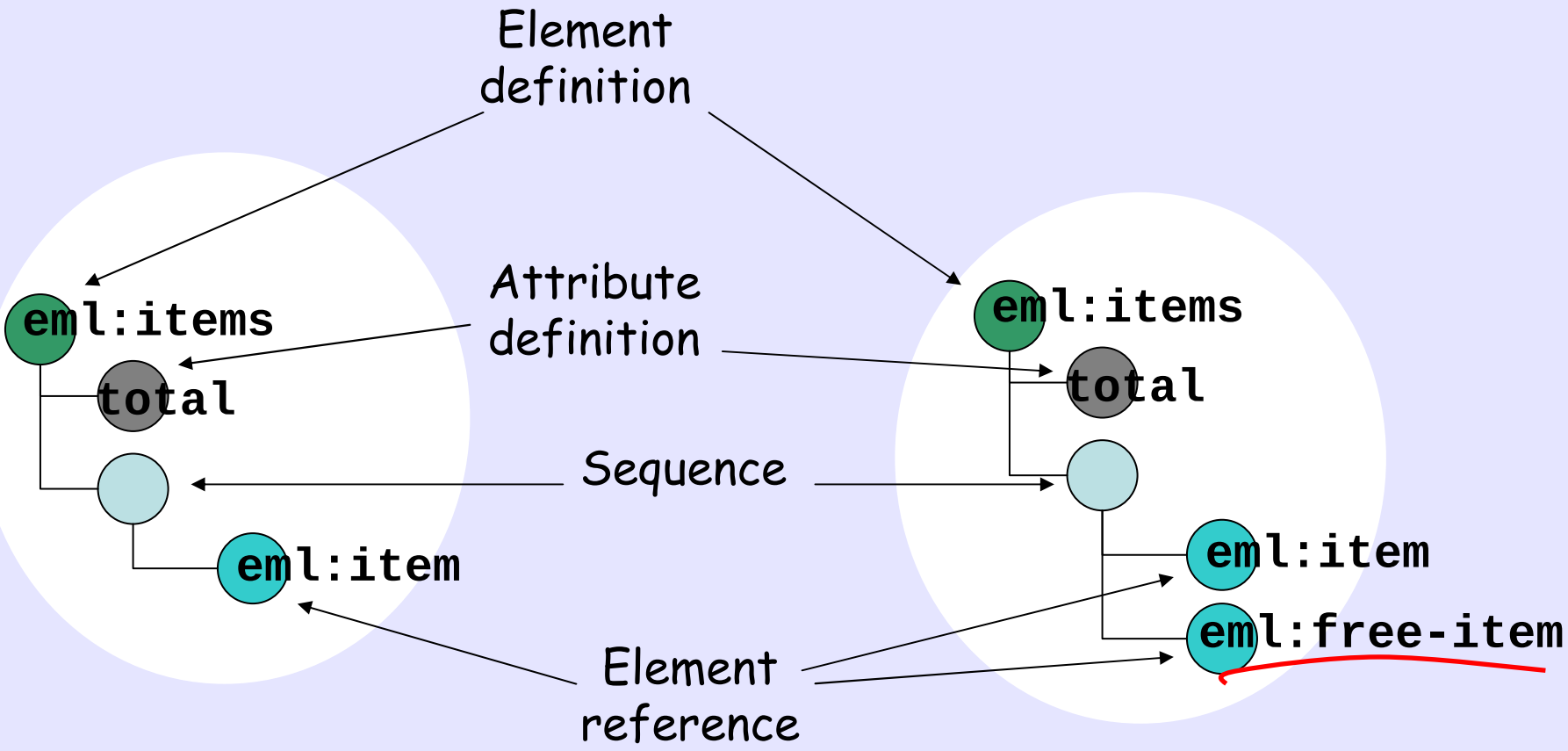
allowed only if the total amount exceeds 500$

```
<asl:element name="eml:items" root="never">
  <asl:attribute name="total" type="xs:decimal"/>
  <asl:sequence>
    <asl:element ref-elem="eml:item"
                 min-occurs="1"
                 max-occurs="unbounded"/>
    <xcl:if test="{asl:element()/@total &gt; 500}">
      <xcl:then>
        <asl:element ref-elem="eml:free-item"
                     min-occurs="1"
                     max-occurs="1"/>
      </xcl:then>
    </xcl:if>
  </asl:sequence>
</asl:element>
```

Possible "instanciations"

`@total < 500`

`@total ≥ 500`



Schematron can do the job ?

- Schematron doesn't act on the content models
- An editor could suggest an element to insert that Schematron would reject AFTER the insertion

Anyway, there are still things that W3C XML Schema, Relax NG, DTD, Schematron can't do

- Semantic data types :

  `http://reflex.gforge.inria.fr/tutorial-schemas.html#psvi`

- ASL features :

  `http://ns.inria.org/active-tags/active-schema/active-schema.html`

- Overall presentation of the system
  Basic examples
- X-Operable objects
  Browse non-XML objects with XPath
- Macro-tags
  Binding an implementation to an active tag
- Deep mixity of grammar constructs
  Mixing declarative languages with imperative sentences
  The Active Schema Language
- The architecture explained
  A self-defined engine
- Don't break your stream
  XPath-based filtering for streaming pipelines

Active Catalogs : putting them all together

the engine "knows" which tag is an action
and which tag is a literal
→ an Active Catalog is plugged to the engine

compatible

XML catalog : map URI → URIs

Active Catalog : map URI + selector → resources

(URI, module, schema, catalog, active-sheet...)

- Resource management facilities (caching policy)
- Lookup strategies
- Receipe for building the resource

```
<cat:catalog
    xmlns:cat="http://ns.inria.org/active-tags/cat"
    xmlns:exp="http://ns.inria.org/active-tags/exp"
    xmlns:asl="http://ns.inria.org/active-schema"
>
  <cat:resource name="http://www.extrememarkup.com
                      /2007/active-tags/po"
                uri="po-module.exp"
                selector="exp:module"/>
  <cat:resource name="http://www.extrememarkup.com
                      /2007/active-tags/po"
                uri="po-schema.asl"
                selector="asl:schema"/>
</cat:catalog>
```

Implicit selector for XML parser : `xml:external-identifier`

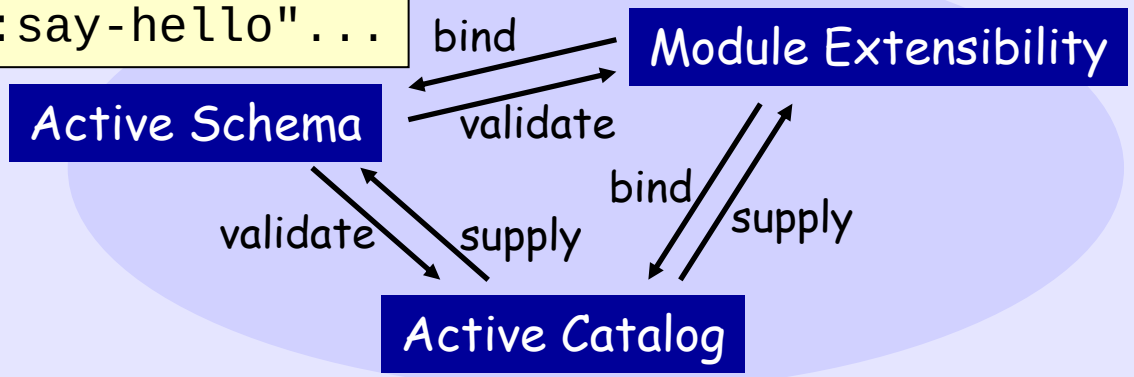
A self-descriptive system

Active tag ⟶ <eml:say-hello>

How is it resolved ?

```
<exp:module target="eml">
  <exp:element
    name="eml:say-hello"...
```

```
<asl:schema target="eml">
  <asl:element
     name="eml:say-hello"...
```




```
<cat:catalog>
  <cat:uri name="http://www.extrememarkup.com
                        /2007/active-tags/hello"
```

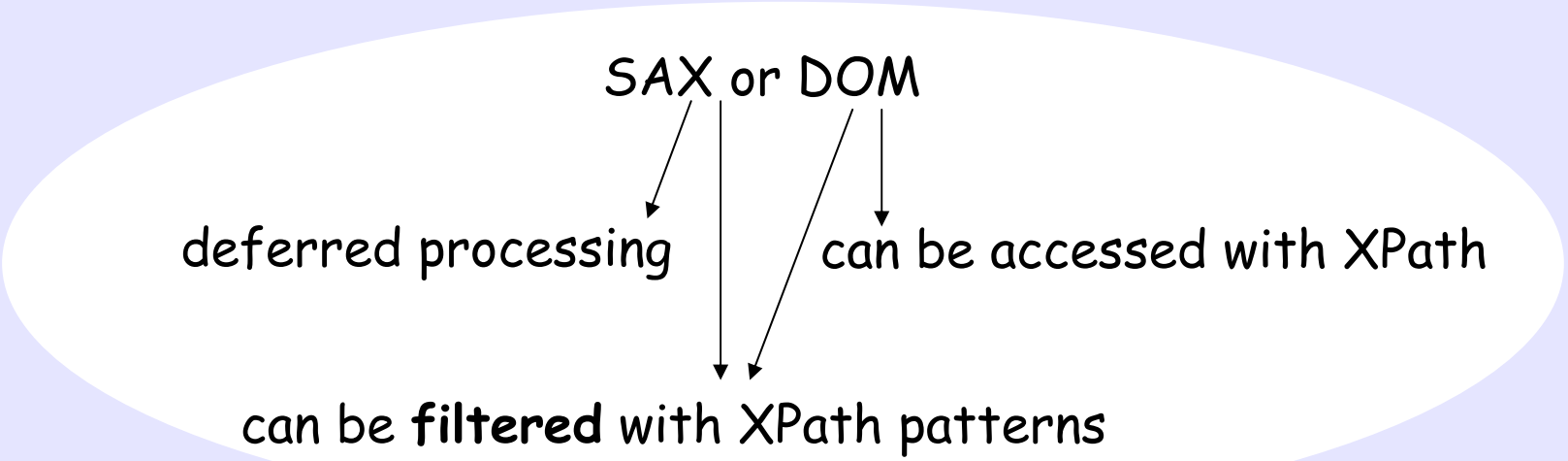
<cat:catalog>
<exp:module>
<asl:schema>
are resolved in
the same manner

- Overall presentation of the system
  Basic examples
- X-Operable objects
  Browse non-XML objects with XPath
- Macro-tags
  Binding an implementation to an active tag
- Deep mixity of grammar constructs
  Mixing declarative languages with imperative sentences
  The Active Schema Language
- The architecture explained
  A self-defined engine

- Don't break your stream
  XPath-based filtering for streaming pipelines

SAX
- Good for big document
- Good for streaming / pipelines
- Bad for its API

XCL filters :
- Mask SAX API
- XPath-based patterns
- No syntaxic limitations
- Also works for DOM trees



Strategy for saving the memory

1. Store a single branch

```
/
a
a/b/c
a//b
/a/b/c
/a//b
```

```
a[@b]
a[not(@b)]
a[@b='c']
a[@b='c']/d[@e]
```

Trivial cases

2. Store counters

```
/a/b/c[1]
a/*[2]
a/comment()[3]
a/node()[position() < 4]
```

The position depend on the node test involved

3. Read forward

```
/a/b/c[last()]
a/*[count() > 3]
a/node()[last()]
```

The size depend on what will be read

4. Read backward

Workaround : anticipate and "cast" a SAX subtree into a DOM subtree

Straightforward with XCL :

```
<xcl:parse name="myDom" style="tree"/>
<xcl:document name="mySax" type="stream">
      {$myDom}
</xcl:document>
```

```
<xcl:parse name="mySax" style="stream"/>
<xcl:document name="myDom" type="tree">
      {$mySax}
</xcl:document>
```

```
<xcl:filter
    xmlns:xcl="http://ns.inria.org/active-tags/xcl">

  <xcl:rule pattern="/">
    <xcl:forward>
      <wrapper>
        <xcl:apply-rules/>
      </wrapper>
    </xcl:forward>
  </xcl:rule>

  <xcl:rule pattern="bar[@delete]"/>

  <xcl:rule pattern="foo[1]/bar[last()]">
    <xcl:rename referent="{.}" operand="lastBar">
    <xcl:forward>
      <xcl:apply-rules/>
    </xcl:forward>
  </xcl:rule>

</xcl:filter>
```

Works like SAX :

- Browse all the input tree
- Specify what change

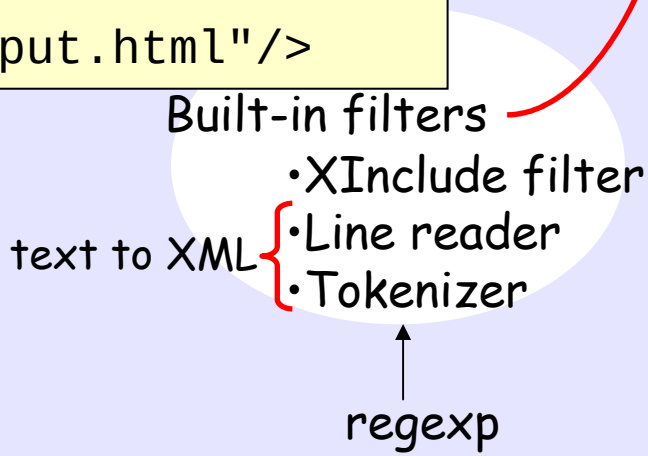
Channels in XCL filters


```
<xcl:rule pattern="/purchase-orders/order">
  <!--create a SAX document for each order-->
  <xcl:document name="order" style="stream">
    <xcl:forward channel="order">
      <xcl:apply-rules/>
    </xcl:forward>
  </xcl:document>
  <!--save each "order" in a file-->
  <xcl:transform source="{$order}"
   output="file:///path/to/purchase-orders/order-{@id}.xml"/>
</xcl:rule>
```

Active Tags

```
<xcl:parse-stylesheet
            name="myXslt"
          source="file:///path/to/stylesheet.xsl"/>
<xcl:parse-filter
            name="xinclude"
          source="http://www.w3.org/2001/XInclude"/>

<xcl:parse  name="myDoc" style="stream"
          source="file:///path/to/document.xml"/>
<xcl:filter name="included"
          source="{$myDoc}"
          filter="{$xinclude}"/>
<xcl:transform
          source="{$included}"
      stylesheet="{$myXslt}"
          output="file:///path/to/output.html"/>
```



Summary

- Allow to handle with ease XML datas and non-XML datas
  - SAX, DOM, XCL filters and pipelines
  - X-Operable objects
- A systemic consideration of XML technologies
  - each component focus on a well-defined problematic
  - component cooperation
- Help the design of XML languages
  - Custom modules (macro tags, macro XPath functions)
  - Facilitate expressiveness and extensibility
  - Declarative-oriented languages
- Active Schema Language
  - A proof of concept
  - Much more expressive than other schema technologies

The Active Tags system is a serious candidate for XML-oriented programming

RefleX

The Active Tags engine, in Java

- Have the RefleX ! `http://reflex.gforge.inria.fr`
- Free, open source
- Viability
  - Self-tested with XUnit
  - Lots of runnable examples and tips in RefleX
  - Already used in production at INRIA
  - Could be closer to XPath2/XQuery data model
  - Some features still experimental or incomplete

And now ?

To go further :

- read carefully the slides !
- read carefully the proceeding !

  `http://www.idealliance.org/papers/extreme/proceedings/html/2007/Poulard01/EML2007Poulard01.html`

- email-me

  `Philippe.Poulard@sophia.inria.fr`

- discuss about Active Tags on the XML-dev list
- download the engine

  `http://reflex.gforge.inria.fr`

- try the tutorials
- send me some money ☺

Questions ?