August 12-15, Balisage 2008 Montréal, Canada
The Markup Conference

Properties of schema mashups: dynamicity, semantic, mixins, hyperschemas

INRIA SOPHIA ANTIPOLIS
Philippe.Poulard@sophia.inria.fr

Missing features in XSD 1.0 :

- co-occurrence constraints

```
<xs:alternative>
```

works like an "if-then-else"
(or "switch-case" or "choose-when")

W3C XML Schema 1.1 is out (working draft) !

Bad new

```
<xs:element name="message" type="messageType">
  <xs:alternative test="@kind='string'" type="typeString"/>
  <xs:alternative test="@kind='base64'" type="typeBase64"/>
  <xs:alternative test="@kind='binary'" type="typeBase64"/>
  <xs:alternative test="@kind='xml'" type="typeXML"/>
</xs:element>
```

<xs:alternative>

…but specialized for type switching only

- Same basic semantic than if-then-else,
  with some arrangements for selecting a type
- Can't be used for other usage

Futur usage (missed in 1.1)

New usage, new semantic, new tag

```
<xs:alternative2>
```

Will be appliable on another context than type selection

…for XSD 1.2 ?

Futur usage (missed in 1.2)

New usage, new semantic, new tag

`<xs:alternative2>`

Will be appliable on another context than type selection

…for XSD 1.2 ?

`<xs:alter>`

…for XSD 1.3 ?

Things already exist…

Why not use this one ?

```
<xsl:if>
```

from XSLT

Why not use this one ?

```
<xsl:if>
```

from XSLT

Or this one ?

```
<scxml:if>
```

from SCXML

Use schema mashups !

Why trying to define everything in a single language ?

- there will be always something missing

  → need to upgrade the language

Define things once, use them when needed

- try to be general
- appliable to schemas

  → combine schemas languages with other languages

Schema mashups

Combine schemas languages
with non-schema languages

- example: **dynamicity** of content models
- example: **semantic** data types

Combine schemas languages
with other schema languages

- example: **mixins**

Side-effects

- example: **hyperschema**

an innovative and experimental schema language

Active Schema Language:

- similar constructs than other schema languages (elem. and attr. references, sequences, choices…)
- can be combined with imperative constructs

  → content models are computed while validating

  → abstract trees of schemas become dynamic

  → expressiveness is boosted

- built on top of a general-purpose XML engine :

  Active Tags

a container for runnable XML languages

(unlike ASL, it is not experimental)

- A set of specifications (language/platform independant)
- Looks like XSLT, but with several instructions set
- XPath-centric
- Can query various data sources (RDBMS with SQL, LDAP directories, XML native databases with XQuery)
- Declarative / imperative / both
- XML scripts are called "Active Sheets"

Modules in Active Tags

- Standard modules:
  - XCL: the XML Control Language
  - ASL: the Active Schema Language (exprimental)
  - EXP: define the Extensions of the XML Processor
  - Active Catalog
  - I/O module
  - System module
  - Web module
- Others:
  - XUnit
  - WUnit

a module defines :

- active tags
- XPath functions
- foreign attributes
- predefined properties
- data types

XCL : the XML Control Language: <xcl:if>, <xcl:then>, <xcl:else>, <xcl:parse>, <xcl:transform>, <xcl:update>…

```
<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"/>
  <!--XPath expressions appear in curly braces-->
  <xcl:transform source="{$myDoc}"
      stylesheet="{$myXslt}"
      output="file:///path/to/output.html"/>
</xcl:active-sheet>
```

Apply XPath on XML-aware objects


```
<!--set a working directory to an environment variable-->
<xcl:set name="dir"
    source="{io:file(string($sys:env/myDir))}"/>
<xcl:for-each name="file"
    select="{$dir/*[@io:extension='xml']}">
  <!--parse each XML file à la SAX-->
  <xcl:parse name="myDoc" source="{$file}" style="stream"/>
  <!--transform it in HTML-->
  <xcl:transform source="{$myDoc}"
    stylesheet="{$myXslt}"
    output="file:///path/to/published/{
                          $file/@io:short-name}.html"/>
</xcl:for-each>
```

How does the engine recognize <xcl:transform> as an active tag and not an XML litteral ?

→there is a module definition for XCL
that the engine "knows"

Binding an implementation to an active tag with EXP:

```
<!--bind a Java class to an active tag of the XCL module-->
<exp:element name="xcl:transform"
             source="res:org.inria.ns.reflex.
                                 processor.xcl.TransformAction"/>
<!--the "res" URI scheme refers to resources
       found in the classpath;
    this is specific to the implementation in Java-->
```

How does the engine knows the module definition of XCL ?

→it is specified in the main catalog of the engine

Like OASIS XML Catalogs

- but for resources (not only for URIs)

```
<!--declare 2 entries related to XCL in the main catalog-->
<cat:group xml:base="res:///org/inria/ns/reflex/processor/">
    <!--where to find the XCL module-->
    <cat:resource
        name="http://ns.inria.org/active-tags/xcl"
        uri="xcl/module.exp" selector="exp:module"/>
    <!--where to find the XCL schema-->
    <cat:resource
        name="http://ns.inria.org/active-tags/xcl"
        uri="xcl/schema.asl" selector="asl:schema"/>
</cat:group>
```

- and it says "miaow"

Example 1: dynamicity

Example 1 : a Purchase Order


```
<purchase-order
    xmlns="http://www.example.com/purchase-order"
    ship-date="2008-08-14" >
  <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>
```

business rule: allowed only if the total amount exceeds $500

Solution with traditional schemas

We must relax the constraint on the business rule

```
<!ELEMENT items (item+,free-item?)>
```

Other schema languages can't do much more better

Schematron ? Wait…

Imitation of the DTD with ASL

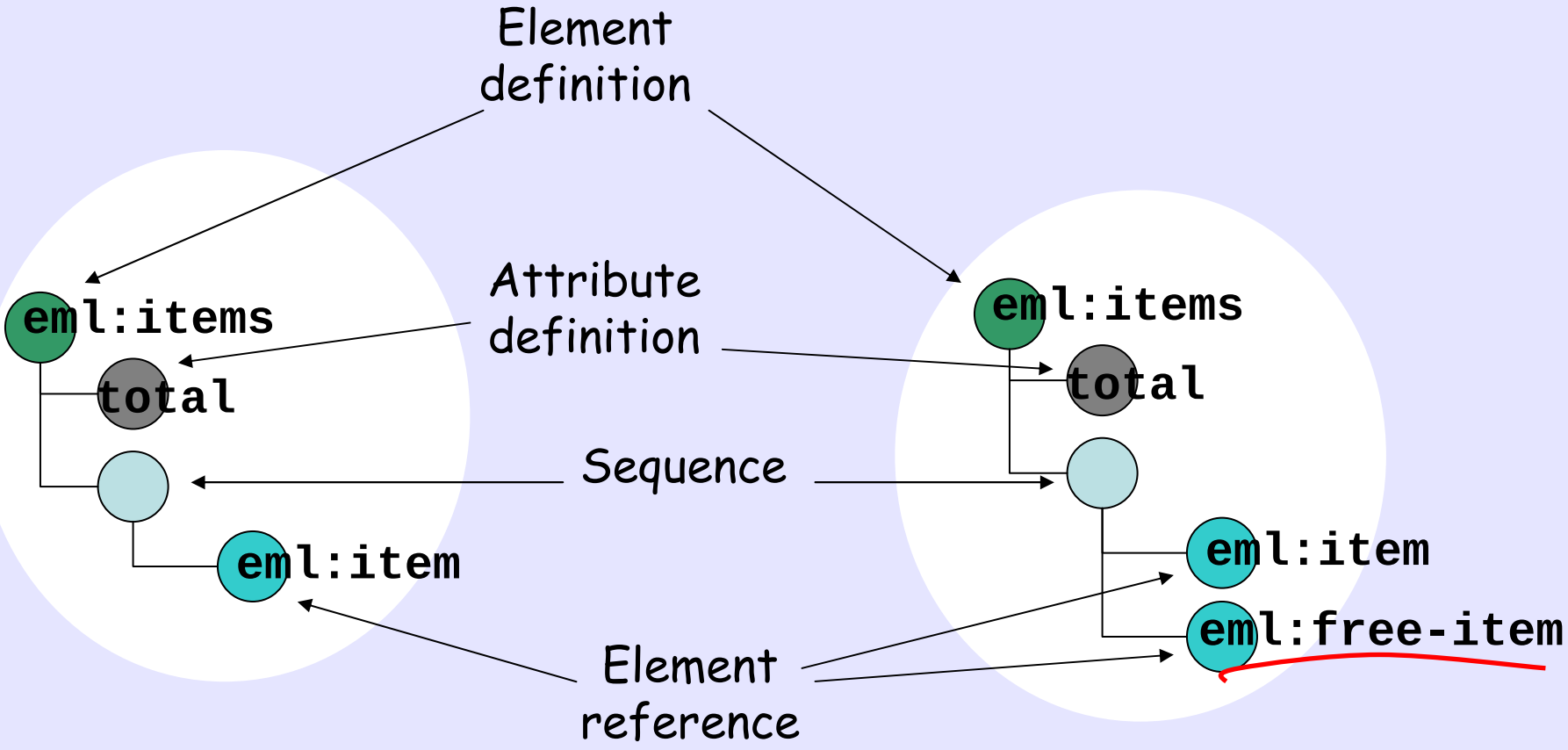

```
<asl:element name="eml:items" root="never">
   <asl:attribute name="total" type="xs:decimal"/>
   <asl:sequence>
      <asl:element ref-elem="eml:item"
                   min-occurs="1"
                   max-occurs="unbounded"/>



      <asl:element ref-elem="eml:free-item"
                   min-occurs="0"
                   max-occurs="1"/>


   </asl:sequence>
</asl:element>
```

Introducing dynamicity

```
<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}">
         <!--asl:element() refer to the current element-->
         <xcl:then>
            <asl:element ref-elem="eml:free-item"
                   min-occurs="0"
                   max-occurs="1"/>
         </xcl:then>
      </xcl:if>
   </asl:sequence>
</asl:element>
```

by injection of imperative instructions

Possible "instanciations"



Both trees can occur during the same validation
→excludes schema generation (XSLT)

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
- Other ASL features :

```
http://ns.inria.org/active-tags/active-schema/active-schema.html
http://reflex.gforge.inria.fr/tutorial-schemas.html
```

Example 2: semantic

Semantic data types ?

The semantic of a data type is related to its level of abstraction:

Model 0: byte view:

```
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E
30 22 3F 3E 3C 77 65 61 74 68 65 72 2D 72 65 70 6F 72 74 20 74 65 6D 70 3D
22 36 38 22 20 75 6E 69 74 3D 22 B0 46 22 3E
```

Semantic data types ?

Model 1: string view:

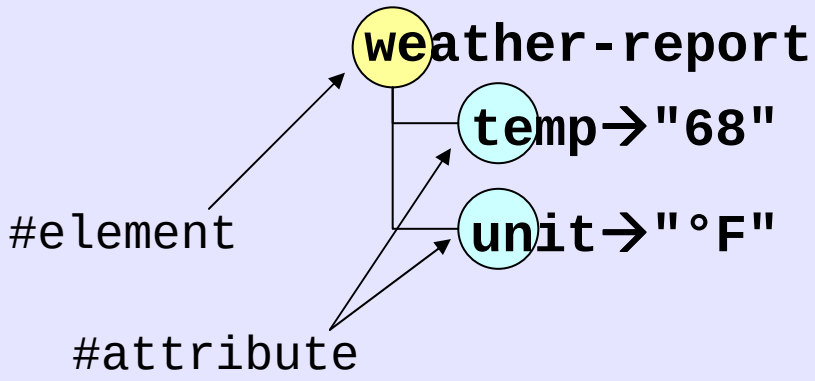
```
<?xml version="1.0"?><weather-report temp="68" unit="°F">
```

Model 0: byte view:

```
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E
30 22 3F 3E 3C 77 65 61 74 68 65 72 2D 72 65 70 6F 72 74 20 74 65 6D 70 3D
22 36 38 22 20 75 6E 69 74 3D 22 B0 46 22 3E
```

Semantic data types ?

Model 2: XML view:
(abstract)



Model 1: string view:

```
<?xml version="1.0"?><weather-report temp="68" unit="°F">
```

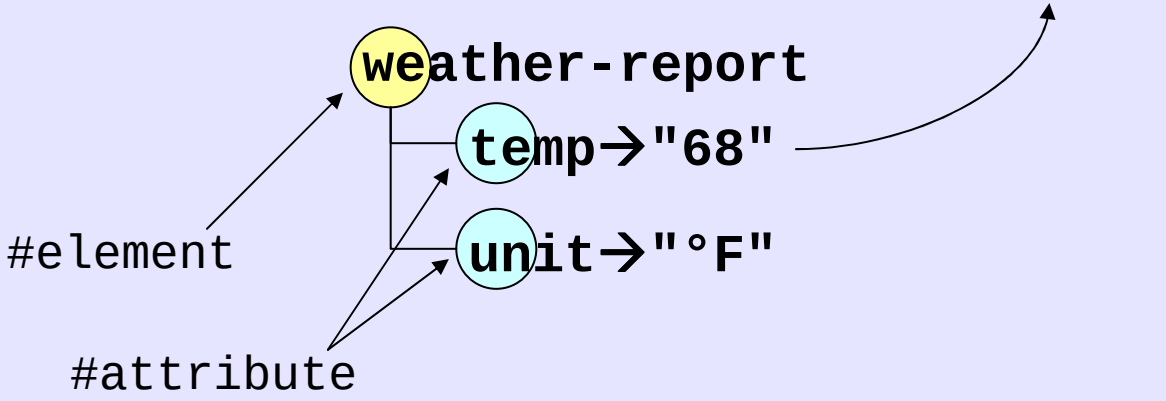
Model 0: byte view:

```
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E
30 22 3F 3E 3C 77 65 61 74 68 65 72 2D 72 65 70 6F 72 74 20 74 65 6D 70 3D
22 36 38 22 20 75 6E 69 74 3D 22 B0 46 22 3E
```

Semantic data types ?

Model 3: data type view: `#xs:decimal temp=68`

Model 2: XML view:
(abstract)



Model 1: string view:

```
<?xml version="1.0"?><weather-report temp="68" unit="°F">
```

Model 0: byte view:

```
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E
30 22 3F 3E 3C 77 65 61 74 68 65 72 2D 72 65 70 6F 72 74 20 74 65 6D 70 3D
22 36 38 22 20 75 6E 69 74 3D 22 B0 46 22 3E
```

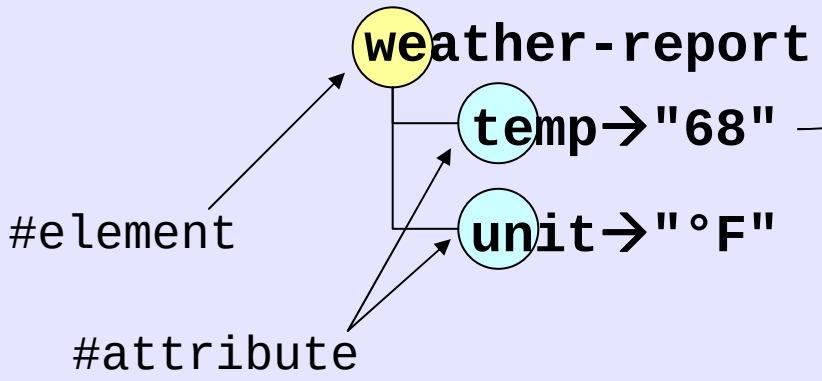
Model 4: semantic view: `68°Fahrenheit`

express relationship between data

Model 3: data type view: `#xs:decimal temp=68`

schema

Model 2: XML view: (abstract)



Model 1: string view:

```
<?xml version="1.0"?><weather-report temp="68" unit="°F">
```

Model 0: byte view:

```
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E
30 22 3F 3E 3C 77 65 61 74 68 65 72 2D 72 65 70 6F 72 74 20 74 65 6D 70 3D
22 36 38 22 20 75 6E 69 74 3D 22 B0 46 22 3E
```

Example 2: a Weather Report


```
<weather-report>
    <city name="Paris"  temp="19" unit="°C"/>
    <city name="Rome"   temp="22" unit="°C"/>
    <city name="Berlin" temp="32" unit="°F"/>
                         <!-- 32°F = 0°C -->
    <city name="Madrid" temp="23" unit="°C"/>
    <city name="London" temp="68" unit="°F"/>
                         <!-- 68°F = 20°C -->
</weather-report>
```

a very simple application: **sorting cities by temperature**

- fail with XML technologies
  (stop at Model 3)
- simple to achieve with other languages such as Java

Why is it so important ?

After all, OO languages (Java) can take the relay to address this issue

→expect a mapping OO-XML (unmarshal)

Tools based on the XML Data Model are left out:

- XSLT
- XQuery

If OO languages are able to support them, native XML languages should support them too

- ✘ XSLT
- ✘ XQuery
- ✓ Active Tags

Refering to a data type with ASL


```
<!--a <city> contains only attributes-->
<asl:element name="city">
    <asl:attribute name="name" ref-type="xs:string"/>
    <!--the @temp attribute refers to our custom type-->
    <asl:attribute name="temp" ref-type="temperature"/>
    <asl:attribute name="unit">
        <asl:text value="°C"/>
        <asl:text value="°F"/>
    </asl:attribute>
</asl:element>
```

Designing a custom data type with ASL


```
<!--#temperature is our custom type
    it will build a typed data based on a #xs:decimal-->
<asl:type name="temperature" base="xs:decimal" init="{.}">
   <!--asl:element() refers to the current element,
       actually a <city>-->
   <xcl:if test="{ asl:element()/@unit='°F' }">
      <xcl:then>
         <!--if @unit="°F", the value of the typed data
                                              is updated
             $asl:data is the structure bound to the
             attribute that handles the current typed data
             "." is the current data, an #xs:decimal-->
         <xcl:update referent="{ $asl:data }"
              operand="{ (value(.) - 32) * 5 div 9 }"/>
      </xcl:then>
   </xcl:if>
</asl:type>
```

Run script


```
<xcl:parse          name="wr"
                    source="weather-report.xml"/>
<asl:parse-schema name="wr-schema"
                    source="weather-report.asl"/>
<!--the "augment" attribute indicates to bind typed datas
to XML items in the XDM-->
<asl:validate schema="{$wr-schema}"
                node="{$wr}"
                augment="yes"
                deep="yes"/>
<xcl:echo value="List of cities, sorted in
                                        temperature order:"/>
<xcl:for-each name="city"
                select="{xcl:sort( $wr/*/city, @temp )}">
    <xcl:echo value="{$city/@temp}{$city/@unit}
                                        {$city/@name}"/>
</xcl:for-each>
```

```
List of cities, sorted in temperature order:
32°F Berlin
19°C Paris
68°F London
22°C Rome
23°C Madrid
```

The string value of the attribute remains the same, whereas the bound typed data was involved in the sort operation

Our type can validate:

```
<city name="Rome"   temp="22" unit="°C"/>
```

We can also define a variant that can validate:

```
<city name="Paris"  temp="19°C"/>
```

and another one that can validate:

```
<city name="Paris"  temp="19°C"/>
<city name="Rome"   temp="22" unit="°C"/>
```

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

Example 3:
mixins

Ability to combine several schemas.

Currently partially supported:

- Relax NG and ASL can use W3C XML Schema datatypes
- Import mechanism in XSD
  (must be XSD too)
- Schematron embedded in XSD
  (both are ignorant of the other)
- MSV: common representation
- NVDL: cooperation of several schema technologies
  (each act on a single namespace URI)
- ISO/DSDL part 9 can use XSD datatypes in DTD
  (intrusive)

We can't design in the same namespace two parts of a schema with two different schema technologies

- think about the DTD community:
  - DTDs are enough
  - hundreds of existing DTDs

→Use ASL to "patch" the DTD

Example 3: a compatible DTD for the weather report


```
<!--FILE: weather-report-legacy.dtd-->
<!ELEMENT weather-report (city)+>
<!ELEMENT city EMPTY>
<!ATTLIST city name CDATA #REQUIRED
               temp CDATA #REQUIRED
               unit CDATA #REQUIRED>
```

Relaxed constraints:

- the temperature is not numeric
- the unit can't be enumerated (invalid XML tokens)
- the relationship between °C and °F can't be expressed

ASL data types


```
<!--FILE: weather-report-datatypes.asl-->
<asl:active-schema
    xmlns:xcl="http://ns.inria.org/active-tags/xcl"
    xmlns:asl="http://ns.inria.org/active-schema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-datatypes"
    target="">

  <!--#temp-units is the type for temperature units-->
  <asl:type name="temp-units">
     <asl:choice>
        <asl:text value="°C"/>
        <asl:text value="°F"/>
     </asl:choice>
  </asl:type>

  <asl:type name="temperature" base="xs:decimal" init="{.}">
      <!-- [this the type we defined earlier] -->
  </asl:type>

</asl:schema>
```

the ASL master schema


```
<!--FILE: weather-report-master.asl-->
<asl:active-schema
    xmlns:xcl="http://ns.inria.org/active-tags/xcl"
    xmlns:asl="http://ns.inria.org/active-schema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-datatypes"
    target="">

   <!--redefine only what needed-->
   <asl:element name="city">
      <asl:attribute name="temp" ref-type="temperature"/>
      <asl:attribute name="unit" ref-type="temp-units"/>
      <!--other definitions are preserved-->
      <asl:apply-definition/>
   </asl:element>

</asl:active-schema>
```

Other considerations:

- prepend
- append

} content model fragments to the content model

```
<cat:catalog
      xmlns:cat="http://ns.inria.org/active-catalog"
      xmlns:asl="http://ns.inria.org/active-schema">
   <!--if our XML structure had a namespace URI,
      the name attribute below would contain it litteraly-->
   <cat:resource name=""  selector="asl:schema"
                 uri="weather-report-master.asl"/>
   <cat:resource name="" selector="asl:schema"
                 uri="weather-report-datatypes.asl"/>
   <!--asl:schema is the selector for all kind of schemas:
               DTD, ASL, W3C XML Schema, Relax NG, others -->
   <cat:resource name="" selector="asl:schema"
                 uri="weather-report-legacy.dtd" />
</cat:catalog>
```

Example 4: hyperschemas

Side effects of dynamicity

Our business constraint can also be expressed with a pure ASL schema !

```
<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"/>
    <asl:element ref-elem="eml:free-item"
       min-occurs="0"
       max-occurs="{number(asl:element()/@total &gt; 500)}"/>
  </asl:sequence>
</asl:element>
```

Potentially, dynamicity can appear:

- within some elements
- within some attribute values

→How to validate ASL (or XCL and others) ?

Example 4: validation of an active tag


```
<asl:element
   ref-elem="eml:free-item"
   min-occurs="0"
   max-occurs="1"/>
```

#xs:int

```
<asl:element
   ref-elem="eml:free-item"
   min-occurs="0"
   max-occurs="{number(asl:element()/@total &gt; 500)}"/>
```

"dynamic" #xs:int ?

#xs:string (relaxing constraint) ?

Dynamicity can occur in attribute values and within elements:

- Not so useful to say that all attributes are #xs:string and all elements contain #xs:any

a New concept: Hyperschema

Need a validation at a higher level: **hyperschema**
(act at the component level)
Schema that act at both level: **multidimensional schema**
(act at the markup and component levels)

Multidimentional schema for ASL:

```
<asl:attribute name="min-occurs"
               ref-type="xs:int"
               dynamicity="[enabled|disabled|mandatory]"/>
```

Won't necessarily validate active tags,
but the underlying software component at runtime

Dynamicity expect sometimes a defered validation

Specific to Active Tags, proposal not implemented

About schemas:

Schemas mashups provide valuable features

- **Dynamicity**: building self-adaptative content models
- **Semantic**: enhancing meaning of data types
- **Mixins**: collecting schema flavours
- **Hyperschemas**: validating high-level components

More generally:

- Don't use XML-processing languages alone
- Don't use declarative languages alone
- Don't use schema languages alone

→Several languages can help each other

RefleX

The Active Tags engine, in Java

- 220,000 lines of code (94,000 lines of comments and 16,000 blank lines)
- Jar size: 1.3MB

- 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
  - Some features still experimental or incomplete

Questions ?