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

Philippe Poulard

Balisage: The Markup Conference
Proceedings 2008

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

Philippe Poulard
INRIA
<philippe.poulard@sophia.inria.fr>

Abstract

W3C XML Schemas specifications were published in 2001, but a large community still uses DTDs. Perhaps they didn't adopt new schema technologies because they are still awaiting some missing features ? What kind of feature would be useful to DTD users and schema practitioners ?

Let's transpose the buzzword "mashups" to schema technologies: mashups of several schema languages and mashups of schema languages with non-schema languages. From this bazaar, we'll discuss some of the emerging features that could make what would be the next generation of schema languages: dynamicity, semantic, mixins, and hyperschemas.

Table of Contents

Introduction

The W3C XML Schema 1.1 specification is in a "last call working draft" status. One of the most significant change is perhaps the support of co-occurrence constraints that was missing in 1.0 and expected by the community. For the other missing stuffs -who knows what they are- we have to wait for W3C XML Schema 1.2 and so on. In this paper, we'll have a prospective approach: we examine some features that the author has already implemented as a demonstration, and imagine others that lead us to consider new usages of schema technologies.

For this purpose, we introduce an experimental but innovative schema technology: the Active Schema Language, or ASL. The strength of ASL comes from its runtime environment: the Active Tags engine that was presented last year at Extreme Markup Languages in Montreal (Poulard). Active Tags can host several runnable tag libraries and a schema language like ASL is just another library like other tag libraries, but built on top of this framework that offers to ASL lots of valuable services that make it much more simple, powerful and expressive than the serie of W3C XML Schema 1.x as well as other schema technologies.

Firstly, we will expose the foundations of the system. In the next chapters, we will explore 4 simple use cases:

- Since an Active Tags engine can make cohabit several tag libraries, even declarative languages like schema languages, we'll see hereafter how to design a schema that can build dynamically its content models (named active content model) by interweaving imperative instructions with declarative ones, in order to solve issues that other schema technologies can't address.
- Next, we'll focus on XML data types and the lack of support of semantic data types in schema languages. A simple example will show how ASL can define simply such a data type, and the advantage got for applications that have to deal with XML datas.
- We'll go on with the concept of mixins (borrowed to other kind of languages) adapted to schema languages. Here, we'll mix several schema technologies for extending an attribute definition of a DTD with a data type (other than CDATA, of course).
- Finally, we'll discuss about the validation of software components represented with declarative XML languages. Although schemas languages can validate the static representation of an assembly of components, they are not designed to validate pieces of components assembled at runtime. An hyperschema would act at this higher level.

We will conclude that designing XML languages in a framework such as Active Tags is extremely valuable. Even (or especially) for declarative languages such as ASL.

Foundations of the system

In this section, we present an overview of the system and its basic functioning.

Basically, an XSLT processor or a schema validator are doing the same things: first, since they are languages that rely on XML, instances are parsed, then unmarshalled to instances of some classes and assembled, and finally processed according to the intended semantic of the tags encountered. This is where the differences occurs. Processing those steps in a pipeline mode or generating some code in a target language are only implementation details. We can simply consider XSLT, W3C XML Schema, Relax NG, SCXML, OASIS XML Catalogs and other markup languages designed for processing purpose as tag libraries, but none of them rely on a common system. Yet, some systems are based on tag libraries such as JSP/JSTL and Jelly but the former is runnable exclusively within a Web server and both are very far from XML problematics: for example, instead of using XPath they rely on the UEL and are tightly coupled to Java.

Active Tags is a system that considers that XML languages designed for processing purpose like those mentionned are tag libraries runnable in an XML-based environment. Active Tags offers to runnable XML languages a set of basic services such as handling XML datas and non-XML datas with XPath, using templates, defining macros, and mapping tags to their implementations. This allows a programmer to focus on its implementation without worrying about the plumbing details.

In the same way that XSLT scripts are called "*stylesheets*", Active Tags ones are called "*active sheets*". Like XSLT, it is XPath centric, and *active sheets* will contain both instructions (the so-called *active tags*) and XML litterals. The main difference is that instead of having a single instruction set, an *active sheet* may contain severals, each bound to a namespace URI. One of the core module of the system is the XML Control Language, or XCL, that supplies a set of tags that covers many common features:

- Usual control structure actions, such as alternative (`<xcl:if>` `<xcl:then>` `<xcl:else>`) or iterative actions (`<xcl:for-each>`).
- XML oriented actions, such as XML parsing (`<xcl:parse>`) and XSLT transforming (`<xcl:parse-stylesheet>` and `<xcl:transform>`); these actions deal with entity and URI resolving, passing parameters (`<xcl:param>`), error handling and many other options used to tune XML processes.
- XML document creation (`<xcl:document>`, `<xcl:element>`, `<xcl:attribute>` etc) and high level Active Update implementation, that allow to perform update operations on XML objects and X-operable objects[1] (`<xcl:delete>`, `<xcl:append>`, `<xcl:update>` etc).

The reader is sent back to RefleX, a Java implementation of Active Tags, where numbers examples are available and runnable. RefleX supplies a command line interface and a servlet, and offer means to query relational databases with SQL, XML databases with XQuery, or LDAP repositories. Moreover, he will find in RefleX all the standard modules that go along with Active Tags (I/O, System, Web, etc) and specific modules for designing XML-based test suites (XUnit) and test suites for Web applications (WUnit).

In this first example, XCL is used in a sequence of three operations for parsing a file and transforming it with XSLT:

```
<xcl:active-sheet xmlns:xcl="http://ns.inria.org/active-tags/xcl">
    <xcl:parse name="input" source="file:///path/to/document.xml"/>
    <xcl:parse-stylesheet name="xslt" source="file:///path/to/stylesheet.xsl"/>
    <!--XPath expressions appear in curly braces-->
    <xcl:transform output="file:///path/to/result.html" source="{ $input }"
    stylesheet="{ $xslt }"/>
</xcl:active-sheet>
```

Active sheets can use a single module (like above) or several ones, can be procedural (like above once again) or declarative or both like we'll see in the next section.

Other core modules include means to bind an implementation to an active tag (i.e. which is not an XML litteral):

```
    <!--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 RefleX implementation in Java-->
```

...and means to lookup for resources such as the actual modules:

```
    <!--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>
```

When an *active sheet* is launched, the engine will look in the main catalog (and custom catalogs if any) if there is a module for each XML tag it encounters, and load the implementation provided by the relevant module, otherwise the actual tag stands for a litteral.

The so-called Active Catalog used in Active Tags like shown above is a compatible extension to OASIS XML Catalog: the former is designed to retrieve resources whereas the latter just to map an URI to another URI. When the engine launches a "module request", the reference specified by the entry in the catalog, tagged with the appropriate role (`selector="exp:module"`), will be unmarshalled to the relevant component (the module expected).

Other core modules and custom modules can be declared and registered to the engine as well. So far, nothing about schemas were mentionned; in the same way, there is a module definition for ASL that bind ASL elements to their implementation, and an entry in the main catalog so that ASL is ready to use by the engine. Therefore, we are able to design schema instances with ASL, and write *active sheets* for validating XML instances with our schema. We'll see hereafter how to combine XCL and ASL to express dynamic content models in a schema.

Dynamicity, or building active content models

Schema processors are building an abstract tree from a schema instance. With a traditional grammar-based schema (DTD, W3C XML Schema, Relax NG), as the schema instance is hard-coded, the abstract tree is static, making the expressiveness of the schema limited to what is allowed by the grammar. The flaw with grammars in XML is that they only allow to constraint content models in a declarative manner, which is in essence very concise and expressive, but when the limits of the declarative syntax are reached, there is no way out; it is still possible to add a new tag to express the missing declarative tag, but the limit still exists a single step further, at the cost of upgrading the language.

ASL has been designed in order to be much more expressive without adding tags again and again. The immediate benefit is to avoid to compromise a user's XML structure just because some constraints can't be expressed by grammar-based schemata, which happens often with traditional schema languages. ASL contains similar constructs than others schema languages: an element declaration is still made of sequences (`<asl:sequence>`) or choices (`<asl:choice>`) of element references (`<asl:element ref-name="...">`), texts (`<asl:text value="...">`) or attributes (`<asl:attribute>`), but they can be mixed with imperative constructs. As the content models are computed at runtime while validating, the result abstract tree becomes dynamic, increasing dramatically the expressiveness of the schema: the content models can adapt themselves to the incoming data to validate in an extreme flexible way. Additionally, ASL allows to compute dynamically thanks to XPath expressions occurrence constraints, that are at best hard-coded in existing schema languages.

The following document is an instance of 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>
    </item>

    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
    </item>

    <free-item partNum="261-ZZ">
      <productName>Kamasutra for dummies</productName>
      <quantity>1</quantity>
    </free-item>

  </items>

</purchase-order>
```

It is constrained by structual rules and business rules, in the circumstances a `<free-item>` element is allowed only if the total amount exceeds 500$ (which makes the above document invalid). Due to the lack of expressiveness of existing schema languages, the best we can do is to relax some constraints and ignore the business rule. The content model of the `<items>` element would be expressed like this in a DTD:

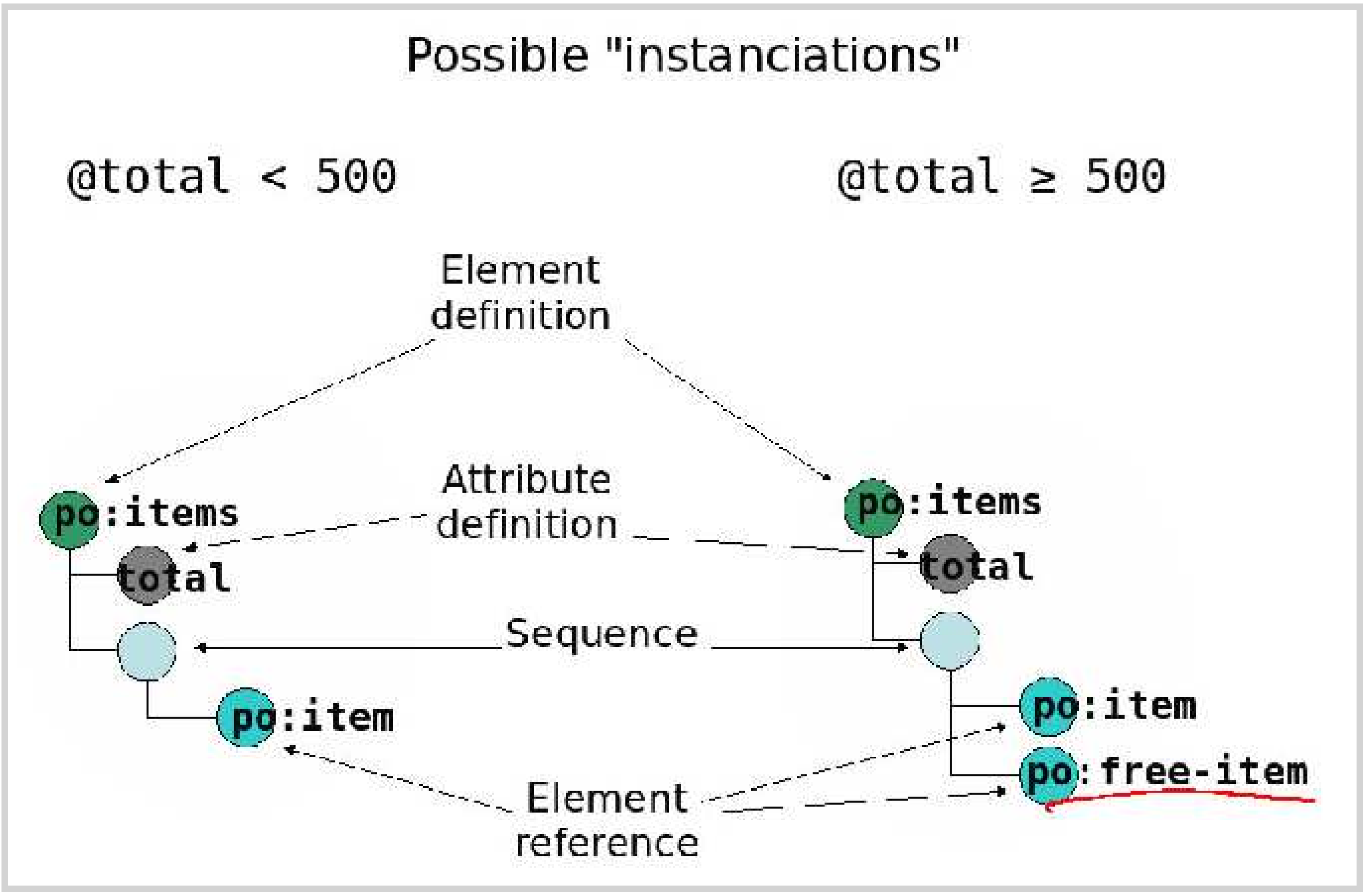
```
<!ELEMENT items (item+,free-item?)>
```

Unfortunately, an instance like the previous one violates our business rule although it is valid regarding the DTD. Other schema languages can't do much more better, except Schematron that we will talk about hereinafter.

ASL allows to write the business rule exactly as it has been expressed, by injecting imperative instructions from the XML Control Language within the content model definition:

```
<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"
    xmlns:po="http://www.example.com/purchase-order"
    target="po"
>

  <!--the root element of a purchase order-->
  <asl:element name="po:purchase-order" root="always">
    <asl:attribute name="ship-date" type="xs:date"/>
    <asl:sequence>
      <asl:element ref-elem="po:items"/>
    </asl:sequence>
  </asl:element>

  <!--a dynamic content model-->
  <asl:element name="po:items" root="never">
    <asl:attribute name="total" type="xs:decimal"/>
    <!--a variable sequence-->
    <asl:sequence>
      <asl:element
          ref-elem="po:item"
          min-occurs="1"
          max-occurs="unbounded"/>
      <!--the test that introduces variability
          asl:element() refers to the current element, actually a <po:items>-->
      <xcl:if test="{ asl:element()/@total &gt; 500 }">
        <xcl:then>
          <asl:element
              ref-elem="po:free-item"
              min-occurs="0"
              max-occurs="1"/>
        </xcl:then>
      </xcl:if>
    </asl:sequence>
  </asl:element>

  <asl:element name="po:item" root="never">
    <!--content model here-->
  </asl:element>

  <asl:element name="po:free-item" root="never">
    <!--content model here-->
  </asl:element>

  <!--other element definitions here-->

</asl:active-schema>
```

This schema demonstrates that an imperative operation is used to build the content model during the validation. The content model of the `<items>` element will vary according to the total amount found in the incoming document. Each of the "realizations" or "instanciations" of the element definition and illustrated by the picture below leads to a different abstract tree of the grammar. But all are expressed in a single self-adaptative schema. A program that would generate a schema wouldn't have the same expressiveness: a content model designed on the fly won't necessary produce the same result each time it is traversed. In fact, the system works like if it was a program that locally build fragments of content models while it is traversed during the validation. The injection of imperative instructions doesn't change the nature of ASL which is declarative.



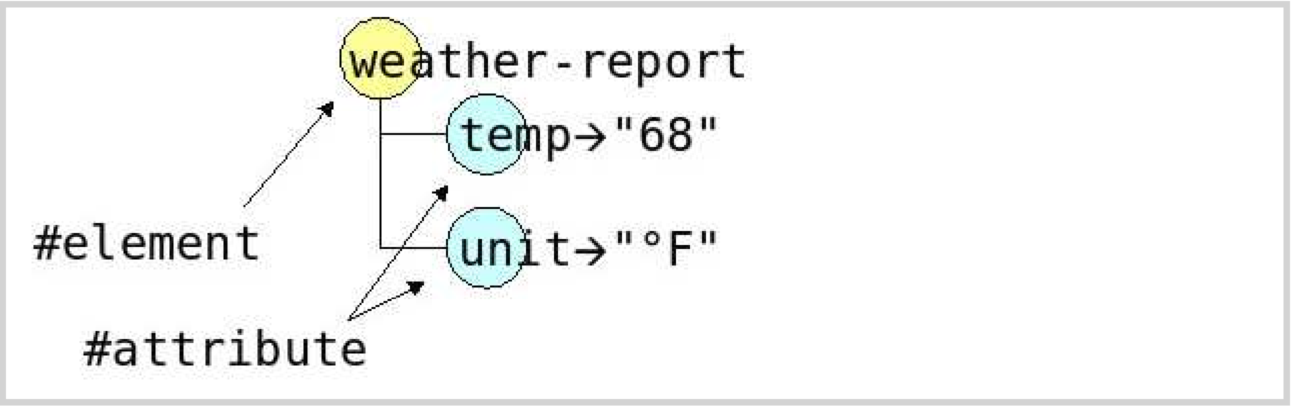
Schematron (as well as assertions in W3C XML Schema 1.1) is a technology that offers similar services; however, there is a fundamental difference: Schematron act outside content models whereas ASL defines them. Schematron will report constraints violations after grammar-based validation. A tool such as an editor will propose to the user to insert a `<free-item>` whereas Schematron will reject it after the insertion ! ASL will introduce it in the content model only when the conditions are fulfilled, and the editor won't propose it to the user if it doesn't have to.

The reader is encouraged to experiment himself dynamic content models by referring to the RefleX web site: for example how an algorithmic rule can act on a content model and other various use cases that are available in RefleX.

Semantic, or enhancing the meaning of data types

DTLL, a language for the creation of data type libraries, provides a rather good support for data types. But neither DTLL nor W3C XML Schema nor XML technologies in general offer means to design semantic data types. The semantic of a data type is related to its level of abstractions: Murata defines the following 4 models with different levels of abstractions (we add a 5th at the bottom):

- Model 4: semantic view: 68° Fahrenheit
- Model 3: data type view: #`xs:decimal temp=68`
- Model 2: XML view:



- 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`

The Model 4 is of course the closest to the concerns of a human being. It is available for systems like Active Tags that can express reliationships between datas.

Unfortunately, XML technologies stop at Model 3, got thanks to a schema; what's happenned if we have to sort the following weather report by temperature ?

```
<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>
```

In Java, a simple class with a compator interface would convert the temperature units properly. With XML technologies you won't be able to get the right result; an inadmissible fact face to other technologies that are able to address this issue. Actually, although there are classes of applications where XML is unmarshalled to objects of another language that takes the relay to address this issue, other classes of applications that relies on the XML data model (XDM) such as XSLT and XQuery can't go without such data types[2]. The environment where typed datas are exposed in surface has to play a fundamental role: if OO languages are able to process them, native XML languages (XSLT, XQuery, Active Tags) should support them as well.

The Active Schema Language supplies means to define data types. As expected after validation, an *active sheet* will be able to process the actual typed datas. With the help of XCL, we are able to augment the amount of informations of an XML document the way we like:

```
<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=""
>

  <!--the root element of a weather report-->
  <asl:element name="weather-report" root="always">
    <asl:sequence>
      <asl:element
          ref-elem="city"
          min-occurs="1"
          max-occurs="unbounded"/>
    </asl:sequence>
  </asl:element>

  <!--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>

  <!--#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>

</asl:active-schema>
```

The Active Schema Language (ASL) can defines content models and data types like other schema technologies, and also unlike them ! In the above instance, the typed data that will be bound to the attribute will vary according to the temperature unit used in the XML input document. The following *active sheet* will sort our weather report correctly:

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

  <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>

</xcl:active-sheet>
```

In the result, we notice that the attribute value remains the same, whereas the bound typed data was involved in the sort operation:

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

Notice that as explained in the foundations of the system, the engine could also be launched with a custom catalog that refer to our schema; the <asl:parse-schema> instruction would then be discarded, and the <asl:validate> instruction wouldn't refer to it.

To go further, we could also imagine another semantic data type that would handle a temperature followed immediately with its scale:

```
<city name="London" temp="68°F"/>
```

...and why not allow a mix of the 2 formats in the same document ? ASL support as well this type and types expressed in terms of a choice between several other candidate types.

Those variants are also available at the RefleX web site. The reader is invited to consult the Active Schema Language specification for further informations about semantic data types and polymorphic data types.

Mixins, or collecting schema flavors

Schema mixins can be understand as the ability to mix several schemas. Several solutions, tools, and techniques are already available:

- NVDL (Namespace-based Validation Dispatching Language) consist on separating the input document to validate according to the namespaces encountered and validating each chunk with the appropriate schema. Although several different schema technologies can be involved, for example Relax NG + W3C XML Schema, each will act on a single namespace URI separately.
- W3C XML Schema has a mechanism for importing and including other pieces of schema. But those external parts must be themselves W3C XML Schema.
- Schematron can be embedded inside a W3C XML Schema, but they are not helping each other. Each does its job independently of the other.
- MSV is a Java tool that makes various schema languages converging to a common representation. This allows the same engine to work with DTD, W3C XML Schema, and Relax NG.
- ISO/DSDL part 9 is a draft that brings data types and namespace URIs to DTD. But it is an extension to the DTD language.
- Relax NG and ASL (as shown in the previous sections) can use W3C XML Schema datatypes.

In all that cases, schemas mixins are partially supported. We can't in the same namespace design two parts of a schema with two different schema technologies. What does it serves for ? Well, some people are addicted to DTDs that after all are part of the XML specifications, and DTD content models are in some cases powerful enough. So why moving to W3C XML Schema ? Perhaps to gain a little in expressivity, or to take the benefits of data types. However, even for this last reason, DTD are still used, the more often simply because they are already written. So, we expect from schema mixins a deeper entanglement of several schema technologies.

Let's take back our previous example: a compatible schema could be also expressed with a DTD:

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

The constraints that has been relaxed in this DTD are:

- the temperature is not numeric
- the unit can't be expressed with an enumeration of values because °C and °F are not valid XML tokens
- the relationship between °C and °F can't be expressed

As seen previously, we can design apart with ASL the expected 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:element>

  <!--#temperature is the type introduced in the previous section-->
  <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>

</asl:active-schema>
```

The last piece of the puzzle is to "patch" the DTD with ASL. For this purpose, a third schema specifies how to override the definitions of the DTD:

```
<!--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>
```

Of course, the two ASL schemas could be merged in a single schema, but having two shemas allow a rather good independence between the definitions of the custom types and their usages (the patch operation).

As explained previously, an Active Catalog is used to declare the relevant schemas:

```
<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="" uri="weather-report-master.asl" selector="asl:schema"/>
    <cat:resource name="" uri="weather-report-datatypes.asl" selector="asl:schema"/>
    <!--asl:schema is the selector for all kind of schemas: DTD, ASL, W3C XML Schema,
    Relax NG, others -->
    <cat:resource name="" uri="weather-report-legacy.dtd" selector="asl:schema"/>
</cat:catalog>
```

We have already seen that an *active sheet* that performs a validation don't need to refer to a parsed schema: the engine will lookup in its catalogs for the relevant resources. With RefleX, the command line interface and the servlet allow to run an *active sheets* with a given set of catalogs. Notice that the order where the resources appear is important: the definitions in the master schema mask those in the legacy DTD. This is a strategy proper to schema lookup, and other kind of resources (modules) have their own lookup strategy. Details are available in the Active Catalog specification.

Other features are available: for example, before applying or after applying the definitions of the DTD, some content models might be prepend or append to the content model of the DTD if needed. But a more complex refactorisation of the content model expressed in the DTD wouldn't be possible without overwriting it entirely.

Unlike the previous examples, mixins are in progress in ASL and are not covered by the implementation.

Basically, they work like imports and includes in W3C XML Schema but it demonstrates clearly that a system based on schema cooperation allow to deal with legacy schemas. Moreover, this is a not intrusive technique unlike ISO/DSDL part 9 mentioned earlier that require to rewrite DTD interpreters. Additionally, with the help of catalogs, we have a schema machinery that prefer to rely on dedicated components rather than trying to do everything itself.

Hyperschemas, or validating high-level XML components

In the same way that there is a Relax NG schema for Relax NG, a W3C XML Schema schema for W3C XML Schema, let's try to write the ASL schema of ASL.

An element definition is composed of attribute definitions or references, choices, sequences, etc. Let's start to write it:

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

  <asl:element name="asl:element">
    <asl:attribute name="name" ref-type="xs:string"/>
    <asl:choice min-occurs="0" max-occurs="unbounded">
      <asl:element ref-elem="asl:attribute"/>
      <asl:element ref-elem="asl:choice"/>
      <asl:element ref-elem="asl:sequence"/>
      <!--other stuff here-->
    </asl:choice>
  </asl:element>

</asl:active-schema>
```

However, we have seen that ASL doesn't work alone: we can inject foreign instructions that the engine will use to build the content model. We could then append in the <asl:choice> a reference to the XCL namespace, which can be written like this:

```
      <!--any element in the XCL namespace-->
      <asl:element ref-ns="xcl"
                   xmlns:xcl="http://ns.inria.org/active-tags/xcl"/>
```

...but it is not enough. In fact, since we don't know how the user will define its content model, almost anything should be allowed: we can't make assumptions about which tag will help him and which one won't. The content model would become:

```
  <asl:element name="asl:element">
    <asl:attribute name="name" ref-type="xs:string"/>
    <asl:choice min-occurs="0" max-occurs="unbounded">
      <asl:element ref-ns="#any"/>
    </asl:choice>
  </asl:element>
```

...which means basically: well, everything is accepted. Not so useful. This is the downside of the system: since almost everything is dynamic, we can't predict which tag will come. Yet some are acceptable, others aren't.

On the opposite, we do know after running the content model (not this one but those with the if-then-else statement shown in the chapter about dynamicity) that the realization of the schema must conform to the schema that we started to write above (not those with ref-ns="#any", but the one before). Unfortunately, it is designed to validate XML (its static representation), not to validate the underlying software components (its dynamic representation): element definitions, sequences, choices are software components whose relationships are expressed with XML tags. We know how to express constraints on XML tags, but we don't know how to express constraints on such pluggable components because although they are exposed as XML tags they are no longer XML tags at runtime.

ASL doesn't face this kind of validation. We could imagine schemas that would act on a higher dimension of validation -hyperschemas- and schemas that would act on both level, multidimensional schemas. This is a funny thought that was encountered while designing Active Tags, and that of course can be generalized for components that are not related to schemas: in many cases, the author had to express components contents in terms of parts that are static and validable with a schema, and parts that are dynamic and validable with an hyperschema.

So far, no acceptable solution was found.

Conclusion and perspectives

Hyperschemas don't have obviously a practical field of application out of the scope of Active Tags. However, they pose the right questions to similar systems that would need more flexibility and dynamicity in software components assembly.

On the opposite, the other use cases exposed (dynamic content models, semantic typed data, and mixins) are more pragmatic since they address elegantly and efficiently common issues in schema technologies.

We have seen that active content models enhance the expressiveness of schemas, that if a data has some meaning for you, it should have some meaning too for the applications that process it, and that mixins allow DTD-nostalgic people to leverage their usage.

Through those few manifestations of "schemas mashups", we have to admit that there are better solutions than running schemas alone. Actually, ASL is not so different from other schema technologies (DTD, W3C XML Schema, Relax NG, Schematron) but immensely more powerful. By extension, ASL should opens the perspective for declarative languages in general: Active Tags combine a bunch of XML technologies that would help significantly the designers of runnable markup languages.

Bibliography

[Active Catalog] Poulard, P. *Active Catalog*.http://ns.inria.fr/active-tags/active-catalog/active-catalog.html.

[Active Tags] Poulard, P. *Active Tags technologies*.http://ns.inria.org/active-tags/.

[ASL] Poulard, P. *The Active Schema Language*.http://ns.inria.fr/active-tags/active-schema/active-schema.html.

[ISO/DSDL part 9] *DSDL (Document Schema Definition Languages) — Part 9: Namespace- and datatype-aware DTDs*, ISO/IEC CD 19757-9.http://dsdl.org/dsdl-9-rev061103.pdf.

[DTLL] Tennison J. (2006). *Datatypes for XML: the Datatyping Library Language (DTLL)*. In Proceedings of Extreme Markup Languages, Montréal, Canada. http://www.idealliance.org/papers/extreme/proceedings/html/2006/Tennison01/EML2006Tennison01.html.

[Jelly] *Jelly: Executable XML*. http://jakarta.apache.org/commons/jelly/.

[JSP] *JSP: JavaServer Pages Technology*. http://java.sun.com/products/jsp/.

[JSTL] *JSTL: JavaServer Pages Standard Tag Library*. http://java.sun.com/products/jsp/jstl/.

[Murata] Murata, M. (2002). *Principles of Schema Languages*. In H. Maruyama (Ed.), *XML and Java (2nd ed.)* (pp. 592-601). Boston, MA: Pearson Education.

[MSV] Kawaguchi, K. *Sun Multi-Schema Validator*. https://msv.dev.java.net/.

[NVDL] *NVDL: Namespace-based Validation Dispatching Language*, ISO/IEC 19757-4 NVDL. http://www.nvdl.org/.

[OASIS XML Catalogs] Walsh, N. (2005). *XML Catalogs*, OASIS Standard V1.1.http://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html.

[Poulard] Poulard, P. (2007). *Active Tags: Mastering XML with XML*. In Proceedings of Extreme Markup Languages, Montréal, Canada. http://www.idealliance.org/papers/extreme/proceedings/html/2007/Poulard01/EML2007Poulard01.html.

[RefleX] Poulard, P. *RefleX: An Active Tags engine in Java*.http://reflex.gforge.inria.fr/.

[Relax NG] *Relax NG: Regular-grammar-based validation*, ISO/IEC FDIS 19757-2.http://www.y12.doe.gov/sgml/sc34/document/0362_files/relaxng-is.pdf.

[SCXML] *SCXML: State Machine Notation for Control Abstraction*, W3C Working Draft. http://www.w3.org/TR/scxml/.

[Schematron] Jelliffe, R. *Schematron: A language for making assertions about patterns found in XML documents*.http://www.schematron.com/spec.html.

[UEL] *UEL: Unified Expression Language*. http://java.sun.com/products/jsp/reference/techart/unifiedEL.html.

[WUnit] Poulard, P. *WUnit: Unit tests for Web applications*. http://reflex.gforge.inria.fr/wunit.html.

[W3C XML Schema] *XML Schema Part 1: Structures Second Edition*, W3C Recommendation.http://www.w3.org/TR/xmlschema-1/.

[W3C XML Schema: Datatypes] *XML Schema Part 2: Datatypes (2nd ed.)*, W3C Recommendation. http://www.w3.org/TR/xmlschema-2/.

[W3C XML Schema 1.1] *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*, W3C Working Draft.http://www.w3.org/TR/xmlschema11-1/.

[XCL] Poulard, P. *The XML Control Language*.http://ns.inria.org/active-tags/xcl/xcl.html.

[XDM] *XQuery/XPath Data Model (XDM) 1.0*, W3C Recommendation. http://www.w3.org/TR/xpath-datamodel/.

[XPath] *XML Path Language*, W3C Recommendation. http://www.w3.org/TR/xpath.

[XProc] *XProc: An XML Pipeline Language*, W3C Working Draft.http://www.w3.org/TR/2006/WD-xproc-20061117/.

[XQuery] *XQuery 1.0: An XML Query Language*, W3C Recommendation. http://www.w3.org/TR/xquery/.

[XSLT] Clark, J. (1999). *XSL Transformations (XSLT) 1.0*, W3C Recommendation.http://www.w3.org/TR/xslt.

[XUnit] Poulard, P. *XUnit: XML-based unit tests*. http://reflex.gforge.inria.fr/xunit.html.

[1] X-operable objects are objects that are exposing their inherent properties as XML properties (name, attributes, children, parent...) and can be processed with XML-related operations and traversed with XPath. For example, one can apply //* on a directory of a file system to get all the files and directories under the tree.

[2] It might be possible to create the target XML result with XQuery, but not to operate the input XDM properly as expected if it doesn't support such data types. Such a query would be like a procedure aside from the typed data, whereas ASL can define a type where the datas and the behaviour are bundled together, like in OO designs.

Philippe Poulard
INRIA
<philippe.poulard@sophia.inria.fr>

Philippe Poulard is a software engineer at INRIA (french national institute for research in computer science and control) where he is involved in Web-oriented problematics. He has been specialized in XML technologies and e-documentation for 10 years. During this period, he has developed XML and SGML-based solutions and prototypes on behalf of the French Army and INRIA. More recently he has designed and implemented a set of XML technologies named "Active Tags" (http://ns.inria.org/active-tags/). He also teaches XML and Java at Nice/Sophia-Antipolis university and Aix/Marseille university. He has an engineer degree (M.Sc) from the Conservatoire National des Arts et Metiers.