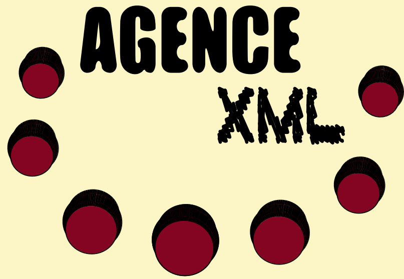
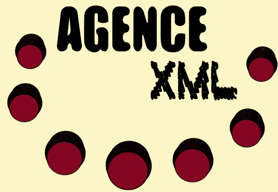


Compiling XQuery code into Javascript instructions using XSLT

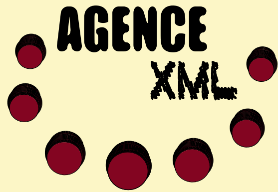
XML Prague 2012

alain.couthures@agencexml.com



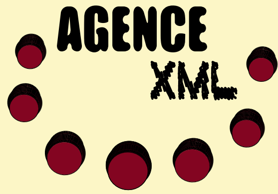
XQuery at browser-side

- What for?
  - high-level XML manipulation
  - same language at server and at client side
  - rich actions in XForms (XPath 2.0 for XForms 2.0)
  - not (yet) for big data
- How to interpret XQuery code?
  - Plug-in
  - Java to Javascript compiler (GWT)



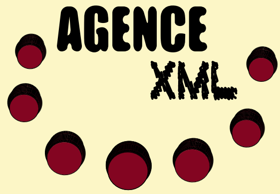
Compiler vs. Interpreter

- Javascript is still very slow
- Dynamic evaluation is rare and evil?
- Produced instructions are optimized
- Produced instructions include objects and function calls: a run-time library is required



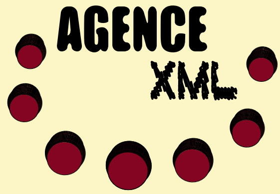
How to write an XQuery Compiler

1. Which programming language to use?
2. Are there similar projects/products?
3. What input format for XQuery grammar?
4. What intermediate XML notation before generating Javascript code?



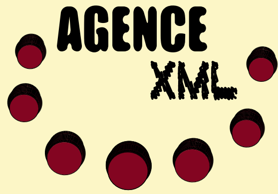
XSLT as a compiler language

- XSLT 1.0 is everywhere!
- XSLT is very powerful
- XSLT engines are fast
- XSLTForms already includes its own XPath 1.0 compiler written in XSLT



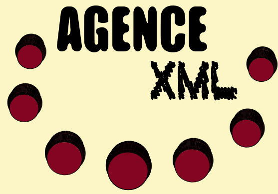
XSLTForms XPath 1.0 Engine

- Each XPath is compiled into a Javascript object
- Compilation at server or at client side
- Error report
- Dependencies detection at run-time
- Limits:
  - XSLT templates hard to maintain
  - Javascript object evaluation not optimized



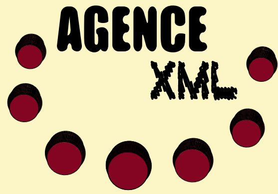
XSLTForms Example

```
new LocationExpr(true,
  new StepExpr('child',
    new NodeTestName('','data')
  ),
  new StepExpr('child',
    new NodeTestName('','PersonGivenName')
  )
)
```



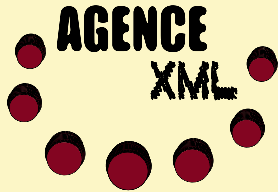
XSLTForms template

```
<xsl:choose>
    <xsl:when test="contains('./@*',$c)">
        <xsl:variable name="t">
            <xsl:call-template name="getLocationPath">
                <xsl:with-param name="s" select="concat($c,$d)"/>
            </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="substring-before($t,'.')"/>
        <xsl:text>.new LocationExpr(</xsl:text>
        <xsl:choose>
          <xsl:when test="$c = '/' and not(starts-with($ops,'3.0./'))">true</xsl:when>
          <xsl:otherwise>false</xsl:otherwise>
        </xsl:choose>
        <xsl:value-of select="substring-after($t,'.')"/><xsl:text>)</xsl:text>
    </xsl:when>
```



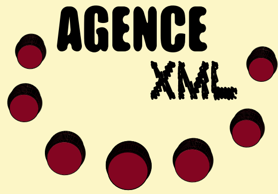
YAPP XSLT : a BNF engine (http://www.o-xml.org/yapp/)

- An XML notation for a BNF grammar
- XSLT 1.0 stylesheets to generate a parser for the grammar
- The generated parser is an XSLT 1.0 stylesheet!
- A BNF parser is included!
- Limits:
  - Uses a tokenizer
  - Heavy use of node-set() extension



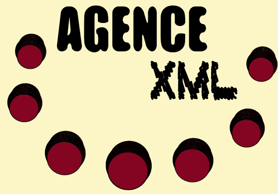
YAPP Grammar

```
<construct name="AbsoluteLocationPath">
  <option>
    <part name="slash"/>
    <part name="RelativeLocationPath"/>
  </option>
  <option>
    <part name="slashslash"/>
    <part name="RelativeLocationPath"/>
  </option>
  <option>
    <part name="slash"/>
  </option>
</construct>
```



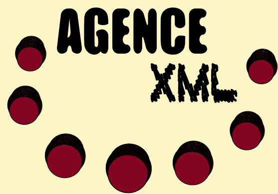
YAPP Lexer

```
<xsl:template name="t:nextToken">
  <xsl:param name="in"/>
  <xsl:choose>
    <xsl:when test="string-length($in) = 0">
      <token type="end"/>
    </xsl:when>
    <xsl:when test="starts-with($in, ' ')">
      <xsl:call-template name="t:nextToken">
        <xsl:with-param name="in" select="substring($in, 2)"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="starts-with($in, 'descendant-or-self::')">
      <token type="axisName">
        <xsl:value-of select="substring($in, 1, 20)"/>
      </token>
      <remainder>
        <xsl:value-of select="substring($in, 21)"/>
      </remainder>
    </xsl:when>
```



YAPP Parser

```
<xsl:template name="p:AbsoluteLocationPath">
  <xsl:param name="in"/>
  <xsl:variable name="option-slash-RelativeLocationPath">
    <xsl:variable name="part1">
      <xsl:call-template name="p:slash">
        <xsl:with-param name="in" select="$in"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:copy-of select="xalan:nodeset($part1)"/>
    <xsl:variable name="part2">
      <xsl:if test="xalan:nodeset($part1)/remainder">
        <xsl:call-template name="p:RelativeLocationPath">
          <xsl:with-param name="in" select="xalan:nodeset($part1)/remainder"/>
        </xsl:call-template>
      </xsl:if>
    </xsl:variable>
    <xsl:copy-of select="xalan:nodeset($part2)"/>
  </xsl:variable>
```



XPath/XQuery Applets

http://www.w3.org/2011/08/qt-applets/

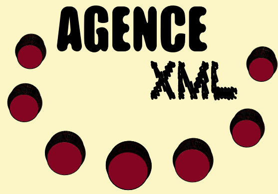
Grammar Test Page for XQuery 1.0

Type in a XQuery 1.0 expression then click on the button:

Parse | Translate to XQueryX

```
/data/PersonGivenName
```

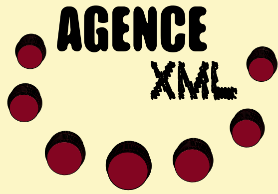
```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                        http://www.w3.org/2005/XQueryX/xqueryx.xsd">
 <xqx:mainModule>
  <xqx:queryBody>
   <xqx:pathExpr>
    <xqx:rootExpr/>
    <xqx:stepExpr>
     <xqx:xpathAxis>child</xqx:xpathAxis>
     <xqx:nameTest>data</xqx:nameTest>
    </xqx:stepExpr>
    <xqx:stepExpr>
     <xqx:xpathAxis>child</xqx:xpathAxis>
     <xqx:nameTest>PersonGivenName</xqx:nameTest>
```



XQuery Grammar

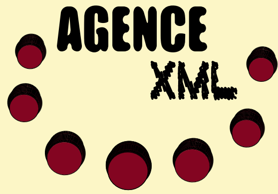
the "xpath-grammar.xml" file

- An XML notation for an EBNF grammar
- A unique file for all versions of XPath and XQuery
- Extra data for JavaCC processing



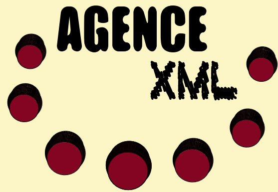
Grammar Languages

<g:language id="**xpath20**" display-name="**XPath 2.0**" if="xpath20"/>
<g:language id="**xpath30**" display-name="**XPath 3.0**" if="xpath30"/>
<g:language id="**xpath1**" display-name="**XPath 1.0**" if="xpath1"/>
<g:language id="**xquery10**" display-name="**XQuery 1.0**" if="xquery10"/>
<g:language id="**xquery30**" display-name="**XQuery 3.0**" if="xquery30"/>
<g:language id="**fulltext**" display-name="**XQuery Full-Text 1.0**" if="fulltext"/>
<g:language id="**xcore**"
display-name="**XML Processing Formal Semantics Core Language 1.0**" if="xcore"/>
<g:language id="**xslt2-patterns**" display-name="**XSLT 2.0 Match Patterns**"
if="xslt2-patterns"/>
<g:language id="**scripting**" display-name="**XQuery Scripting Extension 1.0**"
if="scripting"/>
<g:language id="**update**" display-name="**XQuery Update Facility 1.0**" if="update"/>



Grammar Entry Points

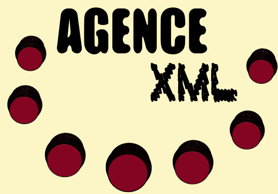
```
<g:start name="ExprSingle" state="DEFAULT" if="xpath1"/>
<g:start name="XPath" state="DEFAULT" if="xpath20 xpath30"/>
<g:start name="Expr" state="DEFAULT" if="xcore"/>
<g:start name="QueryList" state="DEFAULT" if="xquery10 xquery30"/>
<g:start name="Pattern" state="DEFAULT" if="xslt2-patterns"/>
```



Grammar production

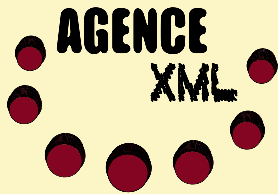
ModuleDecl ::= "module" "namespace" NCName "=" URILiteral Separator

```
<g:production name="ModuleDecl" if="xcore xquery10 xquery30">
  <g:string>module</g:string>
  <g:string>namespace</g:string>
  <g:ref name="NCName"/>
  <g:string>=</g:string>
  <g:ref name="URILiteral"/>
  <g:ref name="Separator"/>
</g:production>
```



XQuery Grammar vs. XQueryX

- Different names:
  - "**FLWORExpr**" vs. "**flworExpr**"
  - "**ForBinding**" vs. "**forClauseItem**"
  - "**DirElemConstructor**" vs. "**elementConstructor**"
- Grouping
  - "**ForwardAxis**" and "**ReverseAxis**" vs. "**xpathAxis**"

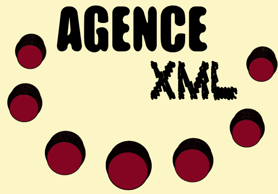


Implementation Rules for the Compiler

- An XSLT 1.0 stylesheet to transform the XQuery Grammar into another XSLT 1.0 stylesheet:

  A named template for each production

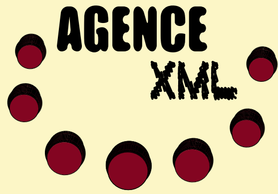
- No node-set() call
- Error reporting
- XQueryX as intermediate XML notation



Formatted Strings

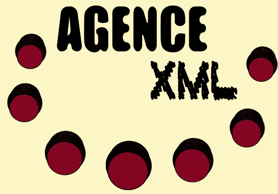
Each generated template returns a string:

- Constant string: "1.0.ends-at.length.value"
- Production: "1.1.ends-at.length.name.value"
- Error: "0.position.error-message"



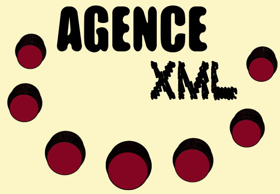
"g:string" Parsing

```
<out:choose>
  <out:when test="starts-with(substring($s,$var_sep{$prefix}{count(preceding-sibling::*)}),'{.}'){$ahead}">
    <out:variable name="var{$prefix}{count(preceding-sibling::*)}">
      <out:text>1.0.</out:text>
      <out:value-of select="$var_sep{$prefix}{count(preceding-sibling::*)} + {string-length(.)}"/>
      <out:text>.<xsl:value-of select="string-length(.)"/>.<xsl:value-of select="."/></out:text>
    </out:variable>
    <xsl:call-template name="nextelt">
      <xsl:with-param name="prefix" select="$prefix"/>
      <xsl:with-param name="offset" select="$offset"/>
    </xsl:call-template>
  </out:when>
  <out:otherwise>
    <out:text>0.</out:text>
    <out:value-of select="$var_sep{$prefix}{count(preceding-sibling::*)}"/>
    <out:text>."<xsl:value-of select="."/>" expected</out:text>
  </out:otherwise>
</out:choose>
```



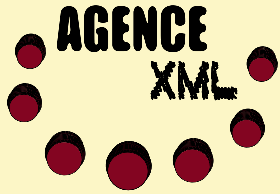
Generated templates

```
<xsl:choose>
   <xsl:when test="starts-with(substring($s, $var_sep0),'xquery') and
                    not(contains($charrange, substring($s, $var_sep0 + 6, 1)))">
    <xsl:variable name="var0">
     <xsl:text>1.0.</xsl:text>
     <xsl:value-of select="$var_sep0 + 6" />
     <xsl:text>.6.xquery</xsl:text>
    </xsl:variable>
    <xsl:variable name="var1">
     <xsl:variable name="var1_0">
      <xsl:variable name="var_sep1_0_0">
       <xsl:call-template name="ltrim">
        <xsl:with-param name="s" select="$s" />
        <xsl:with-param name="pos" select="substring-before(substring($var0, 5), '.')" />
       </xsl:call-template>
      </xsl:variable>
```



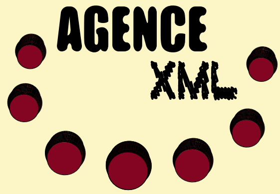
XSLT Generation Limits

- Terminal elements (**charCode**, **charCodeRange**)
- Regular expressions (**PragmaContent**, **XmlPIContentBody**, **CdataSectionBody**, **CommentContentBody**)
example: **(Char* ('(:' | ':)') Char*)**



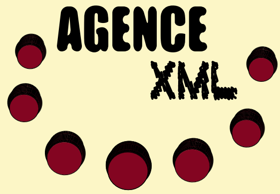
How to validate the parser?

1. Apply the parser to the XQuery test case
2. Transform the resulting XQueryX into XQuery using the W3C XSLT stylesheet (http://www.w3.org/TR/xqueryx/#Stylesheet)
3. Run the test case
4. Run the transformed result
5. Compare



Conclusions about the parser

- Performance to be verified:
  - **Big simple generated templates vs. compact ones**
  - **Possible difficulties ("/ * /*" vs. "/*/*", …)**
- Less complex than XSLTForms XPath 1.0 parser, more reliable and extensible



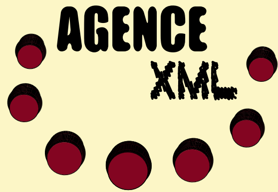
From XQueryX to Javascript

- A Javascript object for each non-leaf element:

  As simple as the identity transformation

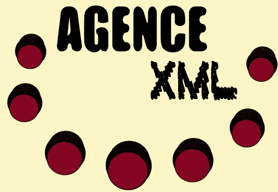
- Instructions, variables and loops:

  Optimized for different situations



Run-time Javascript Library

- Typed values have to be stored within Javascript variables:
  - **Atomic value as a JSON object (type and value)**
  - **Sequence as an array**
  - **Fragment as XML element**
  - **Function as a JSON object**
- Functions have to be defined for standard functions, operators and repetitive treatments



Conclusion

- An ambitious project on track
- Javascript is always faster and machines are more powerful!
- XSLT 1.0 in browsers:
  - Firefox (namespace axis support): https://bugzilla.mozilla.org/show_bug.cgi?id=94270
  - Android (support in Chrome Beta): http://code.google.com/p/android/issues/detail?id=9312