Μετασχηματισμός XSLT

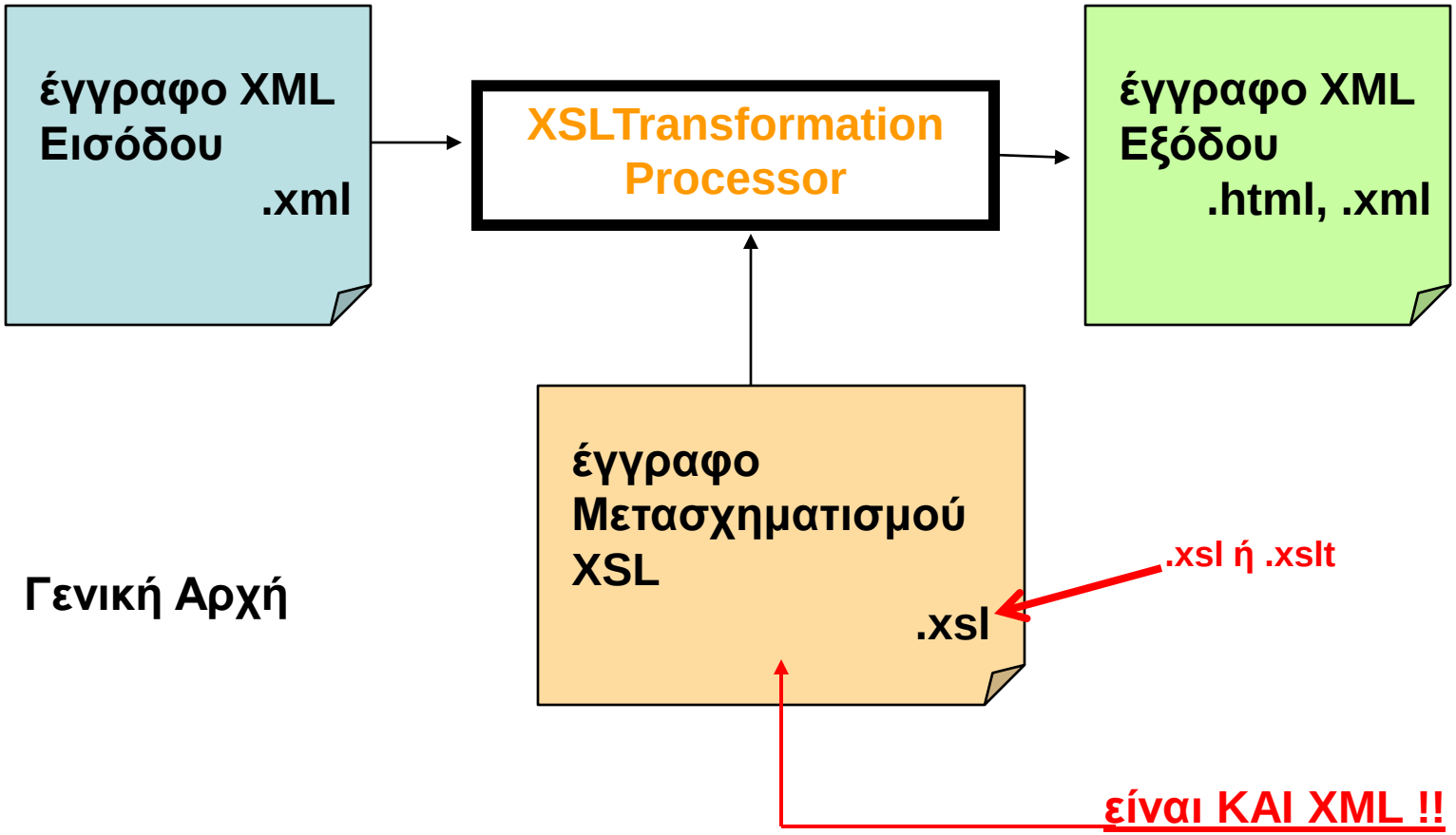
XML StyleSheet Transformation

http://www.w3.org/TR/xslt ισχύον 1999

Μετασχηματισμός XSLT

XSL : XML **S**tylesheet **L**anguage

XSLT: XML **S**tylesheet **L**anguage **T**ransformation

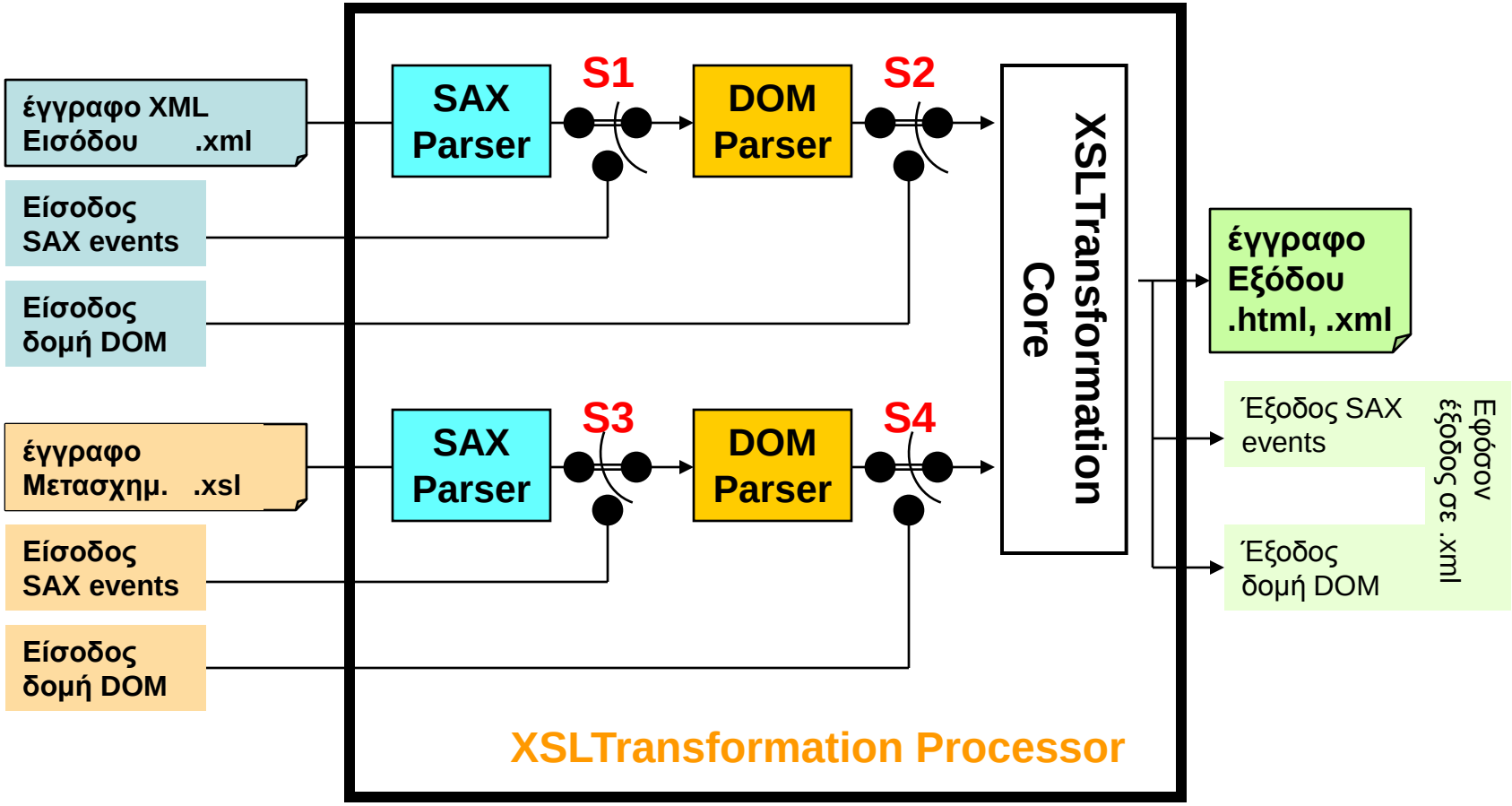


Γενική Αρχή

Μετασχηματισμός XSLT

Εσωτερική δομή

- η είσοδος .xml και ο μετασχηματισμός .xsl είναι .xml έγγραφα
- η πρόσβαση σε αυτά κατά τα γνωστά (αρχείο η SAX events η DOM στην μνήμη)



MP7 in XML

MP7 in XML – XSLTransformation

Το τραγούδι, σαν XML, φέρει μαζί του metadata κατά MP7

Με XSLT μπορούμε να

ψάξομε / εξάγομε / επισημάνομε σημεία στο XML

μετατρέψομε / εμπλουτίσομε το XML με πρόσθετη δική μας πληροφορία / ευρήματα / συμπεράσματα

π.χ. εμπλούτισε το XML με τα λόγια ΚΑΙ σε μία πρόσθετη γλώσσα

Γενική Δομή .xsl

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```
<xsl:template match="/">
    …
    <xsl:apply-templates select = ".. some XPath expression.."/>
    …
</xsl:template>
```

… περισσότερα τέτοια templates …

```
<xsl:template match="… a XPath expression..">
    …
    <xsl:apply-templates select = ".. another XPath expression "/>
    …
</xsl:template>
```

</xsl:stylesheet>

Παρατηρήσεις, Γενικές ιδιότητες

1. Το .xsl είναι πάντα και έγγραφο .xml (βλ. πρώτη σειρά)

2. Το xsl:stylesheet element είναι το document element (μοναδικό)

3. Τα xsl:template elements είναι παιδιά του stylesheet

4. Δεν υπάρχουν templates μέσα σε template (no template nesting)

5. Υπάρχει πάντα σαν default ένα <xsl:apply-templates select = “/”/>

δηλ. χωρίς δική μας φροντίδα η εκτέλεση αρχίζει με το template που έχει το attribute match=“/”

Συνήθως φροντίζομε να υπάρχει ένα τέτοιο και από εκεί αρχίζουν όλα.

Πρώτο (απλό) Παράδειγμα Μετασχηματισμού

με τον XSLTransformation Processor ενσωματωμένο στον **browser**

Στο generalXML.xml γράφομε στην κορυφή 2 πρόσθετες γραμμές

```
<?xml version="1.0"?>
<!-- REFERENCE TO THE STYLESHEET -->
<?xml-stylesheet href="firstXSLT.xsl" type="text/xsl"?>
... κλπ
```

το οποίο είναι directive προς τον browser (δηλ. εκείνον που θα εμφανίσει το έγγραφο) να χρησιμοποιήσει το firstXSLT.xsl για να παρουσιάσει το περιεχόμενο.

Προς το παρόν μπορούμε να περιορισθούμε στο ότι το έγγραφο εξόδου θα είναι απλό κείμενο

Πρώτο (απλό) Παράδειγμα Μετασχηματισμού

Το XML μετασχηματισμού: **"firstXSLT.xsl"**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<xsl:apply-templates select = "*"/>
</xsl:template>

<xsl:template match="/*">
The name of the document element is
 <xsl:value-of select="name()"/>
 <xsl:apply-templates select ="/vehicles/*/car"/>
</xsl:template>
```

Κτυπάμε το generalXML.xml π.χ. με τον IE5 και βλεπομε το αποτέλεσμα

Δοκιμάζομε με IE5 και FireFox

Πρώτο (απλό) Παράδειγμα Μετασχηματισμού

Πως το τρέχομε:

Α τρόπος (όπως είδαμε)

Με τον XSLTransformation Processor που προϋπάρχει στον **browser**

Β τρόπος

Με τον **xalan** (εγκατάσταση πιο κάτω) από command window εντολή:

```
java org.apache.xalan.xslt.Process  -IN generalXML.xml -XSL firstXSLT.xsl - OUT res.txt
```

...και δημιουργείται το res.txt σύμφωνα με τις επιθυμίες μας

Τώρα είναι αδιάφορο αν στο generalXML.xml υπάρχει αναφορά στο .xsl

Πρώτο (απλό) Παράδειγμα Μετασχηματισμού

Β τρόπος (συνέχεια) - **xalan**

από command window

```
java org.apache.xalan.xslt.Process -TT  -IN friends.xml  -XSL addressFormer.xsl - OUT res.txt
```

-IN έγγραφο XML Εισόδου σε .xml
-OUT έγγραφο XML Εξόδου σε .html, .xml, κλπ (εδώ διαλέγομε .txt) – αν λείπει, έξοδος στην κονσόλα
-XSL έγγραφο Μετασχηματισμού XSL σε .xsl
-TT Trace, βλέπομε τους κόμβους από τους οποίους περνά ο μετασχηματισμός

Ο Xalan-Java (ή Java – υλοποίηση, υπάρχει και Xalan-C) χρησιμοποιεί τον Xerces parser και τη Java του, by default.

Μπορούμε όμως να τον μορφοποιήσομε έτσι ώστε να χρησιμοποιεί το περιβάλλον της Java και την γλώσσα την ίδια και για τους parsers SAX και DOM που έχει η Java ενσωματωμένους.

.. και όταν δεν προσδιορίζομε μετασχηματισμό ;;

... πως παρουσίαζε μέχρι τώρα ο IE5 ένα έγγραφο .xml ;;

Εάν δεν έχομε εμείς προσδιορίσει κάποιο έγγραφο .xsl

(στην αρχή, μέσα στο .xml) ,

ο IE5 χρησιμοποιεί ένα δικό του default style sheet, το defaultss.xsl.

Για να το δούμε πάμε με τον IE5 στο **res://msxml.dll/defaultss.xsl**

(επιπλέον, πέραν της παρουσίασης, μας επιτρέπει με ενσωματωμένα scripts το ανοιγοκλείνομε τα στοιχεία)

Περιγραφή του πως εφαρμόζεται το .xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<xsl:apply-templates select = "*"/>
</xsl:template>

<xsl:template match="/*">
The name of the document element is
 <xsl:value-of select="name()"/>
 <xsl:apply-templates select ="/vehicles/*/car"/>
</xsl:template>

<xsl:template match="car">
The number of doors is
 <xsl:value-of select="@no_doors"/>
</xsl:template>

<!-- java org.apache.xalan.xslt.Process -TT -IN generalXML.xml -XSL firstXSLT.xsl -OUT out.txt -->

</xsl:stylesheet>
```

Μετατρέπομε με το παραπάνω stylesheet, ονομασμένο firstXSLT.xsl, το έγγραφο generalXML.xml και λαμβάνομε το out.txt

Αρχή Λειτουργίας XSLT Processor

Υποθέτομε ότι ο XSLT Processor ευρίσκεται σε κάποιον **κόμβο αναφοράς** (**context node**) από το '**τρέχον σύνολο κόμβων**' (**current node set**). Αρχικά αυτό το σύνολο αυτό είναι η ρίζα "/" του κειμένου XML. Κατόπιν

(α) ο XSLT Processor, για κάθε κόμβο "**Χ**" του **τρέχοντος συνόλου κόμβων** ψάχνει για τα στοιχεία <xsl:template match="pattern"> του stylesheet (αρχείο .xsl), τα οποία δυνητικά ταιριάζουν στον κόμβο αυτό. Από αυτά επιλέγει εκείνο πού προσφέρει το πλησιέστερο ταίριαγμα.

(β) το επιλεγέν <xsl:template match="pattern"> εφαρμόζεται, όπου ο "Χ" γίνεται τώρα ο **κόμβος αναφοράς**.

(γ) εάν κατά τη εκτέλεση του (β) ευρεθεί <xsl:apply-templates select="node-set-expression"> δημιουργείται ένα νέο '**τρέχον σύνολο κόμβων**' και η διαδικασία συνεχίζεται αναδρομικά.

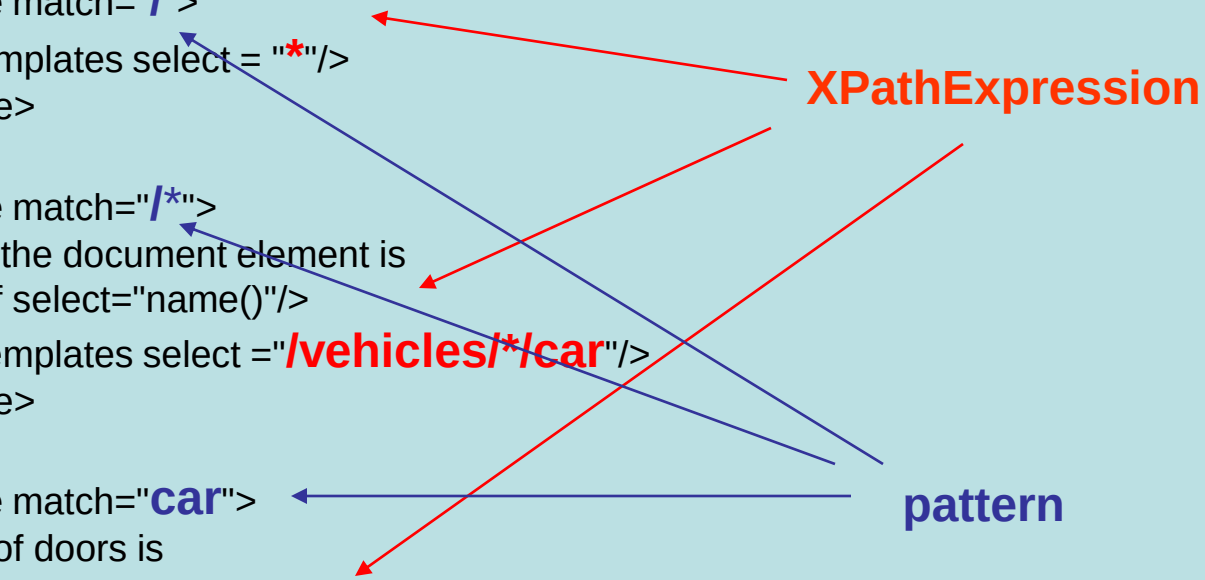
Το παραπάνω "node-set-expression" του (γ) ανευρίσκεται με το XPath
- **σχετικά** ως προς τον **κόμβο αναφοράς** "Χ" αν η έκφραση **δεν αρχίζει με '/'** , ή
- **απόλυτα** (ως προς την ρίζα) αν η έκφραση **αρχίζει με '/'**

Context Node και Current Node Set

συνεχώς αλλάζουν (π.χ. παράδειγμα)

(firstXSLT.xsl, πάνω στο generalXML.xml)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<xsl:apply-templates select = "*"/>
</xsl:template>

<xsl:template match="/*">
The name of the document element is
 <xsl:value-of select="name()"/>
 <xsl:apply-templates select ="/vehicles/*/car"/>
</xsl:template>

<xsl:template match="car">
The number of doors is
 <xsl:value-of select="@no_doors"/>
</xsl:template>

<!-- java org.apache.xalan.xslt.Process -TT -IN generalXML.xml -XSL firstXSLT.xsl -OUT out.xml -->

</xsl:stylesheet>
```



Ένα pattern ορίζει ένα σύνολο συνθηκών για ένα κόμβο

Ο κόμβος που εκπληρει τις συνθήκες αυτές ταιριάζει (match) στο pattern

Οι δυνατές εκφράσεις για την διατύπωση ενός pattern αποτελούν υποσύνολο αυτών που μπορούν να χρησιμοποιηθούν για μια XPathExpression

Ένα pattern επιστρέφει πάντα ένα node-set
Μία XPathExpression μπορεί να επιστρέφει και , π.χ. number (π.χ. μέσω count())

A node matches a pattern if the node is a member of the result of evaluating the pattern as an expression with respect to some possible context; the possible contexts are those whose context node is the node being matched or one of its ancestors.

Context Node και Current Node Set

Χρήση του –TT switch του Xalan

```
java org.apache.xalan.xslt.Process -TT -IN friends.xml  -XSL addressFormer.xsl - OUT res.txt
```

Εκτυπώνει στην κονσόλα το εκάστοτε template το οποί εφαρμόζεται και μας βοηθά στην κατανόηση της αρχής λειτουργίας του XSLT Processor

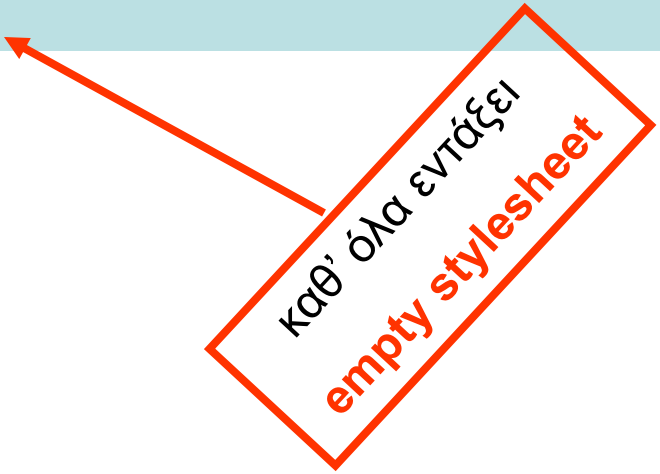
Χρήσιμο και για debugging, βλέπομε από που περνάμε!

ενσωματωμένα templates

Δοκιμάζομε το ‘άδειο’ stylesheet (**empty.xsl**)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Πάνω στο έγγραφο generalXML.xml



…….και κάτι βγαίνει! Γιατί;

Συνυπάρχουν πάντα τα εξής

built-in (ενσωματωμένα) templates:

σειρά προτεραιότητας εφαρμογής

```
<xsl:template match="*|/">
   <xsl:apply-templates/>
</xsl:template>
```

ταιριάζει σε οποιοδήποτε κόμβο στοιχείου (element) (*** OR ρίζα /**)

```
<xsl:template match="*|/" mode="m">
   <xsl:apply-templates mode="m"/>
</xsl:template>
```

ως άνω, αλλά σε σχέση με οποιοδήποτε **mode**

```
<xsl:template match="text()|@*">
   <xsl:value-of select="."/>
</xsl:template>
```

ταιριάζει σε **οποιοδήποτε** κόμβο **text και attribute**

```
<xsl:template
  match=”processing-instruction()|comment()”/>
```

αποτέλεσμα = τίποτα (**PI ή comment vanish**)

τα ενσωματωμένα templates

Εφάρμοσε το άδειο emptyXSLT.xsl επί του generalXML.xml
και δες τι κάνουν τα ενσωματωμένα templates:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- java org.apache.xalan.xslt.Process -TT -IN generalXML.xml -XSL
emptyXSLT.xsl -OUT out.xml -->

</xsl:stylesheet>
```

Διαφορά μεταξύ

<xsl:apply-templates **select** = “/book/chapter/par”/> και

<xsl:apply-templates/>

Με το attribute **select** καθορίζομε μέσω μίας XPath Expression τους κόμβους εκείνους στους οποίου θα πρέπει να εφαρμοσθεί (αν ταιριάζει) κάποιο από όλα τα υπάρχοντα templates (συμπεριλαμβανομένου και αυτού που περιέχει το εν λόγω xsl:apply-templates).

Χωρίς το **select** επιλέγονται **όλοι οι κόμβοι** στοιχείου που είναι **παιδιά** του τρέχοντος κόμβου (all element children of the current node) στα οποία και πάλι πρέπει ….. (ως άνω)

Παράδειγμα

```
<! -- assume par as children of chapter -->
< xsl:template match = “/book/chapter”/>
   <xsl:apply-templates/> <!-- the par elements are selected -->
<xsl:template>
```

Το παραπάνω template ΔΕΝ θα κληθεί αναδρομικά

Ισοδύναμο (στην περίπτωση αυτή)

```
<xsl:template match = “/book/chapter” />
   <xsl:apply-templates select = “*“/> <!-- same effect -->
<xsl:template>
```

Πάντα **<xsl:apply-templates** ..../> **(άνοιγμα / κλείσιμο χωρίς περιεχόμενο) ;;;**

- Συνήθως ΝΑΙ

- Αλλά δυνατόν να θέλομε ταξινομημένη εφαρμογή <xsl:sort>

```
<xsl:template match = “/book/chapter”>
   <xsl:apply-templates select=“/book/chapter/par”><! -- open to put content -->
   <xsl:sort select=“par/@no" />                    <! -- content (=child) -->
   </xsl:apply-templates>                           <!-- close -->
<xsl:template>
```

Το xsl:sort παιδί του xsl:apply-templates και καθορίζει περαιτέρω τον τρόπο εφαρμογής του

Άλλη περίπτωση ανοίγματος / περιεχομένου / κλεισίματος η **κλήση με παραμέτρους**

οπότε περιεχόμενο έχει **<xsl:with-param>** - (θα το δούμε αργότερα)

Identity Transformation σημαντικό για xml -> xml

identity.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="@*|node()">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
        </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

αντιγράφει τον τρέχοντα κόμβο

για το περιεχόμενο του αντιγραμμένου τρέχοντος κόμβου αναδρομική κλήση

Ταιριάζει σε κάθε κόμβο, κάθε attribute

Σημαντική διαφοροποίηση συμβολισμού:

node() : κάθε είδους κόμβος (εκτός attribute node)

* : κάθε κόμβος στοιχείου (element node) MONON

@* : κάθε κόμβος ιδιότητας (attribute node)

Χρήση της Identity Transformation

Επιλεκτική επέμβαση με override κατά την αντιγραφή !

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="@*|node()"> <!-- Identity Template -->
         <xsl:copy>
              <xsl:apply-templates select="@*|node()"/>
         </xsl:copy>
   </xsl:template>
```

<!– Το παρακάτω πιο εξειδικευμένο template επικρατεί της αντιγραφής -->

```
<xsl:template match="@address">
      <!– do something in the place of the ‘address’ attribute →
      <!– (or do nothing: the ‘address’ attribute will be omitted) →
   </xsl:template>
```

```
</xsl:stylesheet>
```

xsl:copy-of select= "... " σε αντίθεση με xsl:copy

το **xsl:copy**
Αντιγράφει στην έξοδο τον **τρέχοντα κόμβο (όνομα και το namespace μόνον)** (**shallow copy**) - ΟΧΙ attributes και παιδιά του

το **xsl:copy-of select= "..... "**
Αντιγράφει στην έξοδο **ότι ορίζεται στο select attribute**
Αυτό μπορεί να είναι ένα
- ένα υποδένδρο,
- ένα node-set,
- εάν τίποτα από τα δύο παραπάνω, το περιεχόμενο του select μετατρέπεται και αντιγράφεται σαν string

Σε αντίθεση με το <xsl:copy>, κάθε κόμβος που αντιγράφεται, αντιγράφεται στην ολότητά του που περιλαμβάνει
namespace nodes, attribute nodes, και child nodes.

xsl:copy-of select= "... " σε αντίθεση με xsl:copy

το **<xsl:copy>**
… content …
</xsl:copy>

Ανοίγει και μπορεί να έχει περιεχόμενο

Σημασία:

Θέλομε να θέσομε νέο, πρόσθετο περιεχόμενο σε κάποιο σημείο:

Γράφομε το Identity Transformation και από κάτω

```
<xsl:template match="Xpath_of_el_to_obtain_additional_content">
        <xsl:copy> <!-- copy of element name only -->
            … new content here
            <xsl:apply-templates/><!-- copy rest of existing content -->
        </xsl:copy>
</xsl:template>
```

το **<xsl:copy-of select= "..... "/>** δεν ανοίγει και δεν μπορεί να έχει (νέο) περιεχόμενο !

node-set

κατασκεύασμα της XSLT, ακριβώς ότι λέει το όνομα του

Σε πάμπολλες περιπτώσεις

αποτέλεσμα μίας αναζήτησης θα είναι ένα σύνολο κόμβων.

Result Tree Fragment - RTF

το node-set έχει φτιαχθεί ακριβώς για να φιλοξενεί αυτό

- **αν το αποτέλεσμα είναι ένας κόμβος θα έχομε node-set με ένα στοιχείο**
- **αν το αποτέλεσμα είναι τίποτα θα έχομε κενό node-set**

το node-set αντικατεστάθη από το sequence στο XLST 2.0 (βλ. παρακάτω)

current()

Επιστρέφει ένα node-set με τον τρέχοντα (current) κόμβο σαν το μόνο του μέλος.
Σχεδόν πάντα ο τρέχων κόμβος (current) είναι ταυτόσημος με τον context node.

Ετσι τα παρακάτω δύο στοιχεί έχουν ίδια σημασία και ίδιο αποτέλεσμα:
<xsl:value-of select="current()"/>
<xsl:value-of select="."/>

Μέσα όμως σε ένα φίλτρο (predicate expression), current node και context node είναι συνήθως διαφορετικοί:

Η τελεία (.) δηλώνει τον τρέχοντα (current node) στο σημείο αυτό του (XPath)
(είδαμε ότι με . δηλώνεται η self axis)
με current() function δηλώνεται ο τρέχων κόμβος
πριν την αρχήν της επεξεργασίας της XPath expression από τον XSLT processor.

Template Modes

Απλό: Έστω ότι σε δύο templates το pattern XPath1 του match ταιριάζει στην ίδια XPath expression

```
<xsl:template match="XPath1"     mode="myMode">
        .....
</xsl:template>
```

… τα templates όμως έχουν διαφορετική τιμή για το **mode** attribute

```
<xsl:template match="XPath1" mode="otherMode">
        .....
</xsl:template>
```

Τότε για το

```
<xsl:apply-templates select="path_matched_by_XPath1" mode="otherMode"/>
```

θα εφαρμοσθεί **μόνον το δεύτερο template**

Named Templates

Φιλοσοφία του functional programming

Αντί

```
<xsl:template match="XPath1">
        .....
 </xsl:template>
```

‘καλούμενο’ από κάτι σαν

```
<xsl:apply-templates select=“path_matched_by_XPath1”/>
```

Το template έχει εδώ ένα όνομα (attribute name)

ξεφεύγει από το functional programming – σαν απλή κλίση μιας ‘procedure’

```
<xsl:template name="name_as_string">
        .....
 </xsl:template>
```

και καλείται σαν υπορουτίνα:

```
<xsl:call-template name=“name_as_string”/>
```

xsl:for-each select="..." σαν mini template

Πέραν από τα κανονικά templates που εφαρμόζονται μετά από **xsl:apply-templates** η **xsl:call-template** υπάρχει και άλλος τρόπος για απλές περιπτώσεις

```
<xsl:for-each select = "books/book">
        <!-- to be applied to every book -->.....
</xsl:for-each>
```

Συνήθως συνδυάζεται με **xsl:sort**

```
<xsl:for-each select = "books/book">
  <xsl:sort select= "@number" />
   <!-- to be applied to every book in order defined by
the value of its attribute number -->
  </xsl:sort>
</xsl:for-each>
```

… για αυτό λέμε ότι το **xsl:for-each** ορίζει ένα **mini template**

Χρήση παραμέτρων

```
<xsl:template match="XPath1">
<xsl:param name="param_no1" select="init_value1"/>
…
<xsl:param name="param_nok" select="init_valuek"/>
   … use these parameters referring to these
   … as $param_no1 … $param_nok …
</xsl:template>
```

Ορισμοί παραμέτρων

Καθορίζομε τις τιμές των παραμέτρων κατά την 'κλήση'

```
<xsl:apply-templates select="path_matched_by_XPath1">
    <xsl:with-param name="param_no3" select="value3"/>
    <xsl:with-param name="param_noj" select="valuej"/>
 </xsl: apply-templates>
```

Χρήση παραμέτρων

default values

Παρομοίως και για τα named templates

Μεταβλητές **(variables)** & Σταθερές (Constants) **προσοχή**

θα συνδεθούν συντόμως με τις παραμέτρους
– δεν διαφέρουν ουσιαστικά

Παραμέτρος (parameter)

– xsl:param ανήκει στο template

Κλήση /Εφαρμογή template

– επιλέγομε τιμή της παραμέτρου στο xsl:with-param και ετσι διαφοροποιούμε την συμπεριφορά του template

Μεταβλητή (variable)

– xsl:variable ανήκει στο stylesheet ή σε στοιχείο

και δεν διαμορφώνεται εξωτερικά (σαν σταθερά)

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

Δύο τρόποι ορισμού:

<xsl:variable name=“nameOfVariable”>contentOfVariable</xsl:variable>

το contentOfVariable μπορεί να είναι ένα ολόκληρο template

<xsl:variable name=“nameOfVariable” select=“XPathExpressionToValue”/>

Όπως κάθε μεταβλητή έχει όνομα (name) και τιμή

- στην πρώτη περίπτωση δίδομε την τιμή απ΄ ευθείας
- στην δεύτερη περίπτωση η τιμή ανευρίσκεται κάπου στο εγγρ. εισόδου και την αποκτάμε πηγαίνοντας εκεί με XPathExpression

Προσοχή: η variable είναι όμως πρακτικά **μία σταθερά (constant)**
ούτε υπάρχει άλλος τρόπος ορισμού ‘σταθεράς’

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

Αν ο ορισμός της μεταβλητής

με όνομα nameOfVariable και τιμή contentOfVariable

- ευρίσκεται σαν **παιδί του xsl:stylesheet element** (δηλ. έξω από κάθε template)
  κάνει την μεταβλητή αυτή ορατή σε όλο το stylesheet → **global variable**

- ευρίσκεται κάπου **μέσα σε ένα template**
  κάνει την μεταβλητή αυτή

  ορατή από εκεί και κάτω μέσα στο ίδιο μόνον template → **local variable**

  το ίδιο και μέσα σε ένα xsl:for-each **(mini template)**

Σε κάθε πέρασμα για το νέο κόμβο * η μεταβλητή ξαναγίνεται 'instantiated'

```
<xsl:for-each select="*">
    <xsl:variable name="curField" select="name()"/>
        …
</xsl:for-each>
```

Όπως πάντα: OXI state, OXI side effects

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

Αναφορά

στην μεταβλητή με όνομα nameOfVariable

Οπουδήποτε γράφομε **$nameOfVariable** μέσα στην περιοχή όπου η μεταβλητή (με όνομα nameOfVariable) είναι ορατή

η έκφραση αυτή αντικαθίσταται με την **τιμή** της μεταβλητής
(όπως έχει καθορισθεί κατά τον ορισμό της)

Έξω όμως από την περιοχή ορισμού η μεταβλητή είναι άχρηστη (δεν αναγνωρίζεται καν)

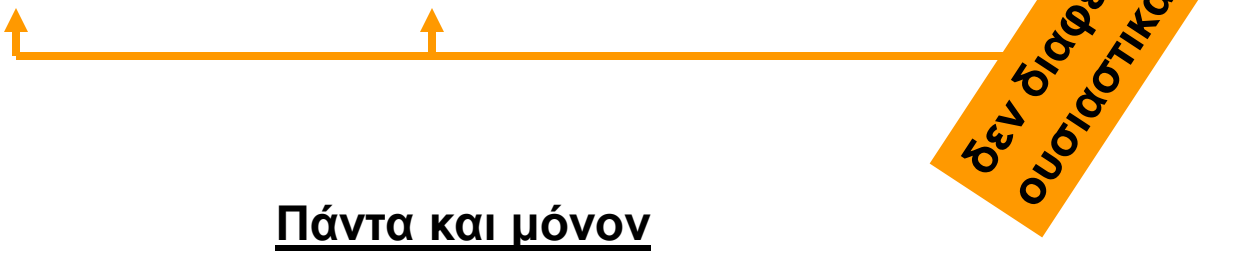
Επομένως ΟΧΙ μηχανισμός περάσματος τιμών (state) από το ένα template στο άλλο !!!!!!!!!!!

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

‘η variable είναι πρακτικά **μία σταθερά (constant)’**

και τότε πως φτιάχνομε, π.χ., έναν δείκτη για μια iteration ;

ΟΧΙ με **variable** αλλά με **parameter**



Πάντα και μόνον

με αναδρομική κλήση ενός παραμετροποιημένου template (iterative call)

Υπόδειγμα έπεται ! Σελ 78

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

Είδαμε την χρησιμότητα της ‘parameter’

στην εφαρμογή / κλήση παραμετροποιημένων templates

ΑΛΛΑ

‘η variable είναι πρακτικά **μία σταθερά (constant)’**

τελικά τι χρειάζεται, πώς χρησιμοποιείται ;;

Μεταβλητές **(Variables)** & Σταθερές (Constants) **προσοχή**

‘η variable είναι πρακτικά **μία σταθερά (constant)’**

τελικά τι χρειάζεται, πώς χρησιμοποιείται ;;

π.χ. για να αποθηκεύσομε το 3.14... σαν pi και να αναφερόμαστε σε αυτό με $pi

Παράμετροι μέσω εντολής:

java org.apache.xalan.xslt.Process **-IN** generalXML.xml **-XSL** firstXSLT.xsl **- OUT** res.txt

δίδονται με

-PARAM name expression

Ξεκαθάρισε αν το - PARAM σαν xalan switch εμπίπτει στις παραμέτρους ή στις μεταβλητές ;;;

Έλεγχος Ροής
- δύο τρόποι -

```
<xsl:if test="booleanExpression">
    <!– do something here →
</xsl:if>
```

Παράδειγμα booleanExpression
(τιμή του attribute test)

test="@color='red'"

αληθής συνθήκη:

αν το attribute color
έχει την τιμή red

```
<xsl:choose>
    <xsl:when test="booleanExpression1">
        <!– do something here →
    </xsl:when>
    <xsl:when test="booleanExpression2">
        <!– do something here →
    </xsl:when>
    <xsl:otherwise>
        <!– do something here →
    </xsl:otherwise>
</xsl:choose>
```

το **xsl:choose** περιτύλιγμα πολλών **xsl:when**

και πιθανώς **ενός μόνον**

xsl:otherwise **(πάντα τελευταίο)**

Προσοχή: Escaping

στις συγκρίσεις

αντί για < γράφομε < (less than)

αντί για > γράφομε > (grater than)

αλλιώς μπέρδεμα με τα tags των elements

Γενικά Attribute **value** με string in double quotes

Ενώ ειδικά για Attribute **select** με string in single quotes μέσα σε double quotes

```
<xsl:attribute name="firstName"><xsl:value-of select ="'John'"/></xsl:attribute>
```

αλλά

ανάμεσα στα " " πρέπει να υπάρχει κάτι που αποτιμάται σαν string, δηλ. 'John' ή το string το αποθηκευμένο στην $firstNameVar

```
<xsl:attribute name="firstName"><xsl:value-of select ="$firstNameVar"/></xsl:attribute>
```

ή

```
<xsl:attribute ourTranformer="firstName">
<xsl:value-of select="system-property('xsl:vendor-url')"/>
</xsl:attribute>
```

A XPath function
returning a string!

XSLT – Ανακεφαλαίωση, Γενικές Παρατηρήσεις

η **XSLT** επηρεασμένη από το '*functional programming* ' και γλώσσες σαν την Lisp

templates

functions

(χωρίς side effects)

κοινά σημεία :

- **immutable variables** (δεν υπάρχουν 'μεταβλητές' παρά μόνον σταθερές)
- **η εφαρμογή ενός template (function) δεν επηρεάζει την εφαρμογή άλλου (function)**
- **προγραμματιστικά το looping αντικαθίσταται από recursion**

XPath

‘για την περιδιάβαση μέσα σε έγγραφο XML’

XPath

Πιο συγκεκριμένα τρεις χρήσεις:

- για το προσδιορισμό και εξαγωγή δεδομένων μέσα από ένα έγγραφο (π.χ. σαν όρισμα του select)

- σαν ‘pattern language’ για να διαπιστωθεί ποίο template ταιριάζει (π.χ. σαν όρισμα του match)

Αλλά και

- για ‘math, string manipulations’ με σχετικές ενσωματωμένες συναρτήσεις

XPath – Γενικές παρατηρήσεις

Γενίκευση του συμβολισμού **‘μονοπατιών’ σε σύστημα αρχείων**

Εξαιρετικά δυνατό

Επεκτάσεις

- **και στην λεπτομέρεια (στο έγγραφο .xml και μετά σε μέρη του κειμένου του (text))**
- και στο περιβάλλον (στην database και μετά στο έγγραφο .xml)

Βασική ιδέα ‘οι άξονες’ **axes** – υπάρχουν 13 !

Δίδουν τις ‘κατευθύνσεις μετακίνησης’ μέσα στο έγγραφο

… αλλά τις ονόμασαν ‘άξονες’ !!;;;

XPath – Γενικές παρατηρήσεις

Όλα είναι κόμβοι !!
7 διαφορετικά είδη

1. η ρίζα του εγγράφου
2. κόμβος στοιχείου (στοιχείο)
3. κόμβος κειμένου (κείμενο)
4. κόμβος ιδιότητας
5. κόμβος PI
6. κόμβος – σχόλιο
7. κόμβος namespace

1. document root
2. element node (element)
3. text node (text)
4. attribute node (attribute)
5. PI node
6. comment node
7. namespace node

Θα ασχοληθούμε με τα παραπάνω και με σύνολα αυτών (node sets)

Axes (Άξονες ;;) του XPath 1/2

Axis	Σύμβολο	Οδηγεί προς (ΤΚ ο τρέχων κόμβος - current node)
child	τίποτα	τα (άμεσα) παιδιά του ΤΚ - **ΕΙΝΑΙ ΤΟ Default Axis**
descendant	**//**	τα παιδιά, παιδιά των παιδιών, κλπ του ΤΚ – ΟΧΙ attributes
parent	**..**	τον γονέα του ΤΚ
ancestor		τους προγόνους του ΤΚ μέχρι και την ρίζα
following-sibling		τα επόμενα αδέλφια του ΤΚ (άδειο αν ο ΤΚ είναι attribute)
preceding-sibling		τα προηγούμενα αδέλφια του ΤΚ (άδειο αν ο ΤΚ είναι attribute)
following		τους κόμβους που στο έγγραφο έπονται του ΤΚ, ΜΗ περιλαμβανομένων των **descendant** του ΤΚ (άδειο αν ο ΤΚ είναι attribute)
preceding		τους κόμβους που στο έγγραφο προηγούνται του ΤΚ, ΜΗ περιλαμβανομένων των **ancestor** του ΤΚ (άδειο αν ο ΤΚ είναι attribute)
self	**.**	τον ίδιον τον ΤΚ

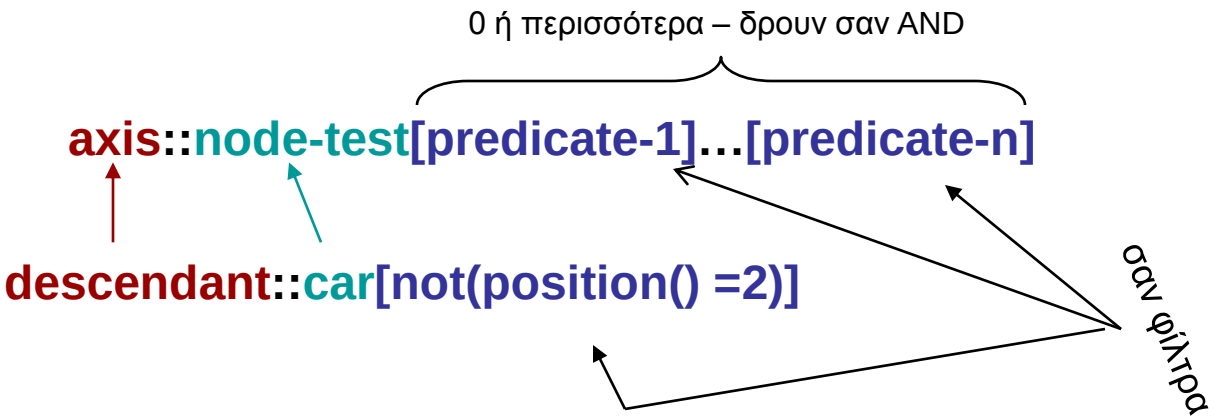
Οι ancestor, descendant, following, preceding και self axes χωρίζουν το έγγραφο (με εξαίρεση των attributes) σε μη επικαλυπτόμενα τμήματα, η ένωση των οποίων αποτελεί το πλήρες έγγραφο.

Axes (Άξονες ;;) του XPath 2/2

Axis	Σύμβολο	Οδηγεί προς (ΤΚ ο τρέχων κόμβος - current node)
attribute	@	τα attributes του ΤΚ (άδειο αν ο ΤΚ δεν είναι κόμβος στοιχείου – element node)
namespace		τους namespace κόμβους του ΤΚ (άδειο αν ο ΤΚ δεν είναι κόμβος στοιχείου – element node)
descendant-or-self		την ένωση των descendant και self αξόνων
ancestor-or-self		την ένωση των ancestor και self αξόνων

Εκφράσεις XPath Expressions

Από **axis** σε **location step**



Ευρισκόμενοι στο cars ή πιο πάνω, μας δίνει (σαν **element node set**) όλα τα στοιχεία car, εκτός του δεύτερου

Εκφράσεις XPath Expressions

Από **location step** σε **location path**

Location path **σχετικό (relative) ως προς τον κόμβο αναφοράς (context node)**

locationStep_1/ locationStep_2 / locationStep_n

Απόλυτο (**absolute**) location path (σχετικό ως προς την ρίζα)

/locationStep_1/ locationStep_2 / locationStep_n

Παράδειγμα

όλα

descendant::**car**[**not**(**position**()=2)]/**attribute** ::*

locationStep_1 locationStep_2

Ευρισκόμενοι στο cars ή πιο πάνω, μας δίνει (σαν **attribute node set**)
όλα τα attributes όλων των στοιχεία car, εκτός του δεύτερου στοιχείου car

Συντομογραφία Εκφράσεων XPath
Abbreviated XPath Syntax

descendant::car[not(position()=2)]/attribute::*

βλ. ‘Σύμβολο’ στον πίνακα αξόνων

:: δεν γράφονται

descendant γράφεται σαν //

attribute γράφεται σαν @

child ΔΕΝ γράφεται

συντομεύεται σε

//car[not(2)]/@*

επίσης

παραμένει

child::cars/child::car **συντομεύεται σε** **cars/car**

διότι το child axis συντομεύεται σε ‘τίποτα’

Default Axis

βλ. ‘Σύμβολο’ στον πίνακα αξόνων

Συντομογραφία Εκφράσεων XPath
Abbreviated XPath Syntax

parent::x **1 location step**

δίδει το πολύ έναν κόμβο,

δηλ. τον πατέρα (αυτός είναι πάντα το πολύ ένας) του context node, ΑΝ αυτός λέγεται x -- Αλλιώς δίνει τον empty node

../x **πήγαινε στον πατέρα και μετά στα παιδιά του που λέγονται x** **2 location steps**

Δίδει όλα τα sibling του context node που λέγονται x, μαζί με το ίδιο τον context node, αν λέγεται x -- Πιθανώς τον empty node

***[@x]** **1 location step με φίλτρο**

Δίδει όλα τα παιδιά του context node που έχουν attribute που λέγεται x-- Πιθανώς τον empty node

***[x]** **1 location step με φίλτρο**

Δίδει όλα τα παιδιά του context node που έχουν κάποιο παιδί που λέγεται x-- Πιθανώς τον empty node

Συντομογραφία Εκφράσεων XPath
Abbreviated XPath Syntax

Δίδει όλα τα παιδιά του context node που έχουν κάποιο παιδί που λέγεται x-- Πιθανώς τον empty node

***[x]** **1 location step με φίλτρο**

συντομογραφία του

***[boolean(child::x)]**

boolean(*nodeset*)
Το τίποτα είναι το default του **child::**

boolean(*nodeset*)
Δίδει true μονον αν το *nodeset* δεν είναι κενό

Η σειρά εφαρμογής των φίλτρων έχει σημασία !

***[name()='x'][1]**

πάρε τα παιδιά,
κράτησε αυτά με το όνομα x
κράτησε το πρώτο

*** [1][name()='x']**

πάρε τα παιδιά,
κράτησε το πρώτο
κράτησε αυτά με το όνομα x

αν έχομε τα παιδιά με την σειρά y,x,z, παίρνομε

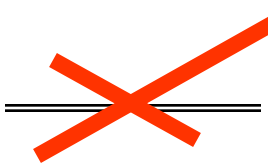
x

άδειο node-set

... και προσοχή

πάρε τα παιδιά,
κράτησε αυτά που έχουν παιδί με το όνομα x
κράτησε το πρώτο

***[name()='x'][1]**



***[x][1]**

current() σε σύγκριση με την τελεία (self axis)

Το current() επιστρέφει ένα node-set με τον τρέχοντα (current) κόμβο σαν το μόνο του μέλος.

Σχεδόν πάντα ο τρέχων κόμβος (current) είναι ταυτόσημος με τον context node.

Έτσι τα παρακάτω δύο στοιχεία έχουν ίδια σημασία και ίδιο αποτέλεσμα:

```
<xsl:value-of select="current()"/>
<xsl:value-of select="."/>
```

Μέσα όμως σε ένα φίλτρο (predicate expression), current node και context node είναι συνήθως διαφορετικοί:

Η τελεία (.) δηλώνει τον τρέχοντα (current node) στο σημείο αυτό του (XPath)

(είδαμε ότι με . δηλώνεται η self axis)

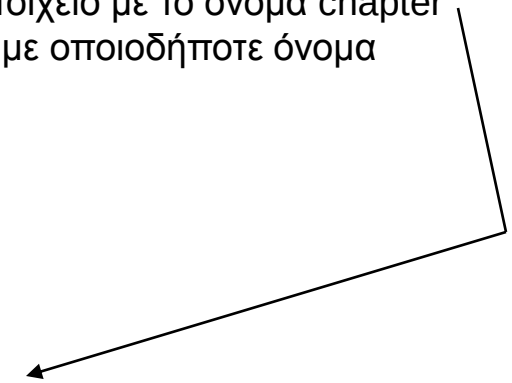
ενώ με την current() function δηλώνεται ο τρέχων κόμβος
πριν την αρχήν της επεξεργασίας της XPath expression από τον XSLT processor

Συντομογραφία Εκφράσεων XPath
Abbreviated XPath Syntax

δοκιμές με myBooks.xml

Επιλογή στοιχείων με βάση την σειρά - αρίθμηση

Ένα predicate περιέχον μέσα σε [] ένα φυσικό αριθμό κ σημαίνει το κ–στο στοιχείο (αρίθμηση αρχίζει από 1) π.χ.
chapter[1] επιλέγει το 1ο στοιχείο με το όνομα chapter
***[2]** επιλέγει το 2ο στοιχείο με οποιοδήποτε όνομα



Αυτό αποτελεί συντομογραφία του

chapter[position()=2]

όπου το position() είναι μία συνάρτηση XPath – θα δούμε αργότερα

chapter[position()=1 OR position()=3] επιλέγει το 1ο και 3ο στοιχείο με όνομα chapter

(Δραστική) Συντομογραφία Εκφράσεων XPath
Abbreviated XPath Syntax

Κατευθείαν στον στόχο με //

(σύμβολο συντομογραφίας του **descendant axis**)

Με // η XPath αντιλαμβάνεται μηδέν ή απροσδιόριστα βήματα εντοπισμού.

Αυτό σημαίνει ότι με /books//line επιλέγομε όλα το στοιχεί με όνομα line τα οποία ευρίσκονται κάτω από το books. Συγκεκριμένα στο myBooks.xml επιλέγομε όλα τα line και των chapterHeader και των chapterText Το ίδιο πετυχαίνομε και με το ακόμα δραστικότερο //line. (linesFromMyBooks.xsl)

Αλλά .... μεγάλο υπολογιστικό κόστος

Η XPath πρέπει να ψάξει από το αρχικό σημείο σε όλο το βάθος και όλο το εύρος διότι δεν καθοδηγείται από συγκεκριμένη XPathExpression προς τον στόχο!

Άρα μόνον λύσις ανάγκης

(αναγκαστική) χρήση μόνον όταν ξέρομε μόνον το όνομα και τίποτα από την δομή)

Κατευθείαν στον στόχο με //

```
<?xml version="1.0"?>
<!-- java org.apache.xalan.xslt.Process -IN myBooks.xml -XSL linesFromMyBooks.xsl -OUT
linesExtracted.txt-->
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/"  >
    Print all lines to be found below of element books:
    <xsl:apply-templates select="/books//line"  /> <!-- change to "//line" -->
</xsl:template>

<xsl:template match="line"  ><!-- change to "chapterHeader/line" -->
    LINE FOUND: <xsl:value-of select="."  />
</xsl:template>

</xsl:stylesheet>
```

Άσκηση (βλ. σχόλια) και για

- **pattern matching**
- **προτίμηση πλέον εξειδικευμένου template**

το myBooks.xml διαθέσιμο

string value ενός στοιχείου

string value of an element node

Είναι η συρραφή όλων των text του προκειμένου κόμβου και των υποκειμένων του (descendant nodes) με την σειρά που απαντώνται στο έγγραφο

- **Δίδεται από το π.χ. <xsl:value-of select="sonnet"/>**

  <xsl:value-of select="/"/> δίνει όλο το κείμενο που ευρίσκεται μέσα στο έγγραφο

- **Επιστρέφεται από την string() function, π.χ. string(sonnet)**

όχι ξεκάρφωτο, αλλά πάντα μέσα σε XPath expression !!

αν είναι πολλοί (node-set) τότε παίρνεται ο πρώτος κόμβος

Πως δημιουργείται το node-set που επιστρέφεται από το XPath

```
<?xml version="1.0"?>
<!-- java org.apache.xalan.xslt.Process -IN myBooks.xml -XSL sectionsFromMyBooks.xsl -OUT
linesExtracted.txt-->
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/"  >
    Select accoding to order:
    <xsl:apply-templates select="/books/book/chapters/chapter[@num='2']/chapterText/line[2]"  />
    <xsl:apply-templates select="/*/*/*/*[@num='2']/chapterText/line[1]"  />
<!--     DOES NOT WORK ???
    Select SEVERAL accoding to order: 1OR 3
    <xsl:apply-templates select="/*/*/*/*[@num='1']/chapterText/line[1|3]"/>
 -->
     Why do we get a  nodes set with multiple nodes !!!??? :
    <xsl:apply-templates select="//line[2]"/>
</xsl:template>

<xsl:template match="line" >
    LINE FOUND: <xsl:value-of select="."  />
</xsl:template>
```

Άσκηση (βλ. σχόλια) και για

- **χρήση φίλτρων (predicates μέσα σε [])**
- **τρόπο δημιουργίας του node set**

το myBooks.xml διαθέσιμο

Εκφράσεις XPath Expressions

παραδείγματα, τεχνάσματα

Οι παρακάτω εκφράσεις επιστρέφουν ένα κενό (empty) node set

/.. (η ρίζα δεν έχει πατέρα) /@* (η ρίζα δεν έχει attributes)

Το following, preceding οδηγεί στα επόμενα, προηγούμενα αδέλφια (ίδιο επίπεδο) -

αν θέλομε όλους τους κόμβους (στοιχεία) στο ίδιο επίπεδο όπως ο ΤΚ, γράφομε: **./***

(πίσω στο πατέρα και μετά σε όλα τα στοιχεία παιδιά)

Τα επόμενα αδέλφια του ΤΚ, περιλαμβανομένων και των παιδιών τους που δεν ονομάζονται azs

following-sibling::subparagraphs[not(name()=azs)]

Ένα attribute (είναι και αυτό κόμβος!) έχει σαν ancestor (πατέρα) τον κόμβο στοιχείου στον οποίο ανήκει, αλλά ΔΕΝ είναι child του κόμβου αυτού

Προσοχή: //someElementName είναι απολύτως σωστό,

αλλά βαρύ ως προς την απόδοση (πρέπει να ψαχθεί ΟΛΟ το δένδρο !)

Μία έκφραση XPath Expression

το είδαμε !

επιστρέφει:

- **ένα node-set (μία αδιάτακτη συλλογή μη επαναλαμβανόμενων κόμβων (collection of nodes without duplicates)**

(πιθανόν να είναι κενό, π.χ. η έκφραση /..)

- μία τιμή boolean (true or false)
- έναν αριθμό number (floating-point number)
- ένα string (sequence of UCS characters)

βλ. συνέχεια (νέους ορισμούς (XSLT, XPath funstions) και κατόπιν την χρήσιμότητά τους

Χρήση συναρτήσεων XSLT και XPath

Μέσα σε predicates (φίλτρα – τα […] των XPathExpressions)

chapter[last()]
Επιλέγει το τελευταίο στοιχείο από τα στοιχεία chapter (το last() επιστρέφει έναν αριθμό τον οποίον δεν ξέρομε). Αν θέλαμε το πρώτο των στοιχείων chapter θα γράφαμε απλώς **chapter[1]**, συντομογραφία του πιό πλήρους **chapter[position()=1]**

chapter[position() mod 2 = 0]
Επιλέγει όλα τα στοιχεία chapter με άρτιο αριθμό θέσης (2ο, 4ο, 6ο , κλπ)

count(/book/chapter[@status="corrected"])
Επιστρέφει το πλήθος των στοιχείων chapter που

- είναι παιδιά του document element book (επειδή η XPathExpression αρχίζει απόλυτα με /) **και**
- έχουν τιμή του status attribute '*corrected*'

Εδώ μία XPathExpression χρησιμοποιείται σαν μεταβλητή εισόδου μίας συνάρτησης XPath, εν προκειμένω της **count()**

το myBooks.xml διαθέσιμο

Συναρτήσεις XSLT Functions (επιστροφές όχι πλέον μόνον node-set)

Όνομα	Περιγραφή
current()	Επιστρέφει τον τρέχοντα κόμβο (current node)
document()	Για πρόσβαση των κόμβων ενός εξωτερικού εγγράφου XML
element-available()	Ελέγχει αν το συγκεκριμένο στοιχείο (xsl element) υποστηρίζεται από τον εν χρήσει XSLT processor
format-number()	Μετατρέπει έναν αριθμό (number) σε string
function-available()	Ελέγχει αν το συγκεκριμένο στοιχείο (xsl function) υποστηρίζεται από τον εν χρήσει XSLT processor
generate-id()	Επιστρέφει μία τιμή string η οποία μονοσήμαντα χαρακτηρίζει τον τρέχοντα κόμβο
key()	Επιστρέφει ένα node-set χρησιμοποιώντας το index, όπως ορίστηκε σε ένα <xsl:key> element
system-property()	Επιστρέφει την τιμή value των system properties
unparsed-entity-uri()	Επιστρέφει το URI μίας unparsed entity

ιδιαίτερη μεταχείριση αργότερα

Συναρτήσεις XPath Functions

Όνομα	Περιγραφή	Σύνταξη / Παράδειγμα
count()	Επιστρέφει το πλήθος των κόμβων ενός nodeset	number=count(node-set)
id()	Επιλέγει elements μέσω του μοναδικού ID τους	node-set=id(value)
last()	Επιστρέφει το αριθμό της θέσης του τελευταίου κόμβου (last node) στην υπό επεξεργασία node list	number=last()
local-name()	Επιστρέφει το **τοπικό όνομα ενός κόμβου**. Το όνομα ενός κόμβου αποτελείται από ένα prefix, ένα colon και το τοπικό όνομα (local name)	string=local-name(node)
name()	Επιστρέφει το **όνομα ενός κόμβου**	string=name(node)
namespace-uri()	Επιστρέφει το namespace URI (OXI το prefix)	uri=namespace-uri(node)
position()	Επιστρέφει το αριθμό της θέσης μέσα στην the node list του υπό επεξεργασία κόμβου	number=position()

XPath Functions: Συναρτήσεις επιστρέφουσες string 1/2

Όνομα	Περιγραφή	Σύνταξη / Παράδειγμα
concat()	Παίρνει δύο ή περισσότερα strings τα οποία συρράπτει σε ένα	string=concat(val1, val2, ..) concat('The',' ','XML') επιστρέφει 'The XML'
contains()	Επιστρέφει true εάν το δεύτερο string περιέχεται στο πρώτο string, αλλιώς επιστρέφει false	bool=contains(val,substr) contains('XML','X') επιστρέφει true
normalize-space()	Διαγράφει κενά από την αρχή και το τέλος του string	string=normalize-space(string) normalize-space(' The XML ') επιστρέφει 'The XML'
starts-with()	Επιστρέφει true εάν πρώτο string**αρχίζει με το** δεύτερο string , αλλιώς επιστρέφει false	bool=starts-with(string,substr) starts-with('XML','X') επιστρέφει true

XPath Functions: Συναρτήσεις επιστρέφουσες string 2/2

Όνομα	Περιγραφή	Σύνταξη / Παράδειγμα
string()	Επιστρέφει την παράμετρο εισόδου σαν string	**string(value)** string(314) επιστρέφει '314'
string-length()	Επιστρέφει τον αριθμό χαρακτήρων του string	**number=string-length(string)** string-length('Beatles') επιστρέφει 7
substring()	Επιστρέφει μέρος του string	**string=substring(string,start,length)** substring('Beatles',1,4) επιστρέφει 'Beat'
substring-after()	Επιστρέφει το μέρος του string που εμφανίζεται **μετά** το διδόμενο substring	**string=substring-after(string,substr)** substring-after('12/10','/') επιστρέφει '10'
substring-before()	Επιστρέφει το μέρος του string που εμφανίζεται **πριν** το διδόμενο substring	**string=substring-before(string,substr)** substring-before('12/10','/') επιστρέφει '12'
translate()	Αντικαθιστά όλες τις εμφανίσεις του string1 με το string2	**string=translate(value,string1,string2)** translate('12:30',':','!') επιστρέφει '12!30'

XPath Functions: Συναρτήσεις επιστρέφουσες number

Όνομα	Περιγραφή	Σύνταξη / Παράδειγμα
ceiling()	Επιστρέφει το μικρότερο integer, που δεν είναι μικρότερο από τον διδόμενο αριθμό	**number=ceiling(number)** ceiling(3.14) επιστρέφει 4
floor()	Επιστρέφει το μεγαλύτερο integer, που δεν είναι μεγαλύτερο από τον διδόμενο αριθμό	**number=floor(number)** floor(3.14) επιστρέφει 3
number()	Μετατρέπει το διδόμενο value σε number (στην boolean false επιστρέφει 0, στην true επιστρέφει 1)	**number=number(value)** number('100') επιστρέφει 100
round()	Στρογγυλοποιεί το διδόμενο number στον πλησιέστερο ακέραιο	**integer=round(number)** round(3.14) επιστρέφει 3
sum()	Επιστρέφει το άθροισμα των αριθμητικών τιμών ενός συνόλου κόμβων (nodeset)	**number=sum(nodeset)** sum(/cd/price)

τα παραπάνω φυσικά συνδυαζόμενα με τα σύμβολα **+**, **-**, ***, **/**, **div**, **mod**

XPath Functions: Συναρτήσεις επιστρέφουσες boolean

Όνομα	Περιγραφή	Σύνταξη / Παράδειγμα
boolean()	Μετατρέπει την τιμή του διδόμενου value σε boolean επιστρέφοντας true ή false	**bool=boolean(value)**
false()	Επιστρέφει false	**false()** number(false()) επιστρέφει 0
lang()	Επιστρέφει true εάν το language συμπίπτει με το language του στοιχείου xsl:lang element, αλλιώς επιστρέφει false	**bool=lang(language)**
not()	Επιστρέφει true εάν **το** condition είναι false, και false εάν **το** condition είναι is true	**bool=not(condition)** not(false())
true()	Επιστρέφει true	**true()** number(true()) επιστρέφει 1

τα παραπάνω φυσικά συνδυαζόμενα με τα σύμβολα **and**, **or**, **=**, **!=**

Έως εδώ XPath 1.0 ΤΩΡΑ **XPath 2.0**

XPath 1.0 και XPath 2.0 δεν είναι συμβατά (ούτε backward ούτε forward)

Κάθε XPath 1.0 expression επιστρέφει node set

Κάθε XPath 2.0 expression επιστρέφει sequence

κάθε sequence είναι

- ordered,

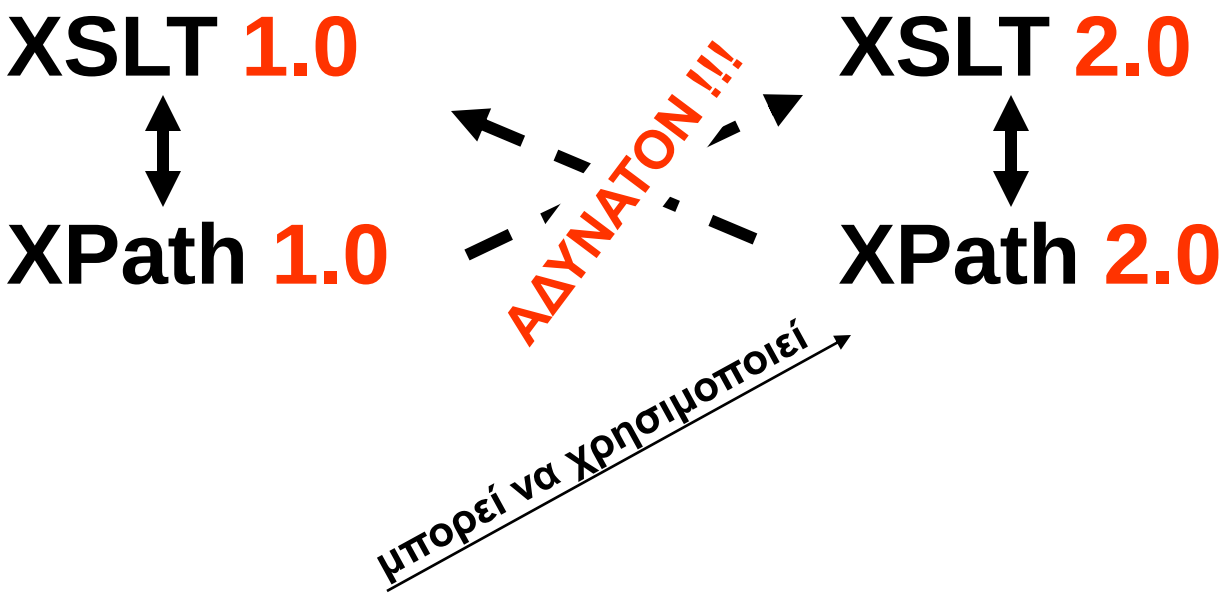
- indexed (στοιχεία της αριθμημένα από το 1 – όχι το 0)

η sequence, πέραν του ότι είναι το αποτέλεσμα μίας XPath 2.0 expression, μπορεί

- να κατασκευασθεί απ΄ ευθείας (εκ του μηδενός),

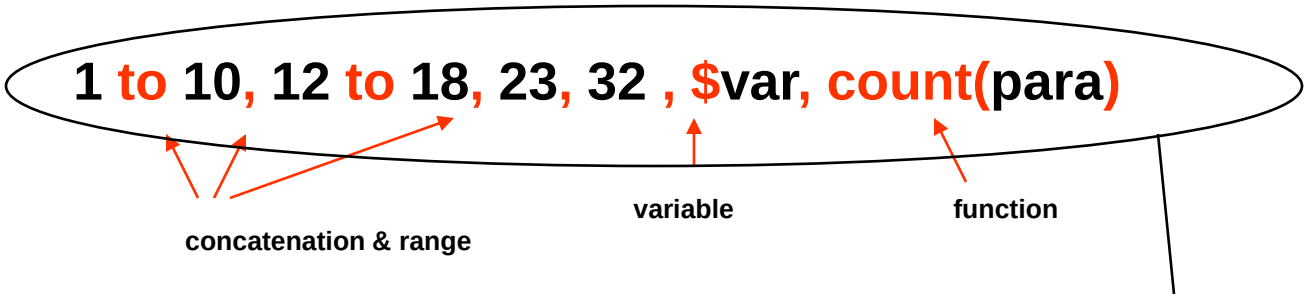
- να επεξεργασθεί

XSLT 2.0 & XPath 2.0



Κατασκευή sequence

()	κενή sequence
1	sequence αποτελούμενη από ένα στοιχείο, το ‘1’
X, Y, Z	όλα τα παιδιά με όνομα X του context node, ακολουθούμενους από τα Y, Z



Επεξεργασία sequence

insert-before($seq1,3, $seq2)

θέσε την **seq2** πριν το **3ο** στοιχείο της **seq1**

seq1[not(position()=1,3,4)]

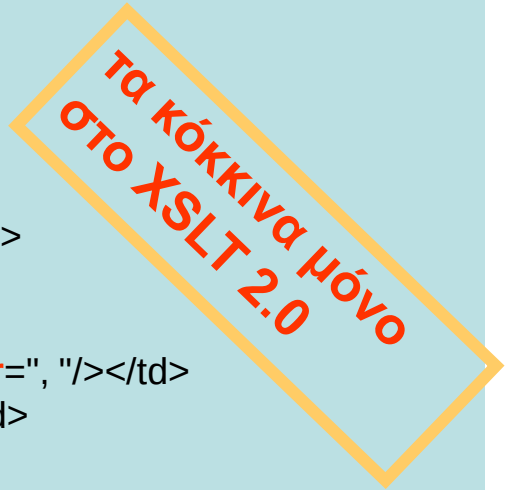
αφαίρεσε το 1ο, 3ο και 4ο στοιχείο της **seq1**

XPath 2.0

Με την sequence μπορούμε να ομαδοποιούμε (grouping)

- πολύ δύσκολο στο XSLT 1.0

```
<?xml version="1.0"?>
<!-- java net.sf.saxon.Transform -s:cities.xml -xsl:grouping.xsl -o:groupedCities.html-->
<xsl:stylesheet version="2.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
   <html>
     <head><title>Cities in Europe</title></head>
       <body background="Money.jpg">
       <h1>Cities grouped by country</h1>
       <table border="2">
            <xsl:for-each-group select="cities/city" group-by="@country">
            <tr>
              <td><xsl:value-of select="@country"/></td>
              <td><xsl:value-of select="current-group()/@name" separator=", "/></td>
              <td><xsl:value-of select="sum(current-group()/@pop)"/></td>
           </tr>
           </xsl:for-each-group>
         </table>
       </body>
    </html>
   </xsl:template>
</xsl:stylesheet>
```



Συγκερασμός templates 1/3

Combining templates

Σκοπός:

Τελικό stylesheet με πολλά έτοιμα, αλλού ευρεθέντα templates ... και μερικά δικά μας

Πολύ σημαντικό λόγω

- **επαναχρησιμοποίησης ετοίμων (εξωτερικών) templates**
- **δυνατότητος συνδυασμού απλών σε κάτι σύνθετο**

  (βάση του software engineering, όπως οπουδήποτε αλλού)
- **εν τέλει αύξησης της παραγωγικότητας**

… ιδίως όταν κάθε template διαμορφώνει την συμπεριφορά του κατά τις επιθυμίες μας με χρήση παραμέτρων

Συγκερασμός templates 2/3

xsl:include href="otherXSLT.xslt" τιμή του href οποιοδήποτε uri

• Το στοιχείο xsl:include πάντα παιδί του xsl:stylesheet, δηλ. sibling όλων των xsl:template του stylesheet που το εισάγει.

• Αποτέλεσμα το otherXSLT.xslt να συμπεριλαμβάνεται στο δικό μας stylesheet, δηλ.

Αποτέλεσμα το ίδιο σαν στην θέση του xsl:include να γράφαμε όλα τα templates του εισαγόμενου stylesheet

Μπορούμε να έχομε πολλά στοιχεία xsl:include, καθένα εκ των οποίων μπορεί να περιλαμβάνει δικά του xsl:include

Προσοχή σε πιθανή σύγκρουση κανόνων εφαρμογής – σφάλμα

xsl:import href="someXSLT.xslt" τιμή του href οποιοδήποτε uri

• Το στοιχείο **xsl:import** πάντα **πρώτο** παιδί του xsl:stylesheet

• Σχεδόν το ίδιο – Αλλά, σε περίπτωση σύγκρουσης κανόνων εφαρμογής, ένα template του εισάγοντος stylesheet υπερισχύει του template του εισαγόμενου stylesheet – το σφάλμα αποφεύγεται

Συγκερασμός templates 3/3

Συνήθης χρήση xsl:include ή xsl:import

Ένα template που φτιάχνει μία ωραία, κοινή επικεφαλίδα που ταιριάζει π.χ. στο στοιχείο pageHeader (match="somePathTo_ pageHeader")

Δεν το ξαναφτιάχνομε, αλλά το βρίσκομε έτοιμο σαν niceHeader.xslt.

Αλλά:

Αν έχομε γράψει και εμείς <xsl:template match="somePathTo_pageHeader">…, τότε

- το <xsl:include href="niceHeader.xslt"/> θα δώσει λάθος
- το <xsl:import href="niceHeader.xslt"/> θα αγνοηθεί

Για debug:

Αλλάζοντας όλα τα xsl:import σε xsl:include ευρίσκομε πιθανές ασυνέπειες

το **<xsl:apply-imports>**

εξαναγκάζει την εισαγωγή εξωτερικών templates

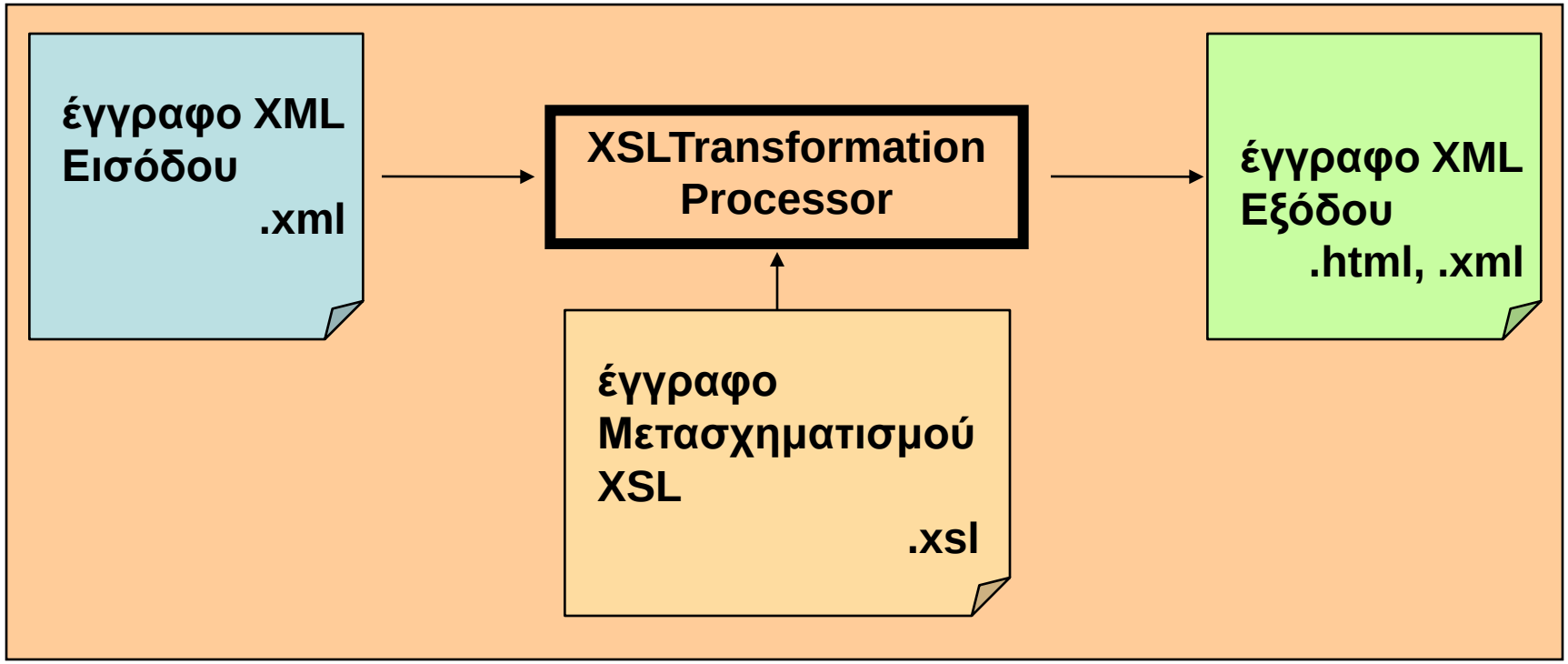
```
<xsl:template match="message">
   ....
  <xsl:apply-imports/>
  ....
</xsl:template>
```

στο σημείο αυτό θα εισαχθούν όλα τα εξωτερικά templates ακόμη ένα πού έχει match="message"

(δεν θα σκιασθεί από το)

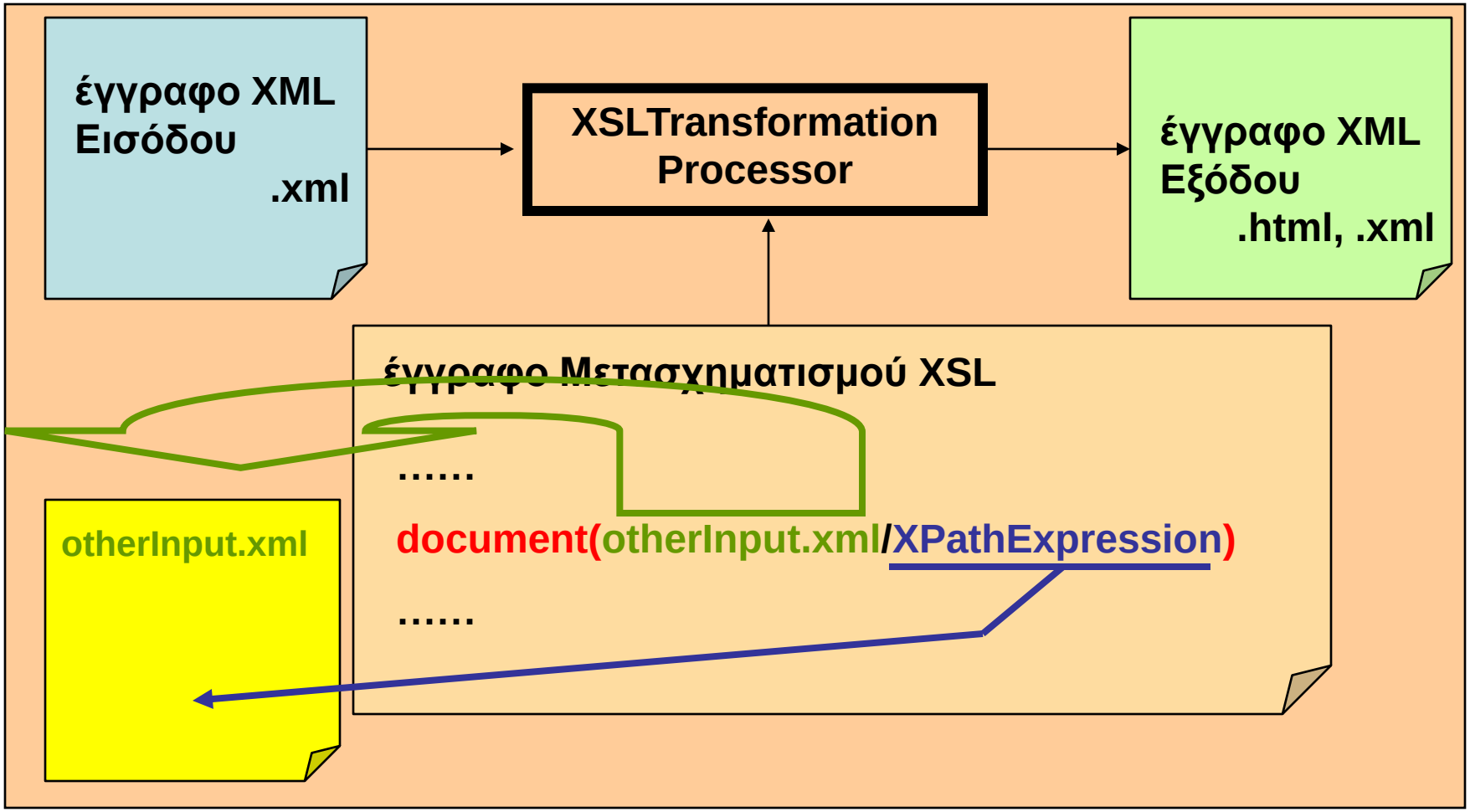
document() (η συνάρτηση XSLT που είχαμε αφήσει σαν εκκρεμότητα)

Η Γενική Αρχή που ακολουθήσαμε:



ΔΕΝ ΜΑΣ ΑΡΚΕΙ – Αλλά με την document() μπορούμε να έχομε σαν είσοδο περισσότερα του ενός έγγραφα .xml !!!

Γενική Ιδέα / Χρήση του document()



Το document παίρνει σαν argument

ένα άλλο κείμενο μαζί με ένα XPathExpression μέσα σε αυτό

Με document() μπορούμε να εξάγομε πληροφορία από πολλά έγγραφα ‘εισόδου’

πολλά έγγραφα ‘εξόδου’ ;

Μόνον με extensions στο XSLT 1.0

Δυνατόν στο XSLT 2.0 (saxon)

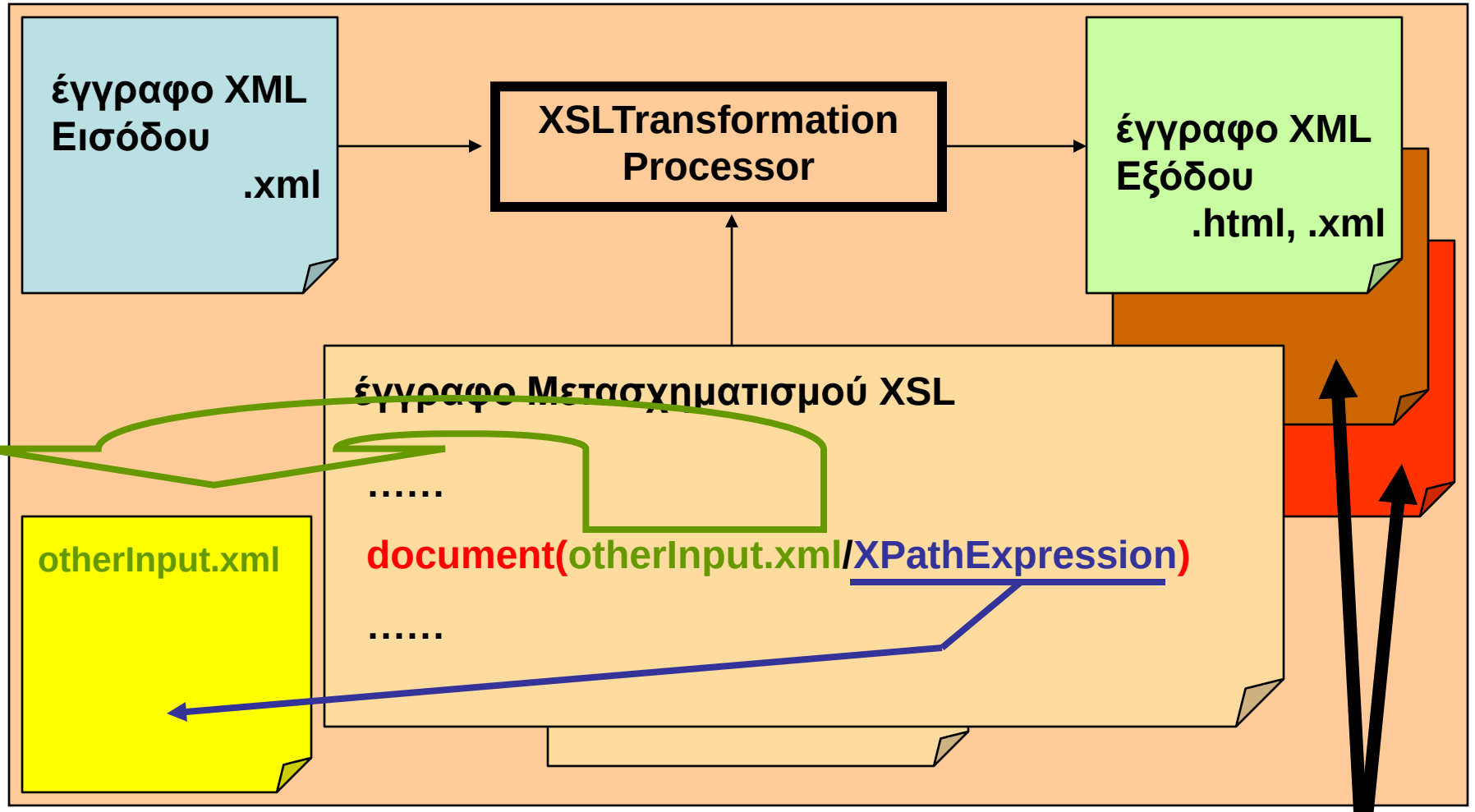
```
<?xml version="1.0"?>
<!-- java net.sf.saxon.Transform -s:generalXML_withoutTransf.xml -xsl:multipleOutputs.xsl -->
<xsl:stylesheet version="2.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <xsl:result-document href="docElName.txt">
          <xsl:for-each select="*">
            <xsl:value-of select="name()"/>
          </xsl:for-each>
      </xsl:result-document>
      <xsl:apply-templates select="*"/>
   </xsl:template>

   <xsl:template match="/*">
      <xsl:result-document href="childElNames.txt">
          <xsl:for-each select="*">
            <xsl:value-of select="name()"/>
          </xsl:for-each>
      </xsl:result-document>
   </xsl:template>
</xsl:stylesheet>
```

Το αποτέλεσμα της εφαρμογής εκάστου των δύο templates εξάγεται σε άλλο αρχείο !!!

χάρις στο **xsl:result-document href="*uri*">**

Πολλαπλά Έγγραφα Εισόδου - ΟΚ

Πολλαπλά Έγγραφα Εξόδου ??



ΔΕΝ ΓΙΝΕΤΑΙ ΜΕ XSLT – Χρειάζεται να μάθομε τα

XSLT Extensions

??