XML Schema - I

Per la descrizione di XML schema, far riferimento a

XML Schema Part 0: Primer Second Edition W3C Recommendation 28 October 2004 http://www.w3.org/TR/xmlschema-0/

XML Schema - II

Sintassi per la specifica della struttura di documenti XML

- più recente delle DTD, permette di definire linguaggi basati su XML in maggior dettaglio
  - specifica la **struttura** dei dati
  - specifica il **tipo del contenuto** dei dati
- NB: DTD specifica la struttura e l'organizzazione dei tag, ma non specifica dettagliatamente il tipo del contenuto
  - <!ELEMENT price (#PCDATA)>
  - <price>6.90</price>
  - <price>**ciao**</price>

XML Schema - III

- definizione della struttura degli elementi dei documenti
  - **tipi semplici** (tipi XML built-in e **derivati**)
  - **tipi complessi** (definiti combinando tipi semplici)
- permette di specificare elementi del documento necessari, opzionali, …
- **un XML schema è esso stesso un documento XML, di estensione .xsd (W3C) o .xml (Microsoft)**
- *NB: esistono proposte di standard di rappresentazione da parte di Microsoft, W3C e altri. Noi vediamo W3C*

Esempio di XML Schema: address.xsd

- Consideriamo documenti XML che descrivono **indirizzi postali.** Es:

```
<?xml version="1.0" encoding="UTF-8"?>
    <addr country="Italy">
      <name>Paolo Bianchi</name>
      <street>via Po</street>
      <num>123</num>
      <city>Torino</city>
 </addr>
```

- ⇒ anzichè specificare la sintassi con una DTD, specifichiamola con un XML-schema
  - **address.xsd** (XML schema): **definisce struttura di documenti XML** che descrivono un indirizzo postale ciascuno

address.xsd - I

Prologo XML (come in tutti i documenti XML)

```
<?xml version="1.0" encoding="UTF-8"?>
```

Riferimento alla grammatica di XMLSchema del W3C (namespace, vd. poi)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="addr" type="IndirizzoType"/>
<xsd:complexType name="IndirizzoType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="num" type="xsd:positiveInteger"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```


address.xsd - II

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

*Dichiaraz. **elemento XML root**: tag "addr", di tipo "IndirizzoType": Tutti i documenti XML validi rispetto allo schema devono avere come radice un tag "addr" che rispetta la definizione data*

```
<xsd:element name="addr" type="IndirizzoType"/>

<xsd:complexType name="IndirizzoType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="num" type="xsd:positiveInteger"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

*Dichiarazione di **tipo di dati complesso globale** (ha un nome ⇒ puo' essere utilizzato in tutto lo schema)*

Dichiarazione attributo degli elementi IndirizzoType

complexTypes

- Permettono di definire la struttura di tag complessi:
  - Elencano le componenti dei tag complessi, che sono obbligatorie (a meno che non si specifichi che una componente e' opzionale)
  - Le componenti sono incluse in una <sequence>
  - Nei documenti XML le componenti devono apparire nell'esatto ordine specificato dalla <sequence> del complexType
  - Una <sequence> puo' contenere
    - elementi di tipo semplice (tipi built-in, come xsd:string, etc. ed elementi definiti da simpleType, vd. poi)
    - elementi di tipo complesso (definiti da altri complexTypes)
    - attributi, che possono solo essere di tipo simpleType

address.xsd – documenti XML validi e non

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="addr" type="IndirizzoType"/>
<xsd:complexType name="IndirizzoType">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="num" type="xsd:positiveInteger"/>
  <xsd:element name="city" type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0"?>
<addr country="Italy">
  <name>Ivo Re</name>
  <street>via Po</street>
  <num>123</num>
  <city>Torino</city>
</addr>
```

```
<addr country="Italy">
  <name>Ivo Re</name>
  <street>via Po</street>
  <num>123/A</num>
  <city>Torino</city>
</addr>
```

```
<?xml version="1.0"?>
<addr country="Italy">
  <name>Ivo Re</name>
  <street>via Po</street>
  <num>123</num>
</addr>
```

```
<addr country="Italy">
  <street>via Po</street>
  <num>123</num>
  <name>Ivo Re</name>
  <city>Torino</city>
</addr>
```

```
<?xml version="1.0"?>
<addr country="Italy">
  …
</addr>
<addr country="Italy">
  …
</addr>
```

XML Schema - tipi di dati globali: recapiti.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="recapiti" type="IndirizziType"/>
<xsd:complexType name="IndirizziType">
<xsd:sequence>
   <xsd:element name="casa" type="IndirizzoType"/>
   <xsd:element name="ufficio" type="IndirizzoType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IndirizzoType">
 <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="num" type="xsd:positiveInteger"/>
   <xsd:element name="city" type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

*Dichiarando **complexType globali** posso definire + tag dello stesso tipo senza ripetere struttura*

iMieiRecapiti.xml: documento XML *(XML instance document)* valido rispetto a recapiti.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
   <recapiti>
        <casa country="Italy">
                <name>Paolo Bianchi</name>
                <street>via Po</street>
                <num>123</num>
                <city>Torino</city>
        </casa>
        <ufficio country="Italy">
                <name>Paolo Bianchi</name>
                <street>via Roma</street>
                <num>25</num>
                <city>Milano</city>
        </ufficio>
   </recapiti>
```

XML Schema – ripetizione di elementi: indirizzi.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:complexType name="indirizzoType">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:string"></xsd:element>
         <xsd:element name="street" type="xsd:string"></xsd:element>
         <xsd:element name="num" type="xsd:positiveInteger"></xsd:element>
         <xsd:element name="city" type="xsd:string"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="indirizziType">
      <xsd:sequence>
         <xsd:element name="addr" type="indirizzoType"
                                  maxOccurs="unbounded"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="agenda" type="indirizziType"></xsd:element>
</xsd:schema>
```

Una <sequence> puo' contenere elementi ripetuti (maxOccurs="unbounded")

laMiaAgenda.xml: documento XML *(XML instance document)* valido rispetto a indirizzi.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
      <addr>
         <name> Paolo Bianchi </name>
         <street> via Po </street>
         <num> 12 </num>
         <city> Roma </city>
      </addr>
      <addr>
         <name>Laura Neri</name>
         <street>via Po</street>
         <num>78</num>
         <city>Torino</city>
      </addr>
</agenda>
```

Nel documento XML potrebbe esserci un numero qualunque (>0) di elementi "addr"

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
</agenda>
```

XML Schema: coffeeOrder.xsd

Vediamo ora un documento XML Schema piu' complesso: **coffeeOrder.xsd**

- coffeeOrder.xsd definisce la struttura di documenti XML che descrivono ordini di caffe'
- Nello schema si usano **complexType anonymous:** sono **tipi di dati senza nome** $\Rightarrow$ non li si puo' referenziare in altre parti dell'XML Schema (utili per definizioni locali ad un elemento)

Esempio di documento XML - coffeeOrd.xml

```
<?xml version="1.0"?>
<coffeeOrder orderDate="1999-10-20">
   <billTo country="Italy">
      <name>Paolo Bianchi</name>
      <street>123 via Po</street>
      <city>Torino</city>
   </billTo>
   <items>
    <item partNum="242-NO" >
          <coffeeName>Lavazza Oro</coffeeName>
          <quantity>5</quantity>
          <price>19.99</price>
      </item>
    <item partNum="242-MU" >
          <coffeeName>Nescafe</coffeeName>
          <quantity>3</quantity>
          <price>19.98</price>
      </item>
   </items>
</coffeeOrder>
```

Documento XML che descrive un ordine di caffè

Schema XML: coffeeOrder.xsd – parte I

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="coffeeOrder" type="CoffeeOrderType"/>
<xsd:complexType name="CoffeeOrderType">
  <xsd:sequence>
    <xsd:element name="billTo" type="Address"/>
    <xsd:element name="items" type="Items"/>
    <xsd:element name=”comment" type=”xsd:string” minOccurs= ”0"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
```

…….. Continua ……

Schema XML: coffeeOrder.xsd – parte II

```
…….. Continua ……
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="coffeeName" type="xsd:string"/>
          <xsd:element name="quantity“ type=“xsd:positiveInteger”/>
          <xsd:element name="price" type="xsd:decimal"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Sequenza di elementi variabile: contiene almeno un item

complexType anonimo

Dichiarazione di attributo necessario

Vincoli di occorrenza di elementi e attributi

- specificano numero min e max di occorrenze degli elementi. Es:

```
<xsd:complexType name=”AnnotatedOrder">
  <xsd:sequence>
    ...
    <xsd:element name=”comment" type=”xsd:string” minOccurs= ”0"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

elemento “comment” è opzionale

- Se non specificati, minOccurs=maxOccurs=1
  - <xsd:element name=”comment" type=”xsd:string” **minOccurs= ”0” maxOccurs= ”3"/>**
- Per attributi, minOccurs >= 0; maxOccurs =1

Attributi: Valori di default

- Definiscono il valore assunto da un attributo quando non specificato nel documento XML. Es:

```
<xsd:complexType name="Address">
  <xsd:sequence>
  ….
  </xsd:sequence>
 <xsd:attribute name="country" type="xsd:string” minOccurs=”0”
                                          default=“Italy”/>
</xsd:complexType>
```

- Si può specificare il valore di default per i soli attributi opzionali

Attributi: Valori fissi

- Impongono valore fisso: se un attributo viene specificato in un documento XML, esso deve avere quel valore. Se viene omesso l'attributo, il valore fisso viene associato automaticamente ($\Rightarrow$ attributo opzionale)

```
<xsd:complexType name="Address">
   ...
   <xsd:attribute name="country" type="xsd:string" fixed="Italy"/>
</xsd:complexType>
```

- Default value e fixed values sono mutuamente esclusivi

Tipi di dati semplici (simpleType) - I

- Tutti i tipi XML base (string, float, date, …)
- Tipi derivati da tipi built-in e derivati stessi (ma semplici)
  - definiti per **restrizione**

```
<xsd:simpleType name="MyInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="9"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - definiti per **enumerazione**

```
<xsd:simpleType name="Città">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Torino"/>
    <xsd:enumeration value="Milano"/>
    <xsd:enumeration value="Venezia"/>
  </xsd:restriction>
</xsd:simpleType>
```

*Si possono anche fare restrizioni per **pattern** (espressioni regolari)*

Tipi di dati semplici (simpleType) - II

- Liste di elementi di tipi semplici (NON di tipi di dati complessi o di altre liste)

```
<xsd:element name=”lista" type=”MyListType"/>
<xsd:simpleType name=”MyListType">
  <xsd:list itemType=“myInteger”>
</xsd:simpleType>
```

```
<lista>0 3 2 8 9 6 7 2 4 1</lista>
```

- Si può specificare lunghezza fissa, minima, massima di lista mediante restrizione

```
<xsd:simpleType name=”SeiNumeri">
  <xsd:restriction base=“MyListType”>
    <xsd:length value=“6”>
  </xsd:restriction>
</xsd:simpleType>
```

Facets:
length: *specifica lunghezza fissa*
minLength: *specifica lungh min*
maxLength: *specifica lungh max*

Tipi di dati complessi (complexType) - I

- Estensioni di tipi semplici (associo attributi)

```
<xsd:element name=”prezzo"/>
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base=“xsd:decimal”>
    <xsd:attribute name=”valuta" type="xsd:string”/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:prezzo>
```

```
<prezzo valuta=”EUR”>35,7</prezzo>
```

```
<prezzo valuta=”EUR” valore=“35,7”/>
```

- Tipi complessi vuoti (senza elementi, solo attributi)

```
<xsd:element name=”prezzo" type=”InternationalPrice"/>
<xsd:complexType name=”InternationalPrice">
  <xsd:attribute name=”valuta" type="xsd:string”/>
  <xsd:attribute name=”valore" type="xsd:decimal”/>
</xsd:complexType>
```

Esempio di uso di simpleType: coffeeOrder.xsd – *versione piu' complessa di quella precedentemente vista*

```
<xsd:complexType>
      <xsd:sequence>
       <xsd:element name="coffeeName" type="xsd:string"/>
       <xsd:element name="quantity">
        <xsd:simpleType>
         <xsd:restriction base="xsd:positiveInteger">
          <xsd:maxExclusive value="100"/>
         </xsd:restriction>
        </xsd:simpleType>
       </xsd:element>
       <xsd:element name="price" type="xsd:decimal"/>
      </xsd:sequence>
     <xsd:attribute name="partNum" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Dichiarazione anonymous di tipo di dato semplice: Per restringere quantity all'intervallo [0, 99]

Tipi di dati complessi - II

- Raggruppamento di elementi: sequence e choice

```
<xsd:element name="billTo" type="Address"/> />
<xsd:complexType name="Address">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  ...
  <xsd:choice>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="PO-BOX" type="xsd:string"/>
  </xsd:choice>
</xsd:sequence>
 ...
</xsd:complexType>
```

Gli elementi di una choice sono mutuamente esclusivi nei documenti XML (in un documento compare solo uno dei tag della choice)

```
<billTo country="Italy" >
 <name>Carla Verdi</name>
 <city>Roma</city>
 <PO-BOX>125</PO-BOX>
</billTo>
```

```
<billTo country="GB" >
 <name>Paul Bley</name>
 <city>Londra</city>
 <street>Oxford Street</street>
</billTo>
```

```
<billTo country="Italy" >
 <name>Carla Verdi</name>
 <city>Roma</city>
 <street>via Salaria</street>
 <PO-BOX>125</PO-BOX>
</billTo>
```

Derivazione di tipi complessi

Servono per dare rappresentazione compatta dei dati e per riutilizzare definizioni di tipi di dati gia' esistenti

- derivazione per estensione
- derivazione per restrizione
- *si possono anche ridefinire tipi di dati, ma trascuriamo*

Derivazione di tipi complessi per estensione

```
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
</xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
```

```
<xsd:complexType name="USAddress">
  <xsd:complexContent>
    <extension base="Address">
      <xsd:sequence>
        <xsd:element name="region" type="xsd:string"/>
        <xsd:element name="zip" type="positiveInteger"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Aggiungo campi "region" e "zip"

Derivazione di tipi complessi per restrizione - I

Restrizione di vincoli di occorrenza

```
<xsd:complexType name="Items" type="itemType">
 <xsd:sequence>
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
 </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="ConfirmedItems" type="itemType">
 <xsd:complexContent>
  <restriction base="Items">
   <xsd:sequence>
    <xsd:element name="item" minOccurs="1" maxOccurs="100">
   </xsd:sequence>
  </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

Restringo cardinalità min e max di items

Derivazione di tipi complessi per restrizione - II

Altri tipi di restrizione:

- assegnazione di valore di default
- assegnazione di valore fisso
- restrizione di tipo (es: da *anyType* a *positiveInteger*)
- ...

Esistono altri costrutti ma li trascuriamo. Per dettagli, vedere le specifiche W3C online

Validazione documenti XML rispetto a XML Schema

Il W3C offre validatore on-line per

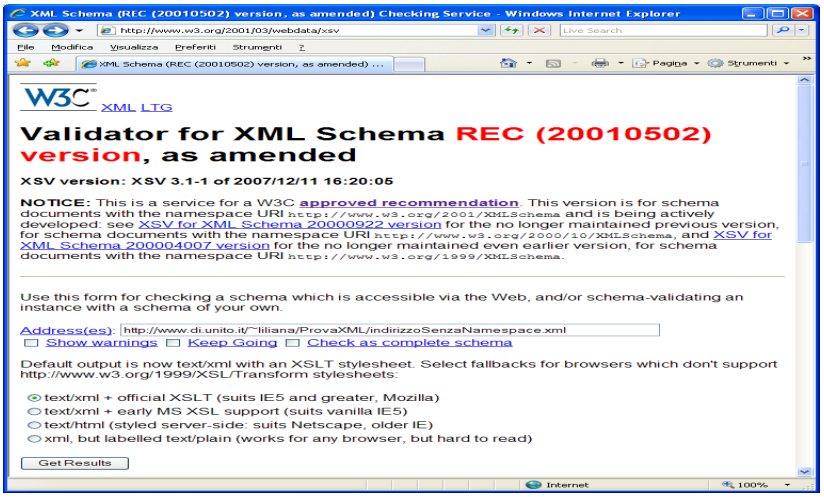
- *Verificare che un documento XML (o XML Schema) sia **ben formato***
- *Verificare che un documento sia **valido** rispetto a un XML Schema di riferimento*

http://www.w3.org/2001/03/webdata/xsv

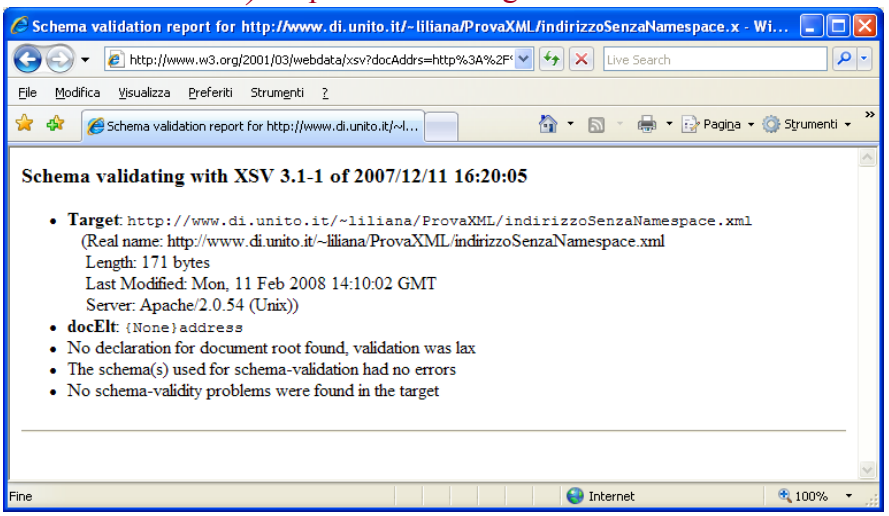
Documento XML: indirizzoSenzaNamespace.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<address country="Italy">
  <name>Paolo Bianchi</name>
  <street>via Po</street>
  <num>123</num>
  <city>Torino</city>
</address>
```

Verifico se indirizzoSenzaNamespace.xml e' ben formato, con validatore del W3C
http://www.w3.org/2001/03/webdata/xsv



indirizzoSenzaNamespace.xml e' ben formato *(lax validation)* http://www.w3.org/2001/03/webdata/xsv



Validazione di documenti XML rispetto a XML Schema *(strict validation)*

Abbiamo visto come associare una DTD ad un documento XML (DOCTYPE)

Dato un XML Schema, come specificare che un documento XML si conforma a tale schema?

Per abbinare documenti a XML Schema di riferimento si usano **i Namespace**

Namespace – spazio di nomi

- Grammatica (documento di tipo XMLSchema) pubblica, accessibile a tutti via rete
  - Es: dialetti di XML proposti come standard (WSDL, ...)
- **Namespace**:
  - **definisce uno spazio di nomi** che raccoglie un insieme di tag e di definizioni
  - serve per **identificare univocamente** etichette e tipi di dati, regole sintattiche
  - **identificato da URI** (Uniform Resource Identifier), tipicamente associato al nome del dominio del DNS, perchè univoco
  - **Documentazione W3C: http://www.w3.org/TR/xml-names/**

Esempio: XML Schema namespace - I

URN: **http://www.w3.org/2001/XMLSchema**

- specifica la grammatica degli XML Schema secondo il W3C *(versione del 2001)*
  - **tipi di dato built-in di XML**: string, decimal, ...
  - **parole riservate** (tag): element, attribute, sequence, simpleType, complexType, restriction, …

Esempio: XML Schema namespace - II

- **Tutti i documenti di tipo XMLSchema specificano** lo schema di riferimento nel prologo:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

- In questo modo si dichiara quale grammatica usare per interpretare i tag utilizzati
  - Es: se un documento XMLSchema fa riferimento al namespace del W3C, i suoi tag “element”, etc., e i tipi di dati utilizzati (es., “string”), sono quelli definiti nel namespace di XMLSchema pubblicato dal W3C

Tag qualified e unqualified - I

Dato il namespace di riferimento di uno XMLSchema (o di uno XML document instance), i tag del documento possono essere scritti in due modi:

- ***Unqualified****:* si riporta il nome del tag, senza namespace di riferimento (es: element)
- ***Namespace aware (qualified):*** si riporta il nome del tag con name space di riferimento (es: **xsd:**element)

Normalmente si usano tag qualified perche' per ogni tag si puo' specificare esattamente il namespace che lo definisce $\Rightarrow$ *non ambiguo*

XML schema address.xsd (tag qualified)

```
<?xml version="1.0" encoding="UTF-8"?>
```

Namespace rappresentato da etichetta "xsd"

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Tag qualified rispetto al namespace http://www.w3.org/2001/XMLSchema

```
<xsd:element name="address" type="IndirizzoType"/>
<xsd:complexType name="IndirizzoType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="num" type="xsd:positiveInteger"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

XML schema address1.xsd (tag qualified)

```
<?xml version="1.0" encoding="UTF-8"?>
```

Namespace rappresentato da etichetta "ppp"

```
<ppp:schema xmlns:ppp="http://www.w3.org/2001/XMLSchema">
<ppp:element name="address" type="IndirizzoType"/>
< ppp :complexType name="IndirizzoType">
  <ppp:sequence>
    <ppp:element name="name" type="ppp:string"/>
    <ppp:element name="street" type="ppp:string"/>
    <ppp:element name="num" type="ppp:positiveInteger"/>
    <ppp:element name="city" type="ppp:string"/>
  </ppp:sequence>
  <ppp:attribute name="country" type="ppp:string"/>
</ppp:complexType>
</ppp:schema>
```

Normalmente si usa xsd (xml schema definition)…

XML schema address2.xsd (tag unqualified)

```
<?xml version="1.0" encoding="UTF-8"?>
```

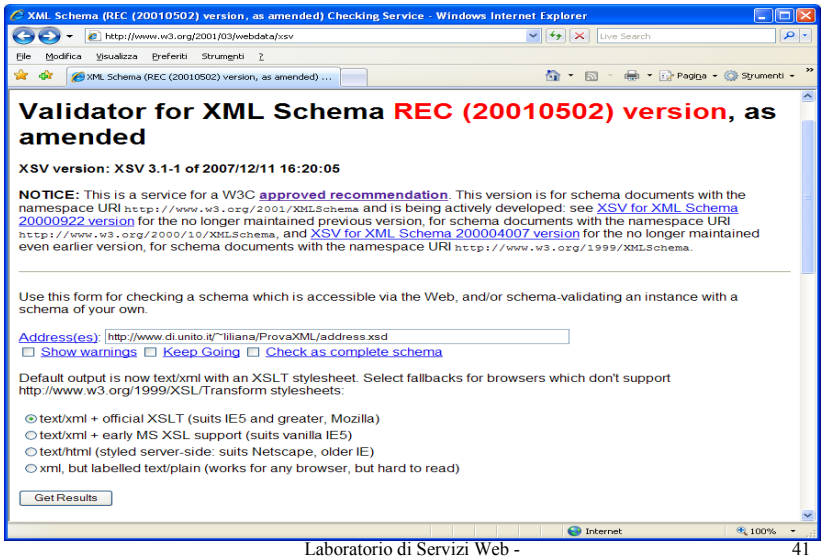
Namespace rappresentato da etichetta "xsd"

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

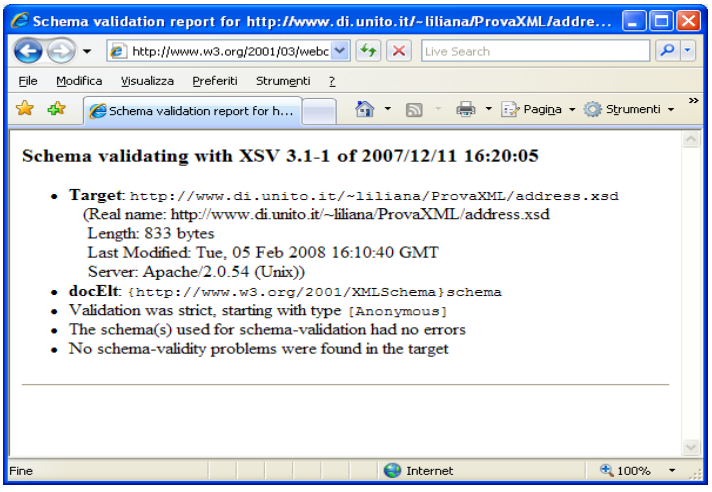
Tag unqualified

```
<element name="address" type="IndirizzoType"/>
<complexType name="IndirizzoType">
  <sequence>
    <element name="name" type="string"/>
    <element name="street" type="string"/>
    <element name="num" type="positiveInteger"/>
    <element name="city" type="string"/>
  <sequence>
  <attribute name="country" type="string"/>
<complexType>
</schema>
```

Validazione di address.xsd (rispetto a namespace di XMLSchema) http://www.w3.org/2001/03/webdata/xsv



Validazione di address.xsd (rispetto a namespace di XMLSchema) http://www.w3.org/2001/03/webdata/xsv



Validazione di documenti XML rispetto a XMLSchema da noi definiti

Per permettere la validazione di un documento XML (XML document instance) rispetto a un mio schema (es: address.xsd)

1. Dichiarare l'XML schema come namespace (per es., *ns*)
2. Indicare nel documento XML che deve rispettare il namespace *ns*

Definiamo namespace address.xsd - I

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/address"
        xmlns:tns="http://xml.netbeans.org/schema/address"
        elementFormDefault="qualified">
   <xsd:complexType name="indirizzoType">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:string"></xsd:element>
         <xsd:element name="street" type="xsd:string"></xsd:element>
         <xsd:element name="num" type="xsd:positiveInteger"></xsd:element>
         <xsd:element name="city" type="xsd:string"></xsd:element>
      </xsd:sequence>
      <xsd:attribute name="country" type="xsd:string" default="Italy"/>
   </xsd:complexType>
   <xsd:element name="addr" type="tns:indirizzoType"></xsd:element>
</xsd:schema>
```

Definiamo namespace address.xsd - II

…

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace=http://xml.netbeans.org/schema/address

Lo schema viene indicato come target namespace (namespace a cui si puo' fare riferimento nei documenti XML)

xmlns:tns=http://xml.netbeans.org/schema/address

tns: etichetta del target namespace all'interno dello schema

elementFormDefault="qualified">

…

</xsd:schema>

Si impone che i documenti XML che fanno riferimento a questo namespace riportino i loro tag in forma qualified *(se specifico "unqualified" impongo che i tag siano unqualified)*

namespace address.xsd
NB: Tutti i tag si riferiscono al namespace che li definisce (vd. colori)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/address"
        xmlns:tns="http://xml.netbeans.org/schema/address"
        elementFormDefault="qualified">
   <xsd:complexType name="indirizzoType">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:string"></xsd:element>
         <xsd:element name="street" type="xsd:string"></xsd:element>
         <xsd:element name="num" type="xsd:positiveInteger"></xsd:element>
         <xsd:element name="city" type="xsd:string"></xsd:element>
      </xsd:sequence>
      <xsd:attribute name="country" type="xsd:string" default="Italy"/>
   </xsd:complexType>
   <xsd:element name="addr" type="tns:indirizzoType"></xsd:element>
</xsd:schema>
```

Associazione di documenti XML (XML document instances) a namespace

Dato un namespace (schema XML con URI)

- Es: 'http://xml.netbeans.org/schema/address'
- per associare documento XML a namespace, aggiungere al tag radice del documento XML:

```
<ns4:addr  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:ns4='http://xml.netbeans.org/schema/address'
  xsi:schemaLocation='http://xml.netbeans.org/schema/address
                                                    address.xsd'>
```

- cioe', si qualifica il tag radice rispetto al namespace di riferimento

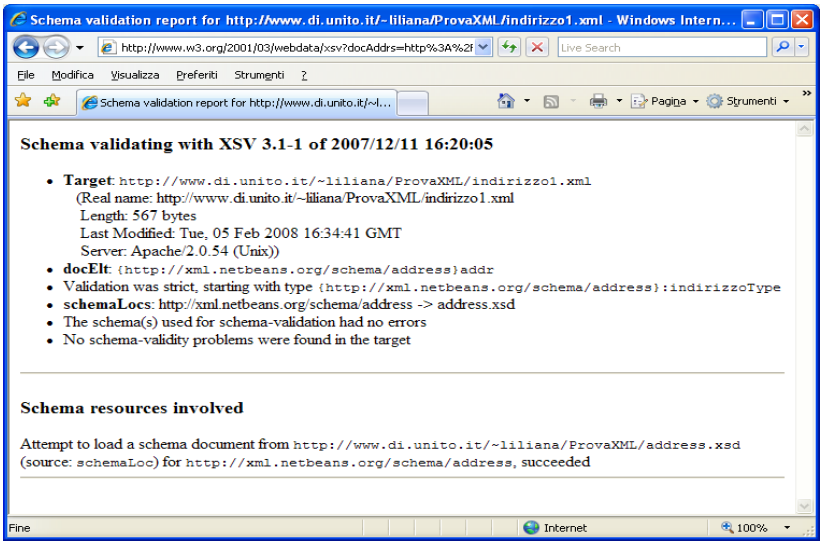
indirizzo1.xml (associato a namespace address.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>

<ns4:addr  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:ns4='http://xml.netbeans.org/schema/address'
  xsi:schemaLocation='http://xml.netbeans.org/schema/address address.xsd'>

    <ns4:name> Paolo Bianchi </ns4:name>
    <ns4:street> via Po </ns4:street>
    <ns4:num> 123 </ns4:num>
    <ns4:city> Torino </ns4:city>
</ns4:addr>
```

Validazione (OK) con http://www.w3.org/2001/03/webdata/xsv



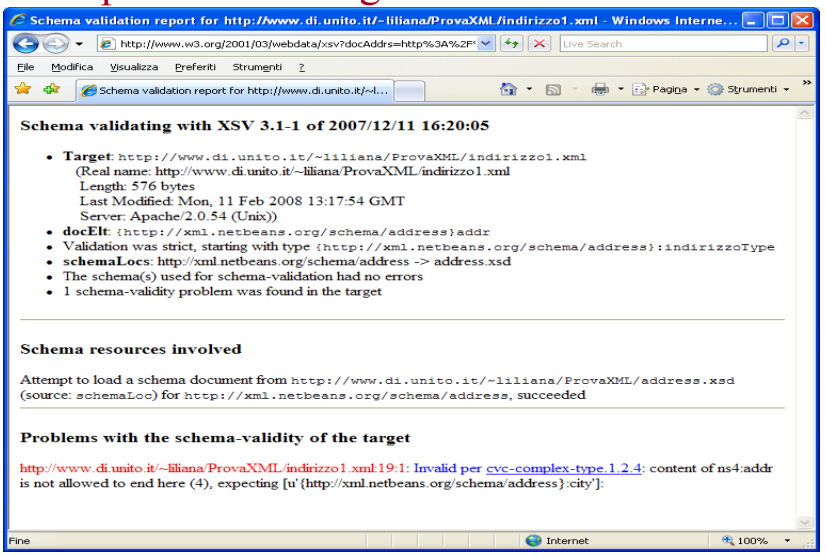
indirizzo1.xml: ben formato, ma non valido rispetto ad address.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<ns4:addr  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:ns4='http://xml.netbeans.org/schema/address'
  xsi:schemaLocation='http://xml.netbeans.org/schema/address address.xsd'>

    <ns4:name> Paolo Bianchi </ns4:name>
    <ns4:street> via Po </ns4:street>
    <ns4:num> 123 </ns4:num>
    <!– manca l'elemento city -->
</ns4:addr>
```

Validazione (KO) con http://www.w3.org/2001/03/webdata/xsv

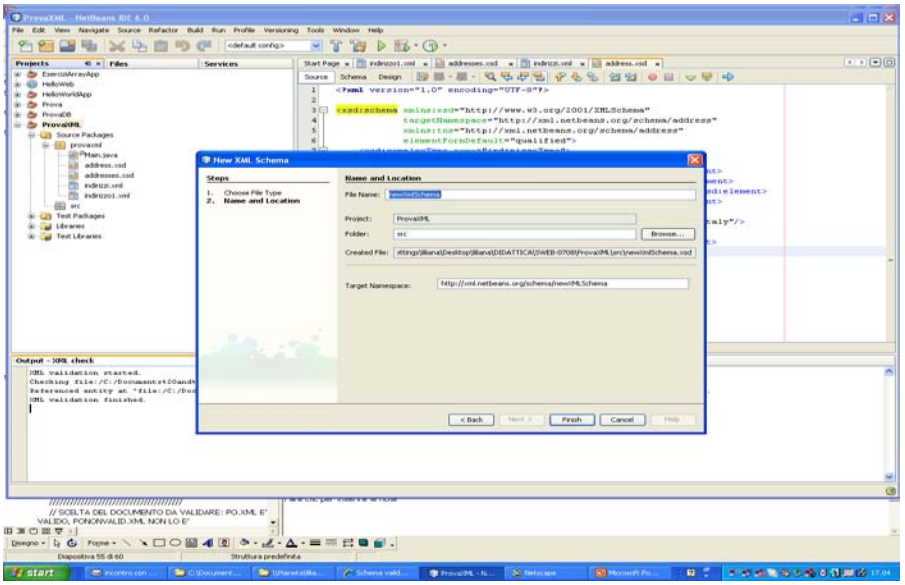


Gestione di documenti XML con NetBeans

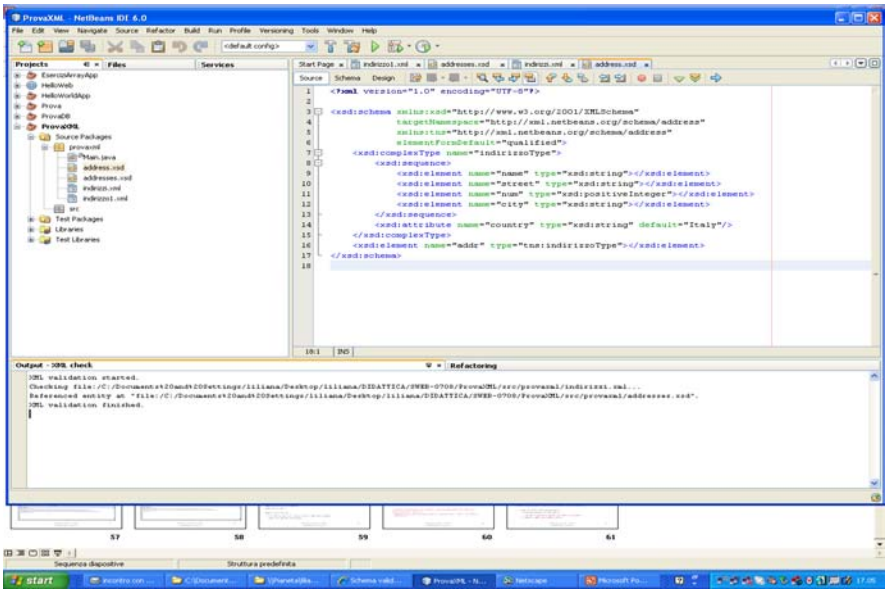
NetBeans offre supporto allo sviluppo di documenti XML e XML Schema, e ne permette verifiche di ben formatezza e validazione

- creare progetto (ProvaXML)
- nel progetto, New File (XML Schema)

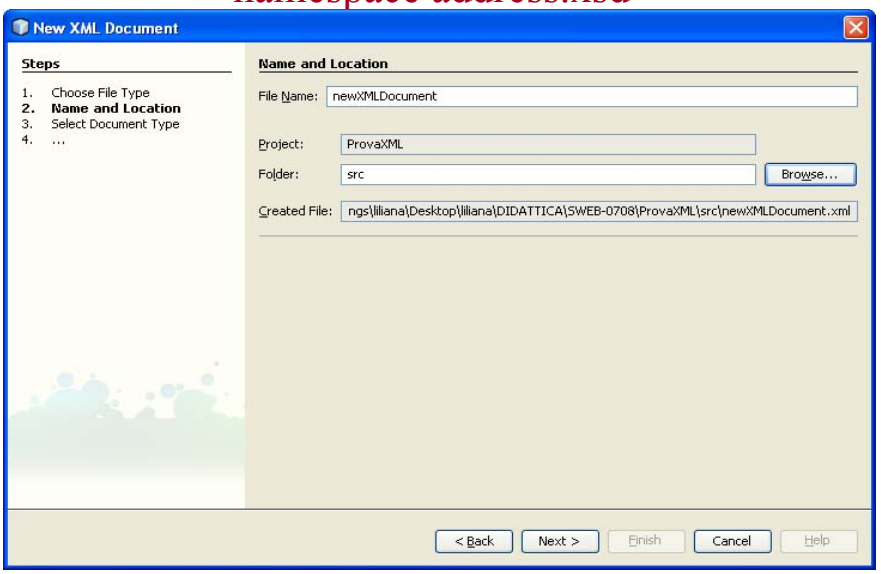
Creazione XML schema



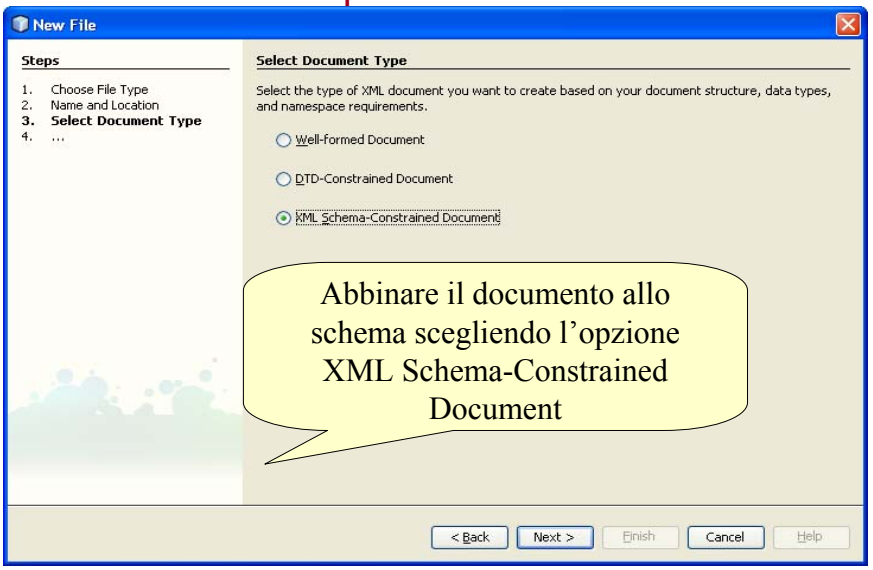
address.xsd



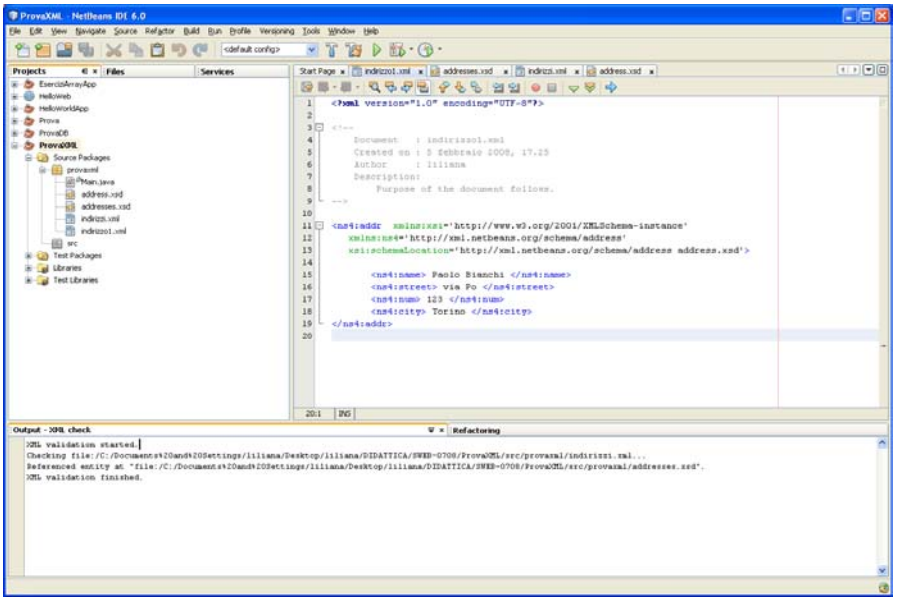
Creazione documento XML da validare rispetto a namespace address.xsd



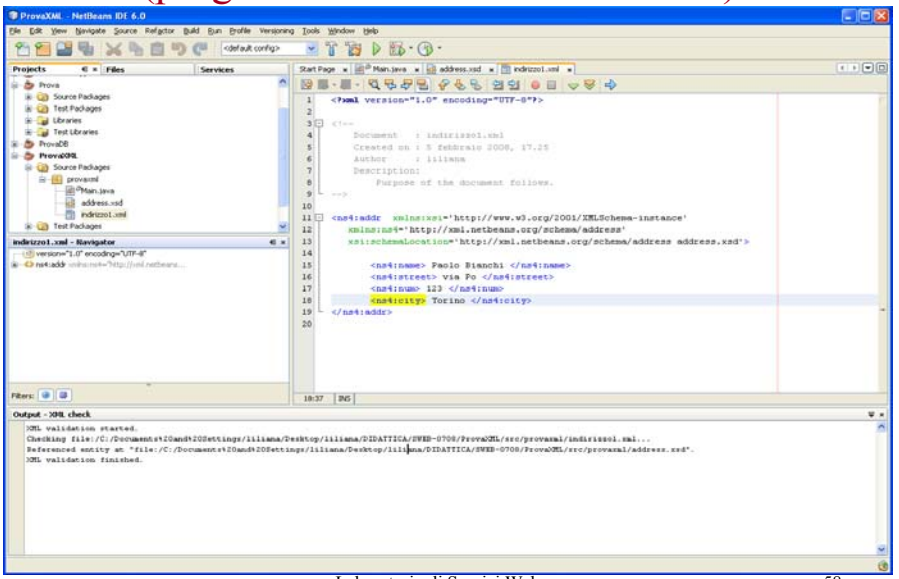
Creazione documento XML da validare rispetto a namespace address.xsd



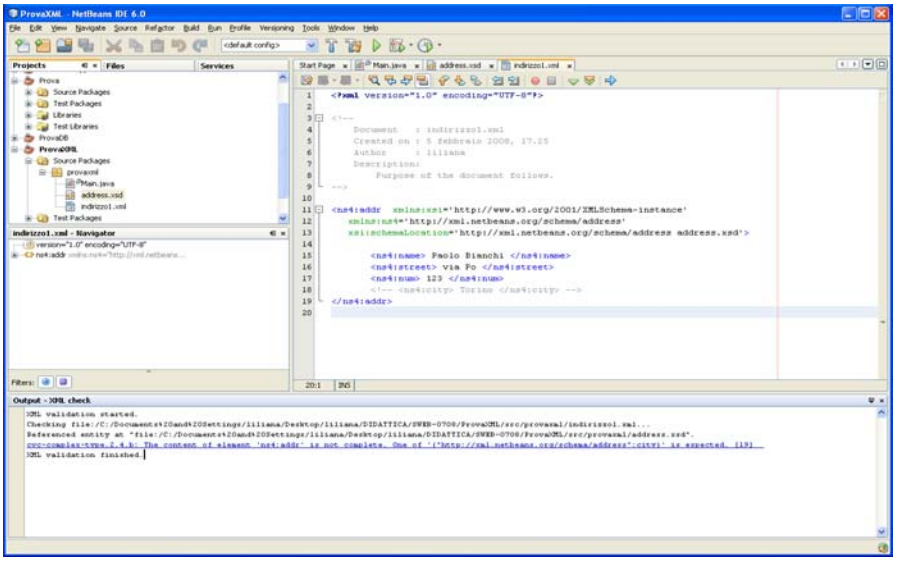
indirizzo1.xml



Validazione di indirizzo1.xml (OK) con NetBeans (progetto di NetBeans – file locali)



Validazione di indirizzo1.xml (KO) con NetBeans (progetto di NetBeans – file locali)



Verifica documenti XML da applicazione Java

Java offre analizzatori sintattici di XML per verificare se un documento XML e' ben formato e se e' valido rispetto a uno schema XML

- Lo schema e il documento XML devono essere dati in input all'applicazione
- L'applicazione deve invocare esplicitamente il parser per effettuare l'analisi sintattica
- *Vediamo velocemente un esempio*

Esempio: validazione documento XML rispetto a XML Schema da applicazione Java - I

```
import javax.xml.validation.*;
import javax.xml.XMLConstants;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;
import javax.xml.parsers.*;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

import java.io.File;
import java.io.IOException;
import java.util.*;

public class Valida  {
   // Global value so it can be referred by the tree-adapter
   static Document document;
…
```

Usa DOM e parser che legge XML Schema

Esempio: validazione documento XML rispetto a XML Schema da applicazione Java - II

```
public class Valida  {
   // Global value so it can be referred by the tree-adapter
   static Document document;

   public static void main(String argv[]) {
   try {
   DocumentBuilder parser =
    DocumentBuilderFactory.newInstance().newDocumentBuilder();


   Document document = parser.parse(new File("po.xml"));

   // create a SchemaFactory capable of understanding WXS schemas
   SchemaFactory factory =
    SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA
    _NS_URI);
…
```

Usa DOM per gestire documento

Esempio: validazione documento XML da XML Schema - III

Imposta lo schema per validare

```
// load a WXS schema, represented by a Schema instance
Source schemaFile = new StreamSource(new File("po.xsd"));
Schema schema = factory.newSchema(schemaFile);

// create a Validator instance, which can be used to validate an instance
 document
Validator validator = schema.newValidator();

System.out.println("Prima della validazione del documento");

// validate the DOM tree
validator.validate(new DOMSource(document));

System.out.println("Dopo la validazione del documento");

}   catch (SAXParseException spe) {
      // Error generated by the parser
      … catturo e gestisco molte possibili eccezioni…
      // vedere codice java della classe
…
}
```