XML

Today

XML: Extensible Markup Language

XSD: XML Schema

XQUERY/XPATH: Accessing XML

Semantic Tower Defense ;)

Game The state of a game of STD

Tower A tower in the STD world which defends the exit

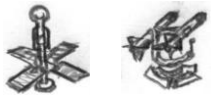
Creep A creepy bad character trying to reach the exit

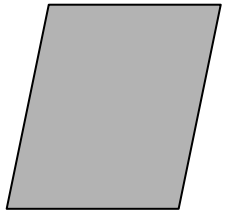
Creep

Type
MoveSpeed
Armor
Cost
CanFly
…

Tower

Type
ReloadSpeed
Power
Cost
…

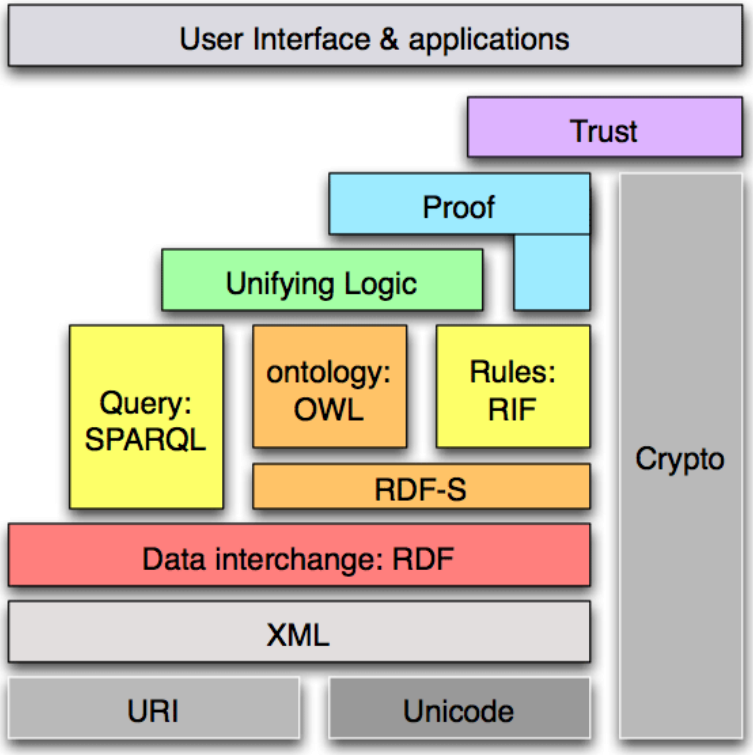




Game

Turrets
Creeps
Level
Money
…

Semantic Web Stack revisited



Unicode

Unicode is short for the **Universal Character Set**
Unicode is an International Industry Standard
Unicode is an encoding Standard for text and URIs
It combines different character sets (e.g. Arabian)
Find more information at http://www.unicode.org/

►**Resources in the Semantic Web are encoded using Unicode**

URI

http://www.std.no/creeps?name=sadsight

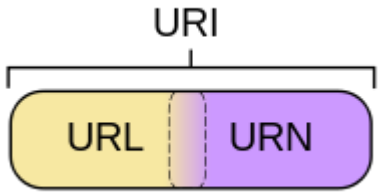
URI stands for **U**niform **R**esource **I**dentifier
URIs are a standard for **identification of resources**
An URI is a **Virtual Pointer** to a concept, thing, blurp
A central goal is to reduce or **remove ambiguity**
„The Who“ ?=? „Who“
“Apache” (Helicopter) ?=? “Apache” (Server)
„Michael Granitzer“ ?=? „M. Granitzer“ ?=? „Dr. Granitzer“
Another goal is to **support decentrality** through unique identification of resources
URIs are defined by **RFC 3986 - Uniform Resource Identifier (URI)**
They are a **generalization of URL‘s (e.g. http://)** and URN‘s (e.g. ISBN)

► **Resources in the Semantic Web are identified via URI‘s**

Does not mean the URI is a valid URL associated with content!

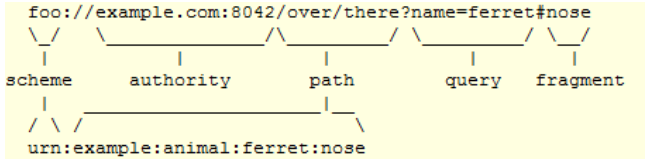
URI, URL, URN, IRI?

An **URL** is a **U**niform **R**esource **L**ocator **on the Web**
http://en.wikipedia.org/wiki/The_Last_Unicorn
An **URN** is a **U**niform **R**esource **N**ame for **identification**
urn:isbn:0451450523



► **An URI can be an URL, an URN or both**

An URI may consist of **scheme**, **authority**, **path**, **query** and **fragment**
Every URI must feature **scheme** and **path**



An IRI is a sequence of characters from the Universal Character Set
► **IRIs** are an **internationalization of URIs**

XML

XML stands for E**X**tensible **M**arkup **L**anguage
It defines a **set of rules** for **encoding documents**
It is utilized for the structured storage and exchange of data
It provides constructs for specifying **encoding**, **markup**, **content** and **comments**

► **XML is a standard for defining standards. It does not "do" anything.**

XML employs a **tree-based markup representation** of documents
Opening and closing **Tags** are used to express document structure in **elements**
Attributes are used to express properties of tags
Content is stored in elements and may include **entities**

► **XML documents can automatically be validated for being well-formed**

http://validator.w3.org/#validate_by_input

XML Syntax

Must have a **single root** element encapsuling all other elements

Must **correctly nest** tags

Case sensitive tags. No **syntax characters** outside tags (use escapes)

Attributes must be **quoted**

Character Entities must be used in content

< **<** > **>** & **&** ' **‘** " **“**

Comments: <!-- This is a comment -->

http://validator.w3.org/#validate_by_input

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

<game>
  <description>
    name snakes & ladders
  </description>
</game>

<tower type=Cannon>
  <x>47</x>
  <y>23</y>
</tower>

<tower type=Freeze>
  <x>48</x>
  <y>23</y>
</tower>

// This is a basic bug
<creep type=Bug>
  <x>16</x>
  <y>11</y>
</creep>
```

wrong.xml

XML Syntax

YES

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

<?xml version="1.0" encoding="UTF-8" ?>
<game name="snakes &amp; ladders">
     ...
</game>

<?xml version="1.0" encoding="UTF-8" ?>
<game name="snakes &amp; ladders">
  <tower type ="missile">
    <x>47</x>
    <y>23</y>
    <desc>the &amp;basic&amp; tower</desc>
  </tower>
</game>
```

NO!

```
<?xml version="1.0" encoding="UTF-8" ?>
<tower>...</tower>
<creep>...</creep>


<?xml version="1.0" encoding="UTF-8" ?>
<Game name="snakes & ladders">
     ...
</game>

<?xml version="1.0" encoding="UTF-8" ?>
<game name="snakes &amp; ladders">
  <tower type="missile">
    x=47
    y=23
     <desc>the "basic" tower</desc>
  </tower>
</game>
```

Selected XML topics

CDATA marks sections containing only **C**haracter **Data** and no markup
No need to use entities like & in CDATA
CDATA syntax: <![CDATA[…]]>
CData does not „protect“ data from parsers. It does not encode binary!

XML Namespace declarations avoid confusion between XML vocabularies
Declare a namespace using pseudo-attribute **xmlns:prefix** or xmlns (default)
Namespaces are **identified by URIs**
xmlns:xhtml=http://www.w3.org/1999/xhtml
xmlns=http://www.w3.org/1999/xhtml
Scope of declaration is within the element where you specified it

Binary XML would store xml in a binary format which is machine readable
This would save storage space in some applications but is highly controversial
Use compression if your xml documents get too large (does not speed up parsing)

Selected XML topics

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

<game xmlns:std="http://www.std.org/std" name="snakes &amp; ladders">
  <std:tower type ="missile">
    <std:x>47</std:x>
    <std:y>23</std:y>
    <std:desc>the &amp;basic&amp; tower</std:desc>
  </std:tower>
</game>
```

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

<game xmlns:std="http://www.std.org/std" name="snakes &amp; ladders">
  <std:tower type ="missile">
    <std:x>47</std:x>
    <std:y>23</std:y>
    <std:desc>
      <![CDATA[ now here i cancutloose with "forbidden" & fancstuff ]]>
    </std:desc>
  </std:tower>
</game>
```

Accessing XML

DOM parsers grant random access to the tree structure of an xml document
Memory intensive, slow, but full access to xml document in memory
SAX parsers grant sequential (streaming) access to the tree structure
Memory efficient, fast, but you have to use callbacks to do your magic

Higher-Level APIs provide **Data Bindings**, as known from databases
XML parsing is hidden, XML data exposed as **Objects**
(Un)Marshalling descibes the conversion of objects to and from XML

► **XML is everywhere, parsers and bindings available for all major languages**

Defining XML Structure

XML can be checked for being well-formed, but not for being valid (content)
Schema languages specify rules to which valid documents have to accord
XSD (XML Schema) and **DTD** (Document Type Definition) are most prominent
XSD is recommended. It includes **Datatypes**, **Namespaces** and is itself XML

► **XSD defines rules for a valid XML document**

Element declarations define properties of elements, like name, type and ns
Attribute declarations define properties of attributes, like name, type and ns
Simple types define which (textual) content may appear in an element or attribute
Complex types define permitted element content, including attributes and children

► **XSD enables automatic checking of XML document validity**

XSD Nutshell

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<game>
  <description>my first game</description>
  <score>3000</score>
</game>
```

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="game">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="description" />
        <xs:element name="score" type="xs:integer"
     minOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

www.utilities-online.info/xsdvalidation

XSD Datatypes

Simple types map to well-established primitive types
xs:string, xs:decimal, xs:integer, xs:float, xs:boolean…
Further XML-specific simple types are available
xs:anyURI or *xs:language*

Range restrictions can be applied **based on** the **simple type**
Numeric restrictions for number types
<xs:restriction base="xs:integer"><xs:minInclusive value="1"/></xs:restriction>
Pattern restrictions for text types
<xs:restriction base="xs:string"><xs:pattern value="(A)?[0-9]{4}"/></xs:restriction>
Range can also be restricted to a list of alternatives

Union type available for merging simple types
List of simple type entries available

XSD Datatypes

```
<xs:simpleType name="coord">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="63"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="tower">
  <xs:sequence>
    <xs:element name="x" type="coord"/>
    <xs:element name="y" type="coord"/>
    <xs:element name="type" type="towertype"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="game">
  <xs:sequence>
    <xs:element name="player" type="xs:string"  minOccurs="1" maxOccurs="1"/>
    <xs:element name="score" type="xs:integer" minOccurs="1" maxOccurs="1"/>
    <xs:element name="tower" type="tower" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="game" type="game">
</xs:element>
```

XSD Datatypes

Complex types specify simple types and **element structures**

We can specify which elements should occur in which order and how often

xs:sequence Order-aware list of child elements

xs:choice Selecting one of a numer of possible child elements

xs:all Arbitrary list of child elements

Specify **number of occurrences** using attributes **minOccurs** and **maxOccurs**

Extensions and **restrictions** are supported via **xs:extension** and **xs:restriction**

But this does not constitute true inheritance!

Extension types may specify additional elements

Restriction types may place further restrictions on already defined elements

www.utilities-online.info/xsdvalidation

Using XSD

xmlns:xs references XML Schema namespace (valid for all xs elements)

targetNamespace references your namespace

xmlns default namespace references your namespace

elementFormDefault enforces namespace qualified names

In XML, specify schema location in root element

XSD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.std.org"
  xmlns="http://www.std.org"
  elementFormDefault="qualified">

  <xs:element name="game">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>

</xs:schema>
```

XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<game
xmlns="http://www.std.com"
xmlns:xsi="http://www.std.com/xsi"
xsi:schemaLocation=" http://www.std.com game.xsd"

<game>
  ...
</game>
```

XPath

XPath defines a standard for **adressing parts of an XML document**
XPath considers XML documents from a tree perspective
Basic XPath constructed from a **location path**
Path is sequence of **location steps**
Each step has **Axis**, **Test** and optional predicates

Axis defines along which dimension the step is taken
attribute(@), child, parent, following, parent…
/child::A/child::B/child::C

Test defines tests to be passed by nodes selected in the step
//std:, comment(), text('super') processing-instruction('php')*

Predicates restrict nodes by conditions
[@turrettype='cannon']

Xpath Examples

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<game>
  <score>4328</score>
  <level>16</level>
     <turret type="cannon">
             <x>47</x>
             <y>23</y>
     </turret>
     <turret type="cannon">
             <x>46</x>
             <y>23</y>
     </turret>
     <turret type=„missile">
             <x>46</x>
             <y>22</y>
     </turret>
     <creep type="bug">
             <x>31</x>
             <y>12</y>
             <life>8</life>
     </creep>
</game>
```

XPath

```
/game/turret[@type='missile']

<turret type="missile">
<x>46</x>
<y>22</y>
</turret>
```

http://xpath.online-toolz.com/tools/xpath-editor.php

XQuery and XSLT

XQuery is the **standard query language for XML documents**
XQuery operates on **Sequences** and cardinalities
We can define **arbitrary selections** in Xquery
XQuery features **control flow** functions

XSLT is the E**X**tensible **S**tylesheet **L**anguage **T**ransformations
XSLT transforms XML documents based on a specified transform rule
Target does not have to be XML!

XQuery and XSLT overlap
Both can select from XML
Both can transform XML
Both use XPath

Semantic Tower Defense XML

Game PlayerName(String), Score(Integer)

Tower Type(String, can be Cannon/Freeze), Reload(Integer), Position (X,Y Integer >=0 <64)

Creep Type(String), Speed(Integer >0), Position (X,Y Integer >=0 <64), CanFly(bool)

Write an **XSD** which enables validation of a STD savegame

Write an **example XML** which validates against your XSD

Validate both online

Write a **XPath** statement which selects all freeze towers

Test your statement online

www.utilities-online.info/xsdvalidation

http://xpath.online-toolz.com/tools/xpath-editor.php