

Въведение в XML. Общ преглед на XML технологиите и на приложението на XML



Интернет и Уеб
Web 1.0/2.0/3.0
Маркъп езици
От SGML през HTML до XML
Цели на XML
XML езици и технологии

Интернет и Уеб

- Роля на World Wide Web (Световна паяжина), съкр. WWW, Уеб или Световната мрежа
- Нов начин на общуване между хората, общностите и дори начините за комуникация между компютрите
- Услуги за е-бизнес, електронна търговия, е-икономика, електронно правителство, електронна демокрация, електронно обучение, ...
- Информационен обмен и взаимодействие между различни актори – лица, организации, Уеб приложения, интелигентни агенти, Интернет на нещата, Големи данни...

Интернет и Уеб

- ❖ World Wide Web често се счита за синоним на Интернет, но това не е така
- ❖ Интернет - електронна съобщителна мрежа и структура, на която се базира WWW
- ❖ Уеб - част на Интернет, достъпна посредством графичен потребителски интерфейс и съдържаща приложения, услуги и документи, които често са свързани чрез хипервръзки

Развитие на Уеб

- Тим Бърнърс-Лий, Церн, 1991 г. - разработва основите на Уеб
- Проектира прости средства за пренос на взаимосвързани документи със структурирана информация до всякакви компютри, свързани в Интернет и работещи с различни операционни системи :
 - ✓ език за маркиране на хипертекст (**Hypertext Markup Language, или съкр. HTML**), и
 - ✓ протокол за трансфер на хипертекст (**Hyper Text Transfer Protocol, или съкр. HTTP**)
- По-късно - спецификация на URI (**Uniform Resource Identifier**) - нотация за уникално идентифициране на обекти в целия Интернет.

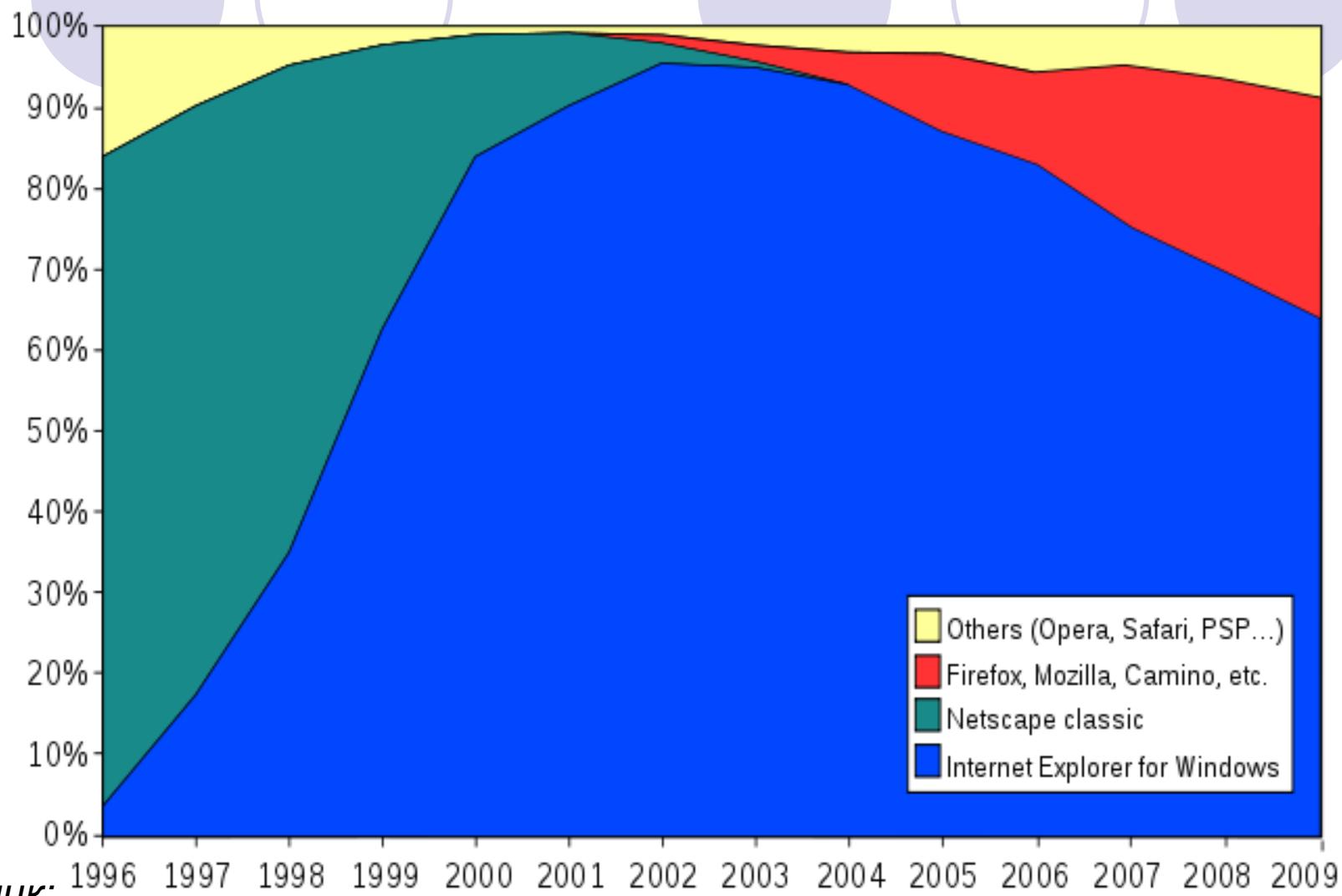
Исторически възход

- 1992 год. - повече от един миллион компютри вече са свързани към Интернет
- 1993 год. - свободна експлоатация на първия браузер NCSA Mosaic
- 1994 г. - Tim Berners-Lee основава ***World Wide Web Consortium (W3C)*** в Massachusetts Institute of Technology, Laboratory for Computer Science (MIT/LCS) с поддръжка от ***Defense Advanced Research Projects Agency (DARPA)***, чиято ARPANET е основата на съвременния Internet
- 1994 год. - Netscape Navigator - "well on its way to becoming the world's standard interface", според Gary Wolfe в статия в сп. *Wired*

Войни на браузърите

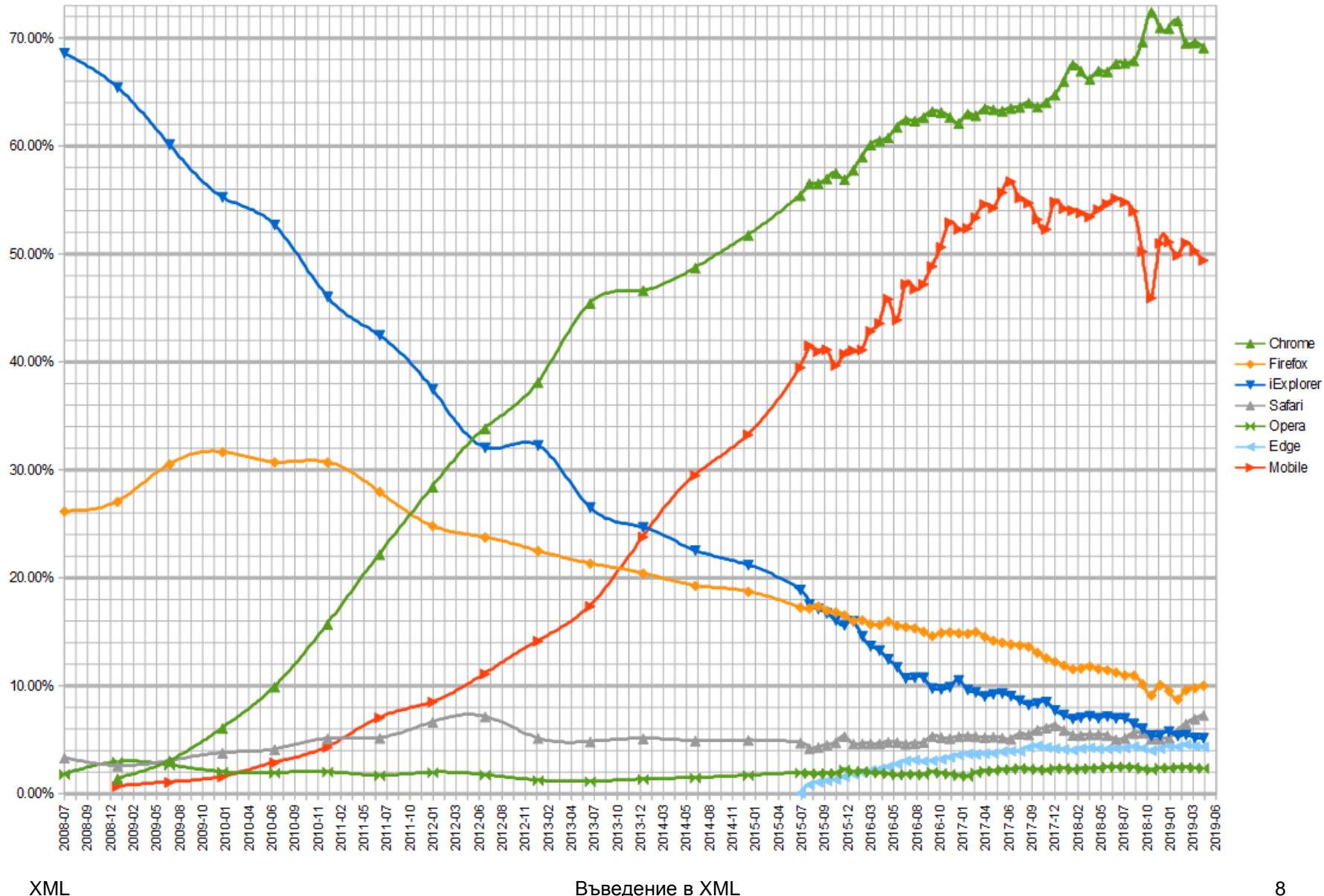
- 1995 год. - Microsoft лицензира Mosaic като Internet Explorer 1.0, пуснат на пазара като част от Microsoft Windows 95 (август 1995)
- Ноември 1995 г. - Microsoft Internet Explorer 2.0 е пуснат за свободна употреба
- 1996 г. - Netscape 3.0 с първа в света поддръжка на стилове - Cascading Style Sheets (CSS)
- 1997 г. - Internet Explorer 4 е интегриран в Microsoft Windows
- Нови видове браузъри с поддръжка на разширени (макар често несъвместими) версии на HTML зализат пазара – виж http://upload.wikimedia.org/wikipedia/commons/7/74/Timeline_of_web_browsers.svg
- Февруари 2011 г. - Internet Explorer 45%, Firefox 30%, Chrome 17%, Safari 5%, Opera 2%, други браузери - около 1%
(източник: StatCounter global market)

Browser Wars



Източник:
Wikipedia
XML

Usage share of browsers (source StatCounter)



Последствие от войните на браузърите

- Надпревара в предлагането на нови функционалности и по-бавно корегиране на проблемите
- Добавяне на уникални за браузера нови маркери (тагове) на съдържание вместо придръжане към стандартите – напр. `<marquee>` в Microsoft's Internet Explorer и `<blink>` в Netscape
- Изоставане в развитието на стандарта HTML, поддържан от W3C
- Проблеми със сигурността – зареждане на вируси при зареждане на документи с активно съдържание (документи-приложения)

Справка за нестандартен HTML

- ❖ Непропоръчителни (deprecated) и патентовани (proprietary):
 - Елементи
 - Атрибути
- ❖ Поддръжка от различни layout engines за съответните браузъри -
[http://en.wikipedia.org/wiki/Comparison_of_layout_engines_\(Non-standard_HTML\)](http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Non-standard_HTML))

Етапи в развитието на Уеб

- Web 1.0
- Web 2.0
- Web 3.0
- ...

Уеб 1.0 (традиционн Уеб) 1/3

- **Web 1.0** - термин за етап от еволюцията на World Wide Web; обхваща периода от 1993 до 2001г.
- **Бизнес модел:**
 - top-down подход за изграждане и използване на WWW - статични страници с хипертекст (Hyperlinks е стандарт на WWW от 1993г.)
 - страници на малцина автори на съдържание (webmasters), зареждани от голям брой потребители с глобален достъп;
 - фокус върху презентацията, а не върху създаването на съдържание;
 - печалби от броя посещения (most visited webpages)

Уеб 1.0 (традиционн Уеб) 2/3

- **Технически характеристики:**
 - статичен хипертекст – без динамика в браузера
 - липса на редактиране на страниците от външни потребители
 - използване на Framesets - рамката (frame) е начин за представяне на няколко Уеб страници и/или медия елементи в един прозорец (или таб) на браузъра:
 - характерен за HTML 3 и 4;
 - липса на поддръжка от много браузъри, лоша индексация от търсачките, трудни връзки и bookmark към рамкираните страници, лошо скролиране при ниска резолюция;
 - изключени от HTML 5

```
<frameset cols="65%, 35%">
  <frame src="URL OF FRAME PAGE 1">
  <frame src="URL OF FRAME PAGE 2">
<noframes> Sorry but your browser do not support frames ☹
</noframes>
</frameset>
```

Уеб 1.0 (традиционн Уеб) 3/3

- **Още технически характеристики:**
 - използване на таблици (`<table>`) за подравняване на съдържанието на страницата
 - отделяне на съдържание с прозрачни 1x1 pixel изображения в GIF format
 - патентовани нестандартни HTML елементи като `<blink>` и `<marquee>`
 - онлайн книги за гости
 - изпращане на HTML форми като ел. поща от статичен хипертекст
 - сървърни технологии като PHP, Ruby, Perl, Python, JSP, and ASP.NET

Уеб 2.0 (социален Уеб) 1/3

- **Web 2.0:**

- ✓ термин за втория етап от еволюцията на World Wide Web;
- ✓ от началото на века до наши дни;
- ✓ въведен от Tim O'Reilly на Web 2.0 conference през 2004

- **Бизнес модел:**

- ✓ top-down + bottom-up подход за изграждане и използване на WWW - динамични хипертекст страници с авторско съдържание и на самите потребители;
- ✓ добавена стойност от споделянето на информация и сътрудничеството между организации и хора;
- ✓ фокус върху създаването на съдържание и персонализирана презентация

Уеб 2.0 (социален Уеб) 2/3

- **Технически характеристики:**

- ✓ динамичен хипертекст
- ✓ съдържание от външни потребители
- ✓ модел "Network as platform" - потребителски интерфейси за достъп до разл. услуги като напр. публични сайтове с галерии на потребителя (Flickr, Picasa Web Albums, ...), частни и споделени хранилища за данни (DropBox), споделени документи (Google Docs), представяне на геогр. обекти върху карта (Google Mail API), ...
- ✓ оперативен обмен на данни (interoperability)
- ✓ Rich Internet Application (RIA)

Уеб 2.0 (социален Уеб) 3/3

- **Нови технологии:**

- ✓ клиентски (client-side/web browser):

- XML или JSON (JavaScript Object Notation)
 - asynchronous JavaScript (Ajax)
 - Adobe Flash
 - Adobe Flex
 - JavaScript/Ajax frameworks като jQuery
 - HTML5 - изиска по-малко изчислителни ресурси отколкото Adobe's Flash; по-малко ел. мощност (батерия при мобилни устройства); замразяване на публичните мобилни Adobe's Flash приставки (plugins)

- ✓ сървърни

Социални феномени в Уеб 2.0

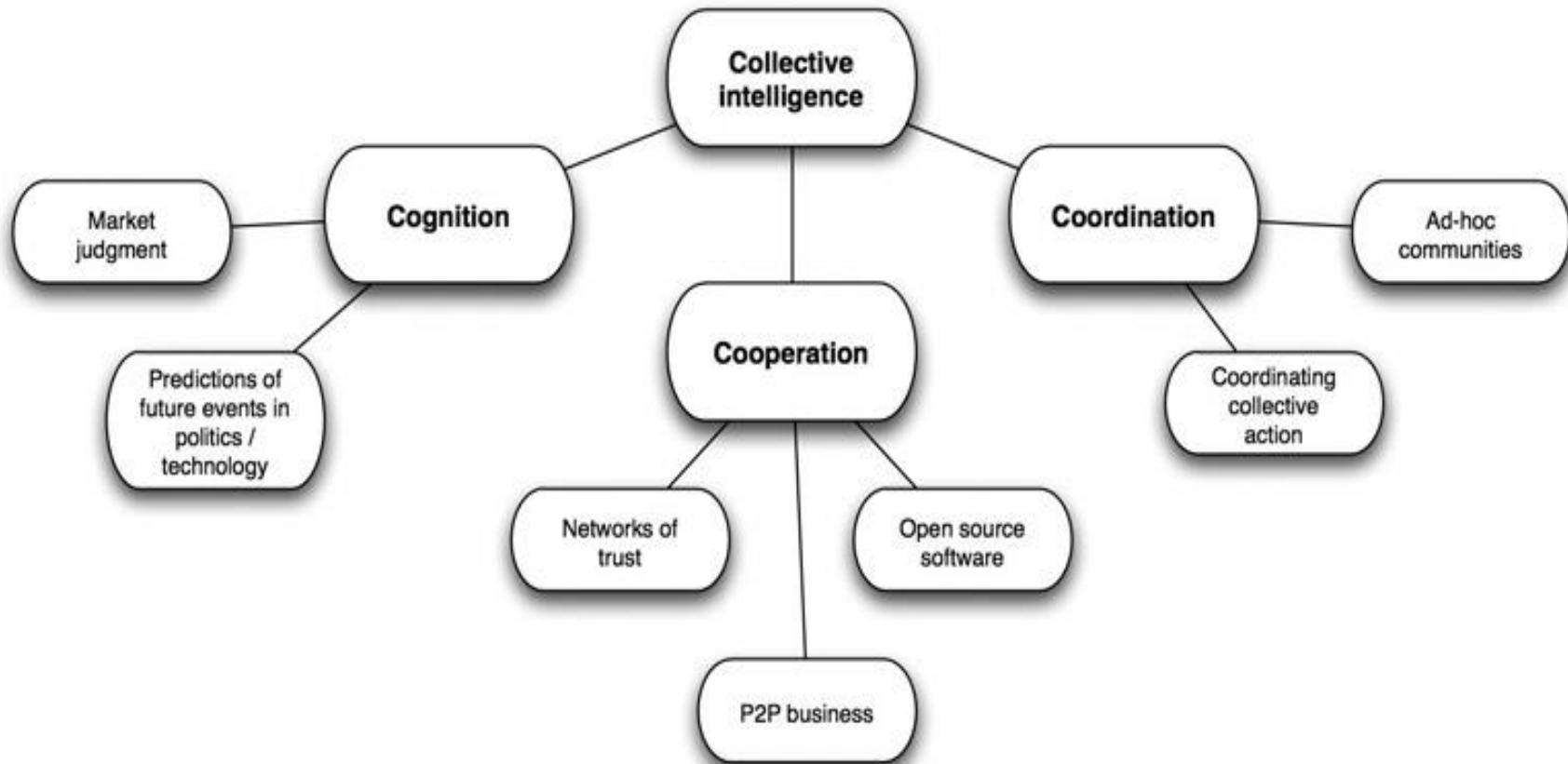
- **Podcasting**
- **Blogging**
- **Tagging**
- **Folksonomy**
- **Social bookmarking**
- **Social networking**

Опишете ги...

Социални феномени в Уеб 2.0

- **Podcasting** - от broadcast и (i)Pod - сваляне на онлайн видео или аудио съдържание от настолни или мобилни компютри
- **Blogging** - web log - поддържане на личен журнал, публикуван в Уеб на дискретни порции (т.нар. posts), показвани в ред, обратен на хронологичния
- **Tagging** - добавяне на метаданни (описания с ключови думи и термини) към съдържание или части от него, с цел да се ползват при търсене и разглеждане (browsing)
- **Folksonomy** (social tagging) - много потребители добавят метаданни като кл. думи към споделени ресурси - Golder, Scott; Huberman, Bernardo A. (2006). "Usage Patterns of Collaborative Tagging Systems". *Journal of Information Science* 32 (2): 198–208.
- **Social bookmarking** - социални отметки - организиране, поддръжка и търсене на отметки към онлайн ресурси
- **Social networking** - сътрудничество и съревнование в Уеб

Колективна интелигентност



- Известник: Olga Generozova, по книгите 'The wisdom of crowds' и 'Smart mobs'

Уеб 3.0 (семантичен Уеб) 1/2

- **Еволюция и переход от сегашното състояние на Световната мрежа към семантичен Уеб**
- ✓ Семантични услуги – базирани на онтологии за представяне на знанието за дадена предметна област:
 - Семантично анотиране на съдържание
 - Семантично търсене
 - Семантично разглеждане
 - Семантично препоръчване
 - ...

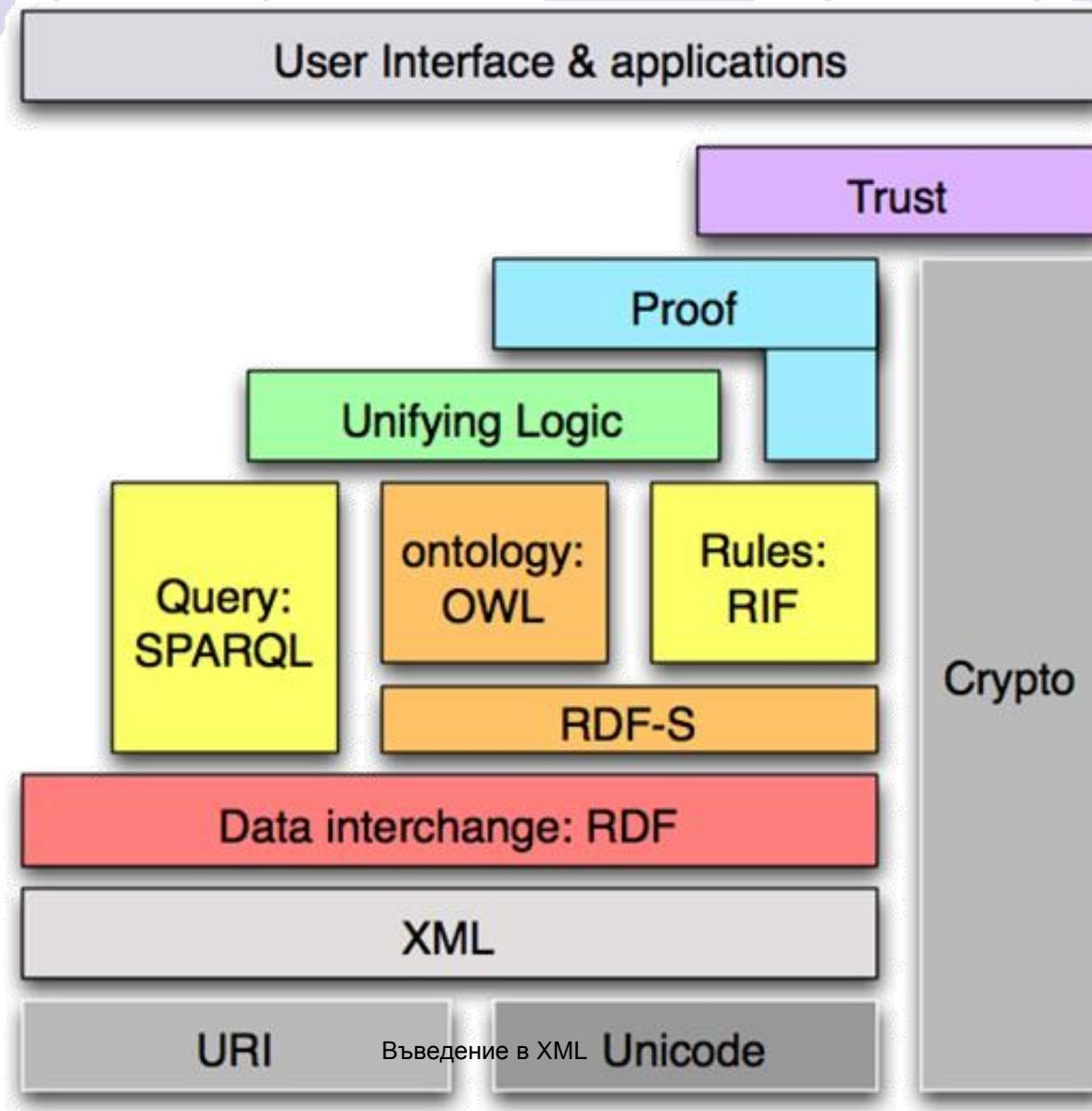
Уеб 3.0 (семантичен Уеб) 2/2

Семантични езици – базирани на XML

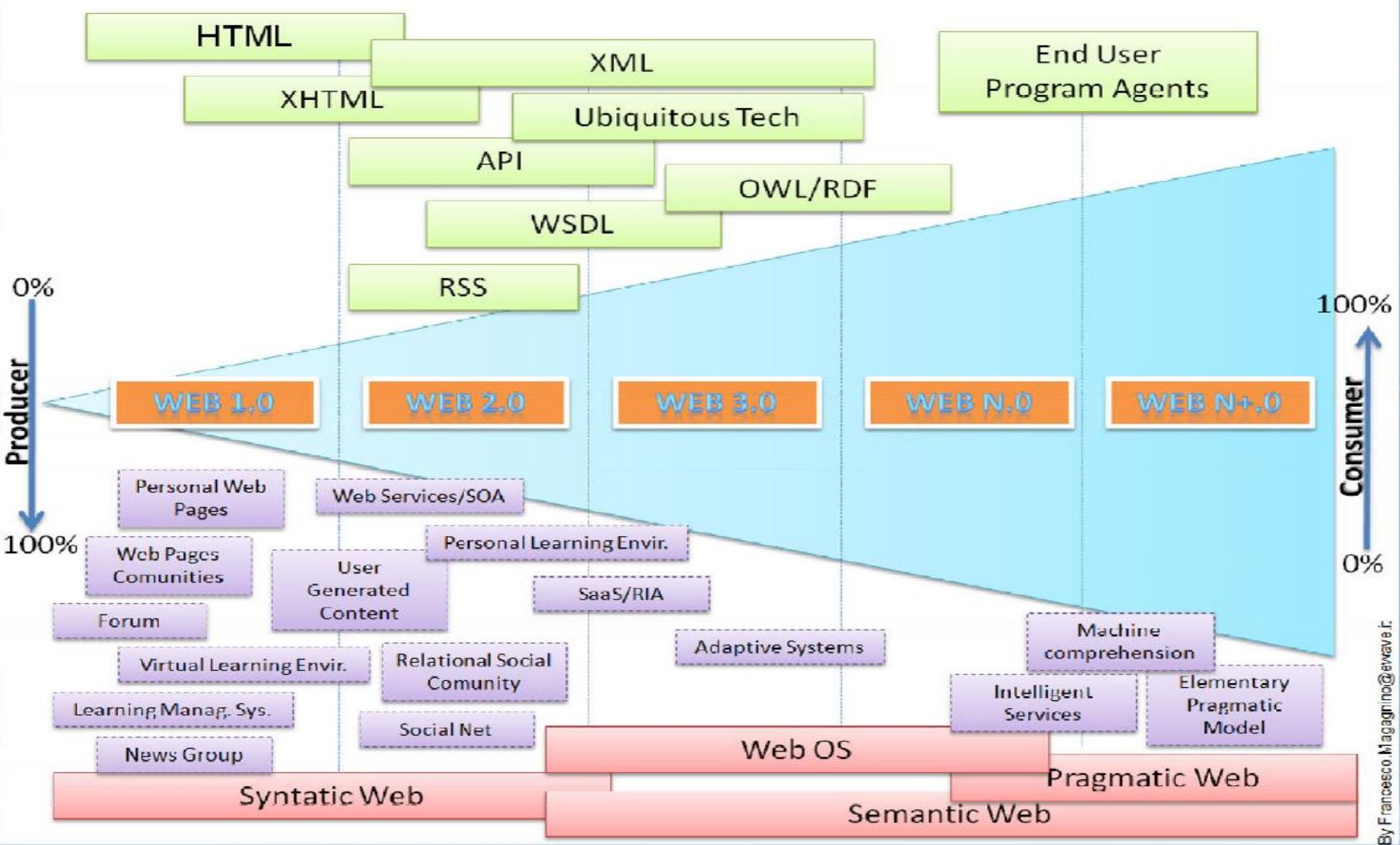
- ✓ Resource Description Framework (RDF) - за описание на модела на метаданните относно Уеб ресурси
- ✓ RDF Schema
- ✓ SPARQL Protocol and RDF Query Language - език за заявки към RDF графи
- ✓ Web Ontology Language (OWL) - фамилия от езици за представяне на знания чрез онтологии
- ✓

Стек на семантичния Уеб (W3C, 2006)

Адаптирано от http://en.wikipedia.org/wiki/Semantic_Web_Stack

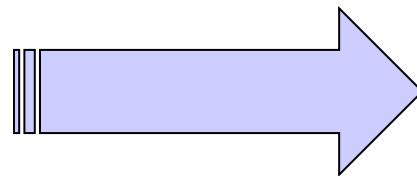


Към Уеб N.0??



Период на переход

- HTML

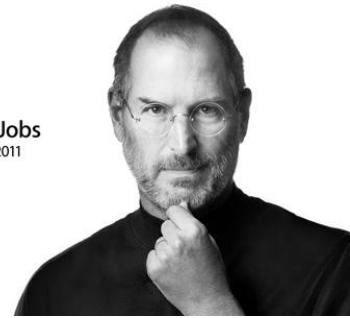


- XML

Струва ли си?

Steve Jobs, 1955-2011

Steve Jobs
1955-2011



- “That’s been one of my mantras — **focus and simplicity. Simple can be harder than complex**: You have to work hard to get your thinking clean to make it simple. But it’s worth it in the end because once you get there, you can move mountains.”

[BusinessWeek, May 25, 1998]

See also:

http://www.ted.com/talks/steve_jobs_how_to_live_before_you_die.html



ML = Markup Language = език за маркиране

- Маркъп езиците (езици за маркиране) възникват на един сравнително късен етап.
- Понятието markup се дефинира от популярния речник Уебстър като подробни инструкции, които обикновено се записват в ръкописа на дадено типографско издание относно типа и стила на шрифта, начина на странициране и други печатни оформления (1915-1920 год.)
- Интуитивен пример за такъв маркъп може да бъдат ръчно записани коментари от коректора на изданието относно правописни грешки или граматически и стилови корекции, включващи изтриване и добавяне на текст, прехвърляне на съдържание на друго място, представяне на думите в курсив, и др.

Езици за маркиране

- Езиците за маркиране (*markup languages*) служат за описание на данни относно структурирането, форматирането, значението и начина за представянето на тези данни.
- Самото маркиране представлява добавяне на допълнителна стойност към основното съдържание – от гледна точка на неговото оформление като структура, значение и представяне.
- По този начин общийят обем на текста се увеличава с допълнителен, маркиращ текст, включен в документа и разграничен от основното съдържание по определен начин.

Черти на езиците за маркиране

- **стилистични** - отнасящи се до външния вид, или представянето на текстовото съдържание - напр. таговете в HTML: **<I> <U>**
- **структурни** - определящи макета на структурата (layout) - напр. таговете в HTML: **<H2> <P>
**
- **семантични** - служещи за определяне на смисъла на съдържанието - напр. таговете в HTML:
<TITLE> <META NAME=keywords CONTENT = " " >
- **функционални** - задаващи определено действие представяне на съдържанието в дадена медия - напр. таговете в HTML:
<BLINK> Щракнете тук</ A>

Компоненти на език за маркиране

- Пример 1: Hello, world!
 - = **start tag**
 - = **end tag**
 - Hello, world! = **content (or data)**
 - < > = **delimiter characters (identify tags)**
- Hello = **element**
 - element = **tags + content**
- Пример 2:
 - name/value pairs identify **attributes** and give **values**

Какво представлява XML?

- XML 1.0 е препоръка на W3C (World Wide Web Consortium: <http://www.w3.org>) от 10.02.1998
- XML наследява Standard Generalized Markup Language (SGML) - през 1986 год. SGML е утвърден като стандарт ISO 8879 и се налага в световен мащаб като език за структурно, стилистично и семантично маркиране на текстови документи.
- Специално за стилистичното и и семантичното маркиране към него е разработен стиловия (stylesheet) език DSSSL (Document Style Semantics and Specification Language).
- SGML е много тежък и труден за употреба

XML е метаезик

- XML, както и SGML, е *метаезик* за маркиране.
- Това е разширяем език, който позволява на потребителя да създава таговете (resp. инструкциите за маркиране), които са му необходими за дадена предметна област, и да ги използва заедно.

HTML

- ❖ HTML е приложение на SGML и поради това е подобен на него, но е значително по-опростен и ограничен.
- ❖ За разлика от SGML, HTML не е разширяем.
- ❖ HTML 4.01 спецификацията определя 91 елемента за използване в HTML документи, като нови елементи не могат да бъдат създавани от потребителя.
- ❖ Крайните тагове могат да бъдат пропуснати за 15 от дефинираните от спецификацията елементи, което показва друга отличителна черта на HTML в сравнение с SGML - липсата на строги синтактични правила.
- ❖ Езикът е прост, лесен за научаване и използване за гъвкаво представяне на съдържание в Уеб браузер.

Недостатъци на HTML при използването му в Уеб 1/2

- HTML не е разширяем
- HTML не позволява представянето на отношения (релации) между елементи и в такъв смисъл има "плосък" синтаксис. HTML представя само двуизмерни таблици, а многомерни таблици трябва да се изобразяват по отделните измерения. Също така, няма как да бъдат представени релациите
- **name = име + фамилия**
- **client = име + адрес (улица + номер + код и др.)**
- Не е възможно да се представи ефективно структурата на документа, макар че таговете за стил като `<H1>`, `<H2>` и др. могат да се тълкуват като структурни тагове, което е обаче крайно недостатъчно.

Недостатъци на HTML при използването му в Уеб 2/2

- HTML притежава много малко възможности за описание на значението на съдържанието. Използването на елементи като
<TITLE> и <META NAME="keywords" CONTENT="metadata ..." >
се оказва крайно недостатъчно.
- Повечето тагове на HTML имат чисто процедурен характер – те описват какво трябва да направи браузъра при представяне на съдържанието, напр. **
** за пренасяне на нов ред, **<hr>** за извеждане на хоризонтална линия, **<i>** за показване на текста в курсив.
- Това определя HTML като процедурен, а не като **дескриптивен език за маркиране.**

XML

- През 1996 г. Джон Босак, Тим Брей, Джеймс Кларк и няколко други Уеб специалисти започват работа по създаването на олекотена версия на SGML
- W3C утвърждава разработката на този език, който не притежава както недостатъците на както на SGML, така и на HTML, през февруари 1998 г. под името:
XML (Extensible Markup Language) 1.0
- През последните години развитието и използването на XML се очерта като най-перспективното направления за развитие на Уеб. XML е основният двигател на процесите на прерастването на съвременния Уеб в Семантична мрежа.

Предимства на XML

- XML е разширяем мета език за дескриптивно маркиране
- Валидиране на типа на документа - XML и всички XML-базирани езици са формални и имат формални граматики
- Оперативен обмен на данни.



The screenshot shows a Microsoft Internet Explorer window with the title bar "C:\Documents and Settings\Boyan\My Document...". The address bar displays "Lbook\examples\example1.xml". The main content area shows an XML document structure:

```
- <client>
  + <name>
  - <address>
    <street>Oborishte St.</street>
    <streetNo>57</streetNo>
    <city>Sofia</city>
    <postCode>Oborishte St.</postCode>
  </address>
</client>
```

XML спрямо HTML

Database publishing

Integration of data from different sources

Extended usability of metadata

Multi-directional and –dimensional links

Reusability of information

XML йерархии

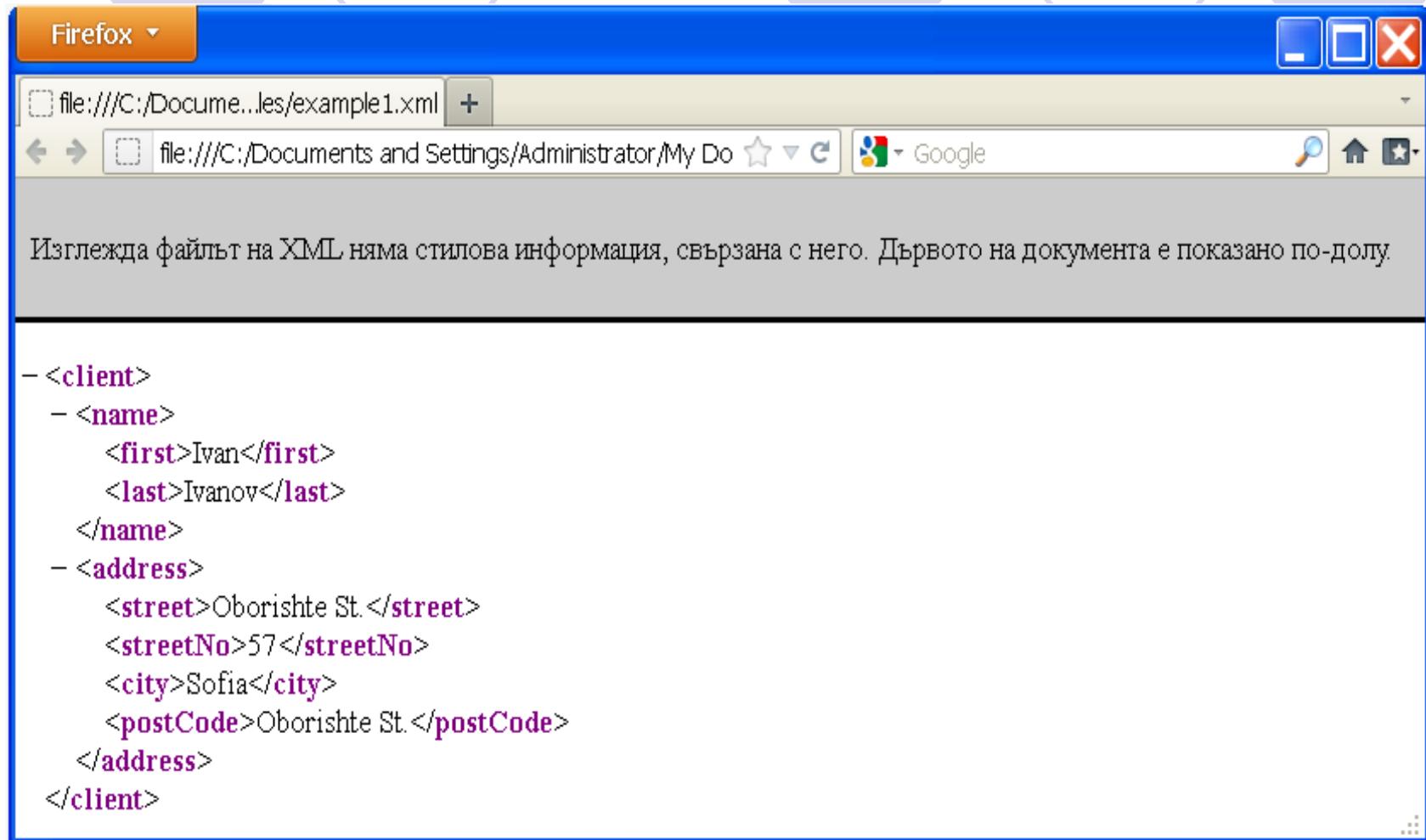
- **textbooks** (root element)
 - **book** (repeating element)
 - **title**
 - **ISBN**
 - **authors**
 - **author_name** (repeating element)
 - **description**
 - **price**

Създаване на XML документи

- Подобно на HTML, но без ограничения за елементите и атрибутите им:

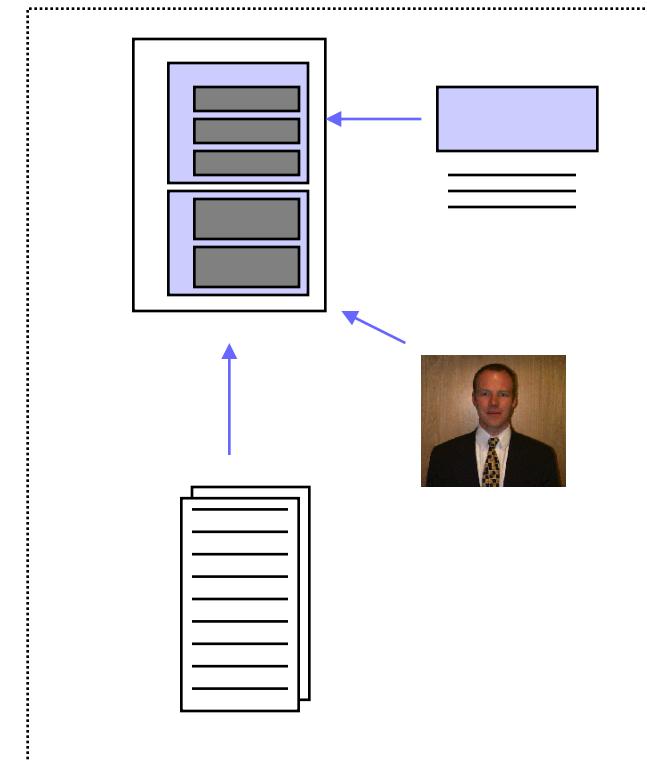
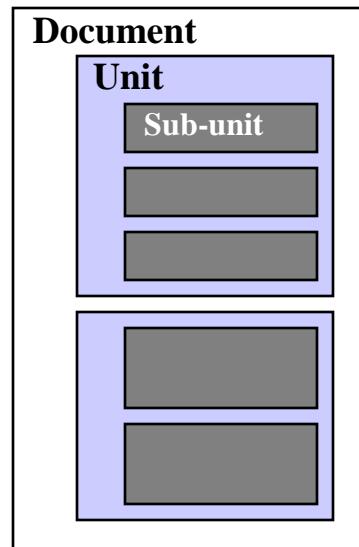
```
<?xml version="1.0"?><!--required-->
<books><!--root element-->
  <textbooks>
    <btitle>Beginning ASP 3.0</btitle>
    <!--other items-->
  </textbooks>
</books>
```

XML в браузър (без стилове)



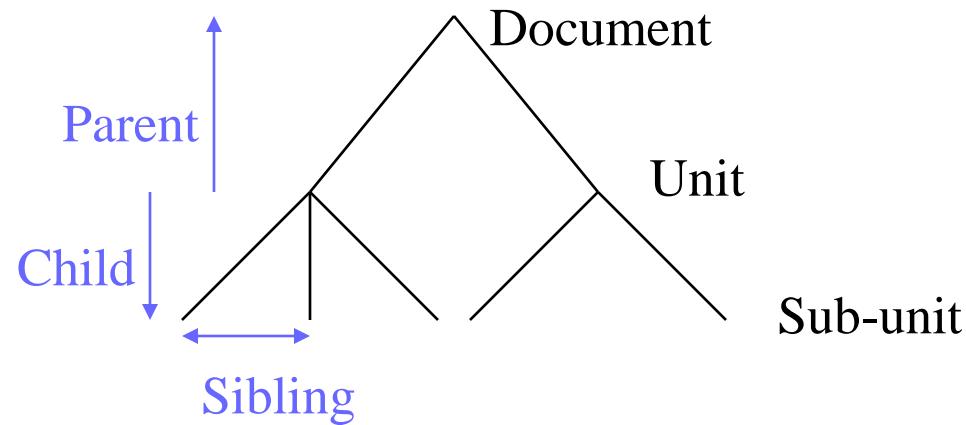
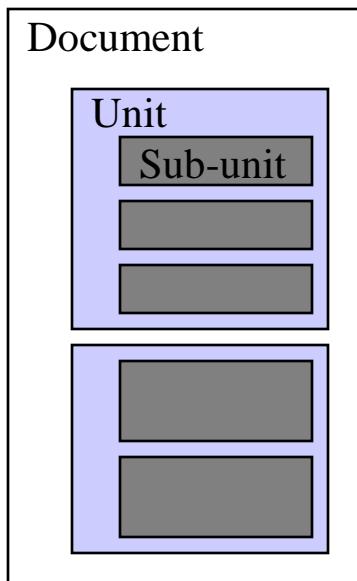
Структура на XML документ

- Логическа структура
 - Елементи
- Физическа структура
 - Единици (Entities)



XML йерархия

Планарно и дърводидно представяне



N.B. All elements must be nested

Средства за работа с XML

- **Минимум**
 - Text editor and XML parser (e.g. msxml-Parser by Microsoft)
 - XML-Browser (XML, XSLT) (e.g. Internet Explorer)
- **Повече**
 - XML-Editor (e.g. XML Spy by Altova)
(<http://www.xmlspy.com>)
 - XML-Browser with XSLT (IE 6)
 - XSL:Fo Formatter (Antenna by Antennahouse)
(<http://www.antennahouse.com>)

XML технологии

DTD

- XML Schema
- XLink and XPointer
- XSL and XSLT/XSL-FO
- XQL
- XPath
- Namespaces
- SAX
- DOM
- StAX
- JAXB
- Други ... ☺



Добре-структурен XML (Well-Formed XML)

XML синтаксис
Йерархии
Елементи и атрибути
Правила
XML кодиране

Основни характеристики на документ

Съдържание

- Информация, съдържаща текст, графика, аудио или видео

Структура

- Подялба и последователност на информационните части

Оформление

- Онагледяване на съдържанието и структурата на документ

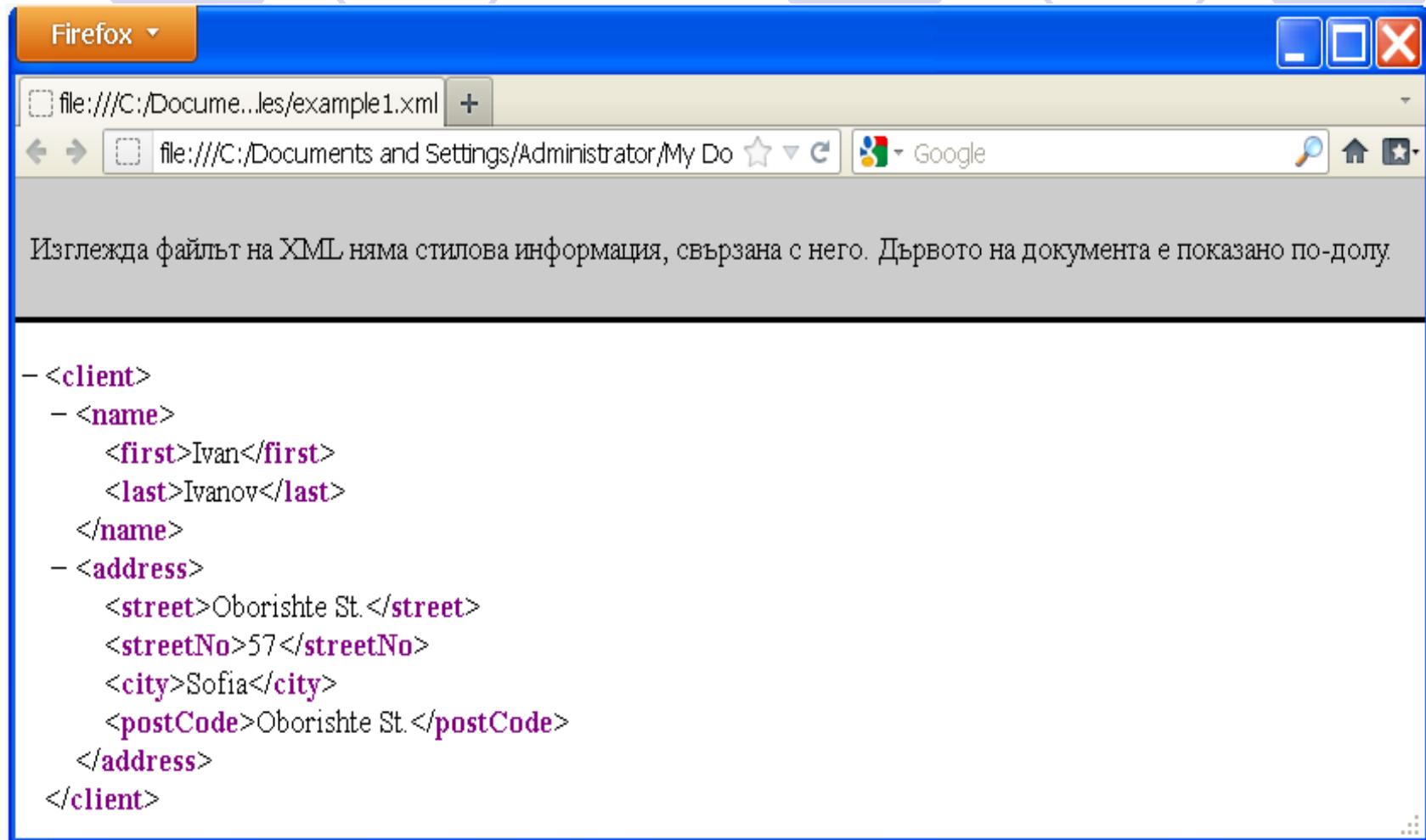
Метаданни

- Описание семантиката на съдържанието

Черти на езиците за маркиране

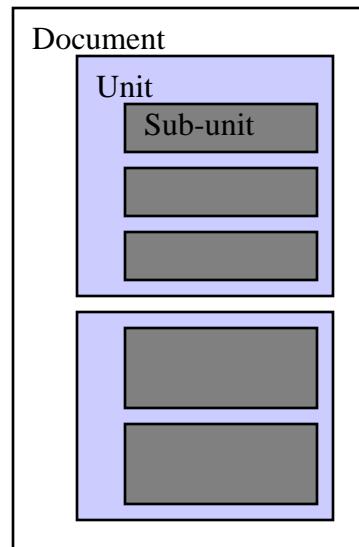
- **Стилистични** (appearance) – напр. в HTML:
`<I><U>`
- **Структурни** (layout) - напр. в HTML:
`<P>
<H2>`
- **Семантични** (meaning) - напр. в HTML:
`<TITLE><META NAME="keywords" CONTENT = ".....">`
- **Функционални** (action) - напр. в HTML:
`<BLINK>`
`Click here`

XML в браузър (без стилове)

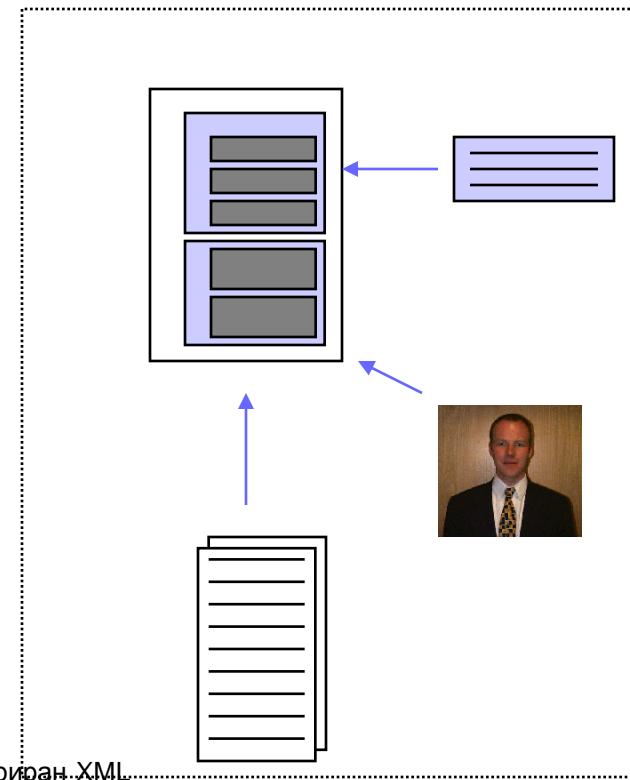


Структура на XML документ

- Логическа структура
 - Елементи
- Физическа структура
 - Единици (Entities)



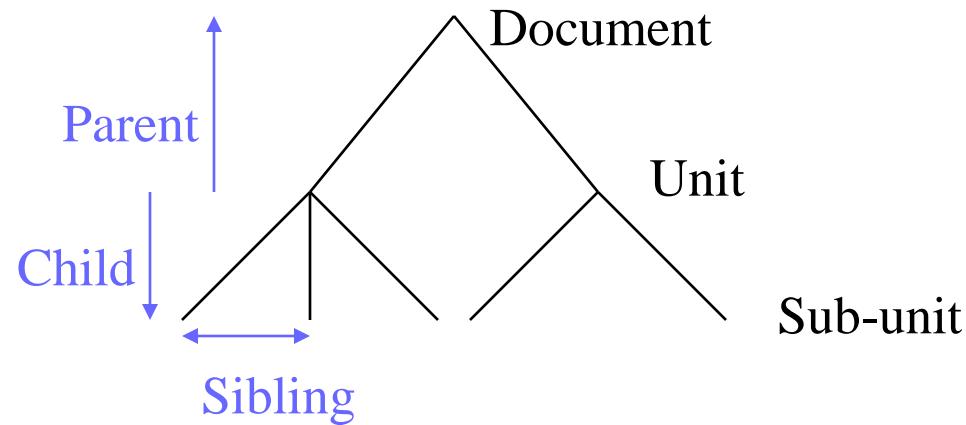
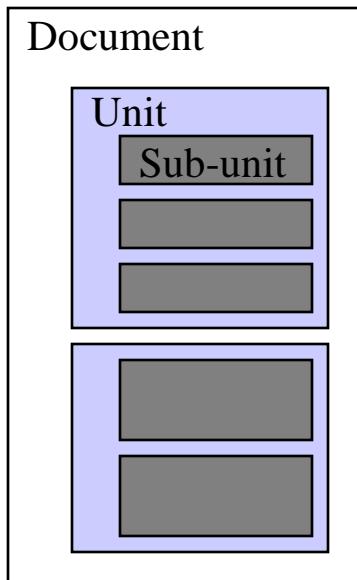
XML



Добре-структурiran XML

XML йерархия

Планарно и дърводидно представяне



N.B. All elements must be nested

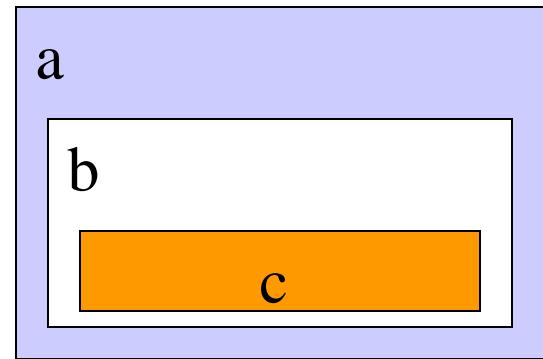
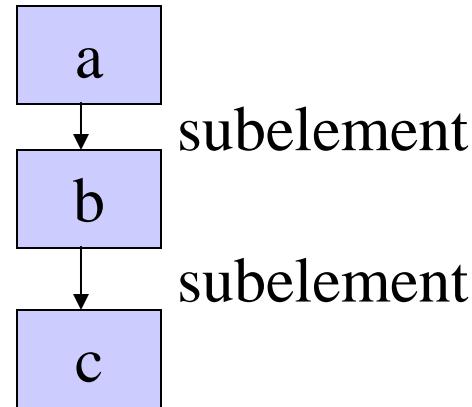
XML синтаксис

- Маркер (таг):
 - Име на елемент, разделено с **< и >**.
 - *Start-Tag* **<elementname>**
 - *End-Tag* **</elementname>**
- Елемент:
 - **<elementname> Content of the element </elementname>**
 - Типът на съдържанието е Parsed Characted DATA (PCDATA)
 - Само един елемент-корен
 - Влагане на под-елементи
 - **<author>**
<first>Charles</first>
<last>Gottlieb</last>
</author>

XML – документна йерархия 1/2

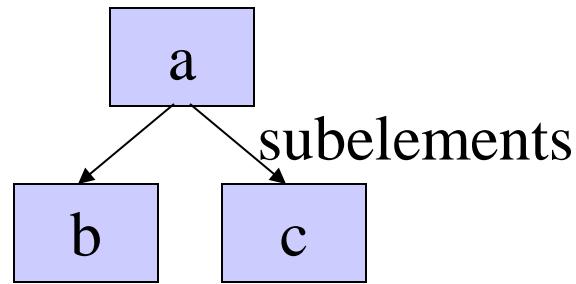


- <a>
-
- <c> </c>
-
-



XML – документна йерархия 2/2

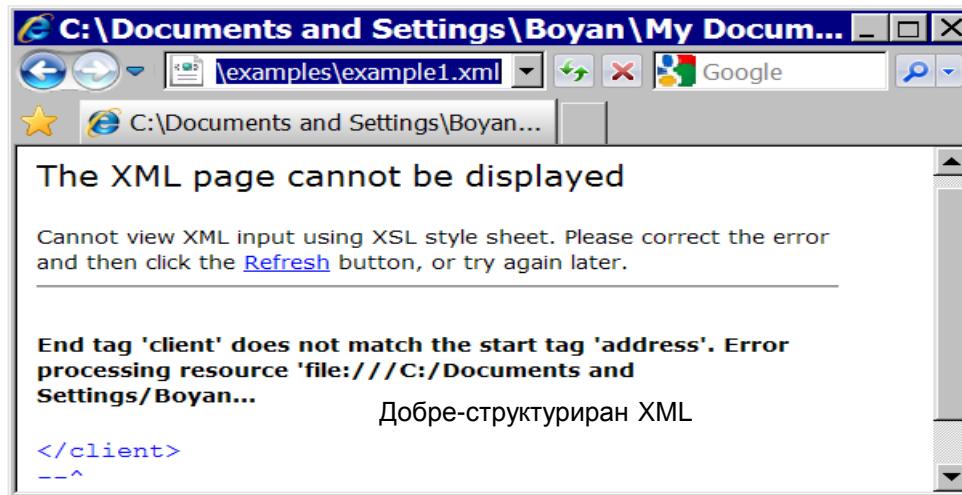
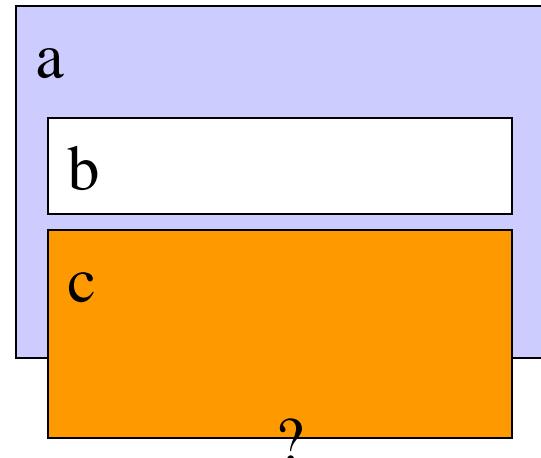
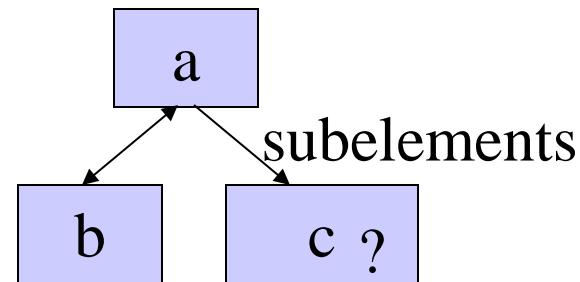
- <a>
- ...
-
- <c>...
- </c>
-



XML – липса на затварящ маркер

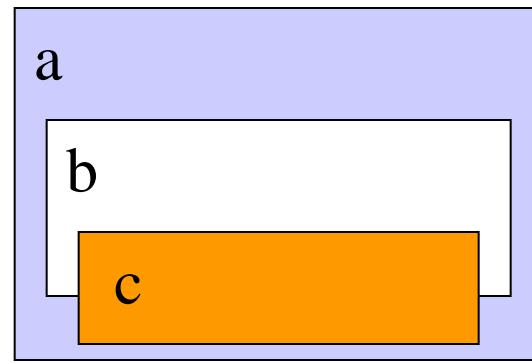
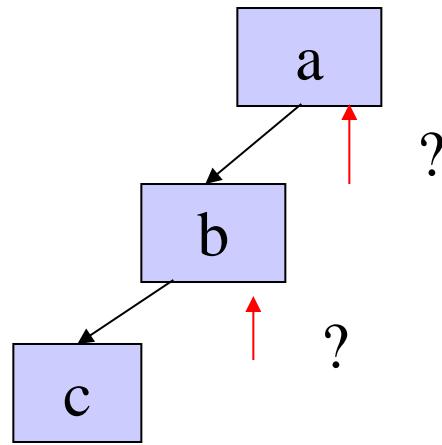
Error: missing C closing element tag

- <a>
-
-
- <C>
-

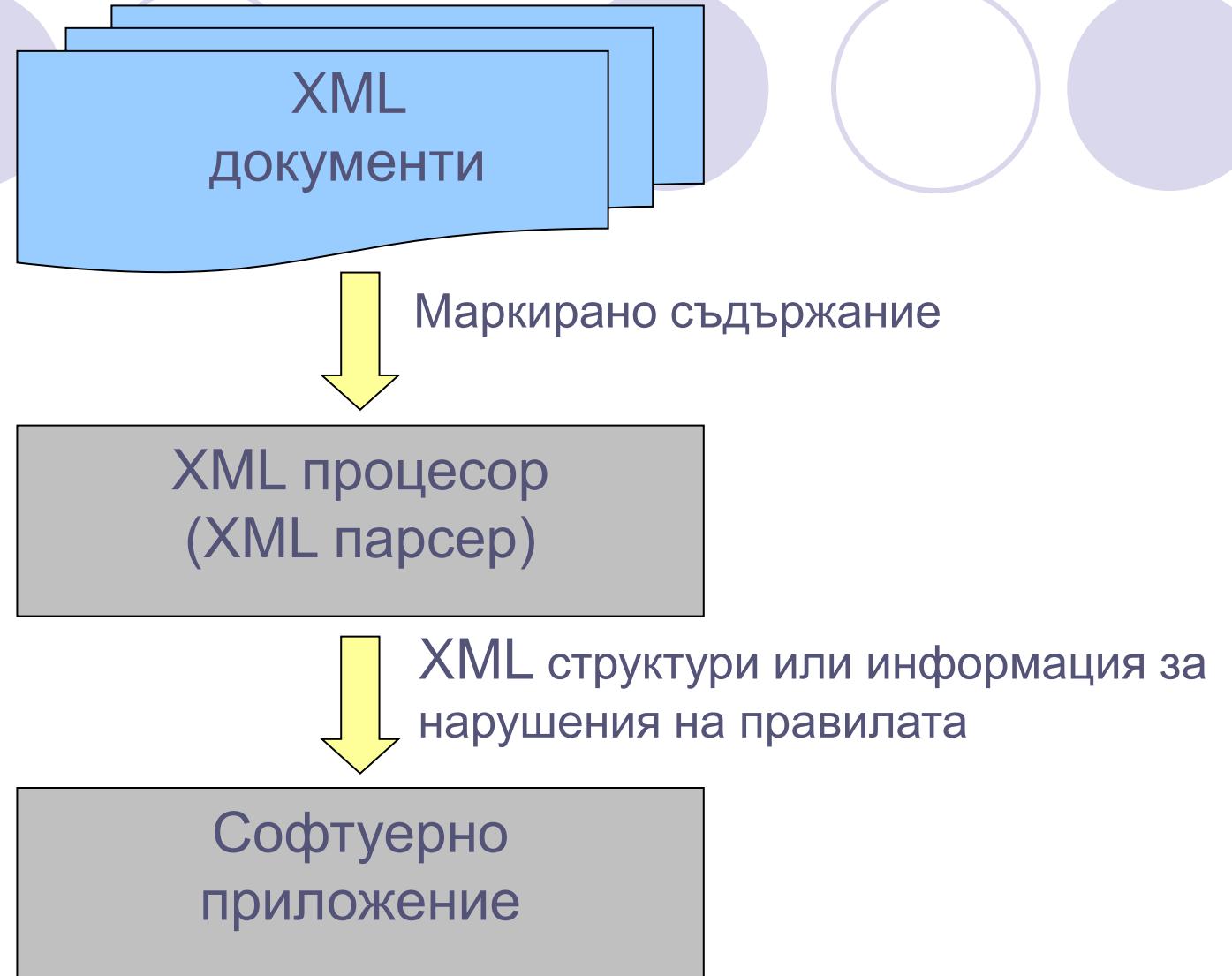


XML- припокриване на елементи

- <a>
-
- <c>
-
- </c>
-



Error: broken elements -
with an *overlap* between the b and c element



Важно:

XML и всички XML-базирани езици са формални и имат формални граматики.

```
<person>
  <name>
    <first>Ganyo</first>
    <last>Balkanski</last>
  </name >
  <title>Bai</title>
  <alias>New European</alias>
  <birth>
    <place>any place</place>
    <time>any time</time>
  </birth>
  <character>
    <physical>
      slim, tall, cheery, attractive, too clean
    </physical>
    <personal>
      dedicated, honest, philanthropist, leader, idealist
    </personal>
  </character>
  <address>anywhere in Bulgaria</address>
  <professions>
    <profession>image maker</profession>
    <profession>politician</profession>
  </professions>
</person>
```

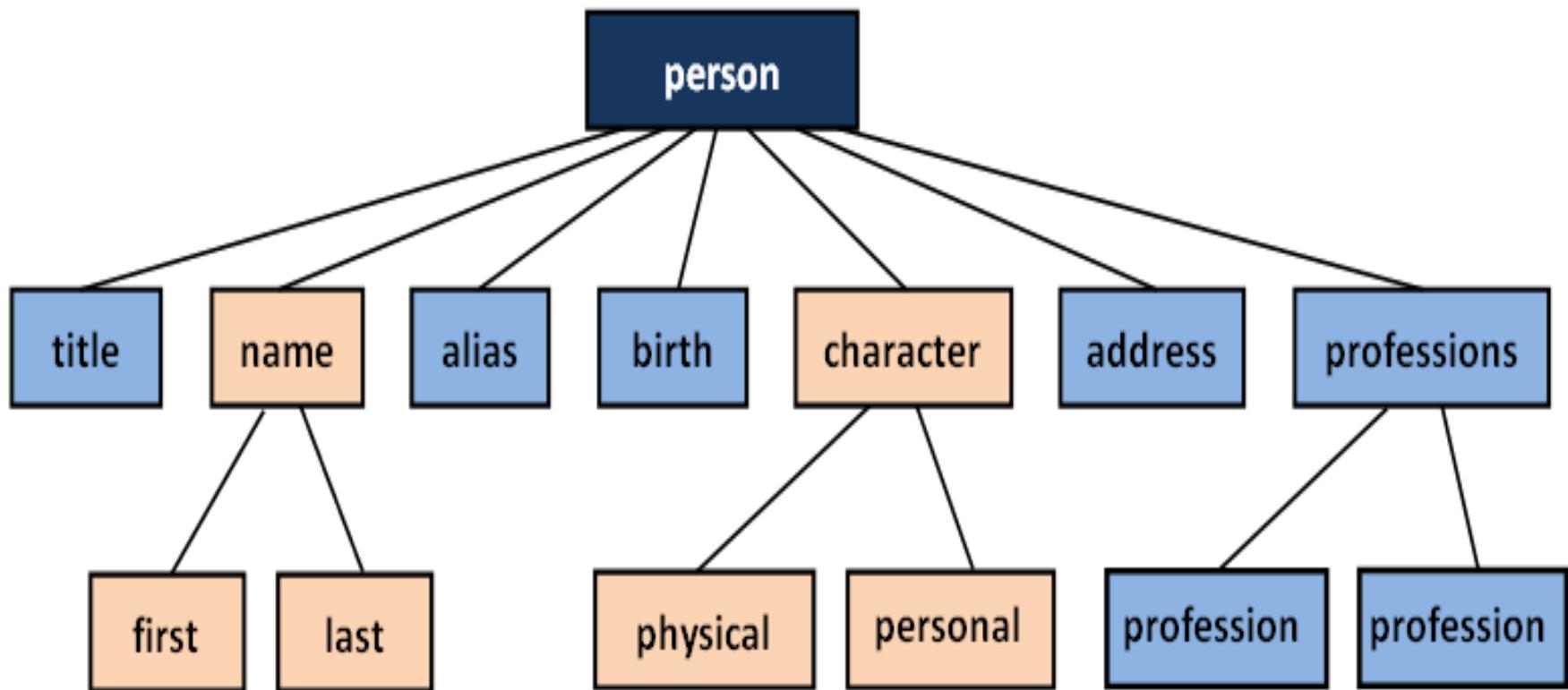
XML

Добре-структурен XML

XML – актуален пример

XML дърво

N.B.: Only one root element!



Видове XML елементи 1/3

- Елементи с текстово съдържание - съдържанието представлява текст, ограден от начален и краен таг на XML елемент.
- По исторически причини, наследени от SGML, такъв тип елементно съдържание се нарича ***parsed character data***, или съкратено ***PCDATA***.
- Празните пространства (whitespace) се задават в текстовото съдържание посредством интервал (space), нов ред (чрез клавиша Enter) или табулация (Tab)

Видове XML елементи 2/3

- Елементи със структурно съдържание представляват такива XML елементи, в които текстовото съдържание е структурирано под формата на вложени елементи,
- организирани в дървовидна йерархия.

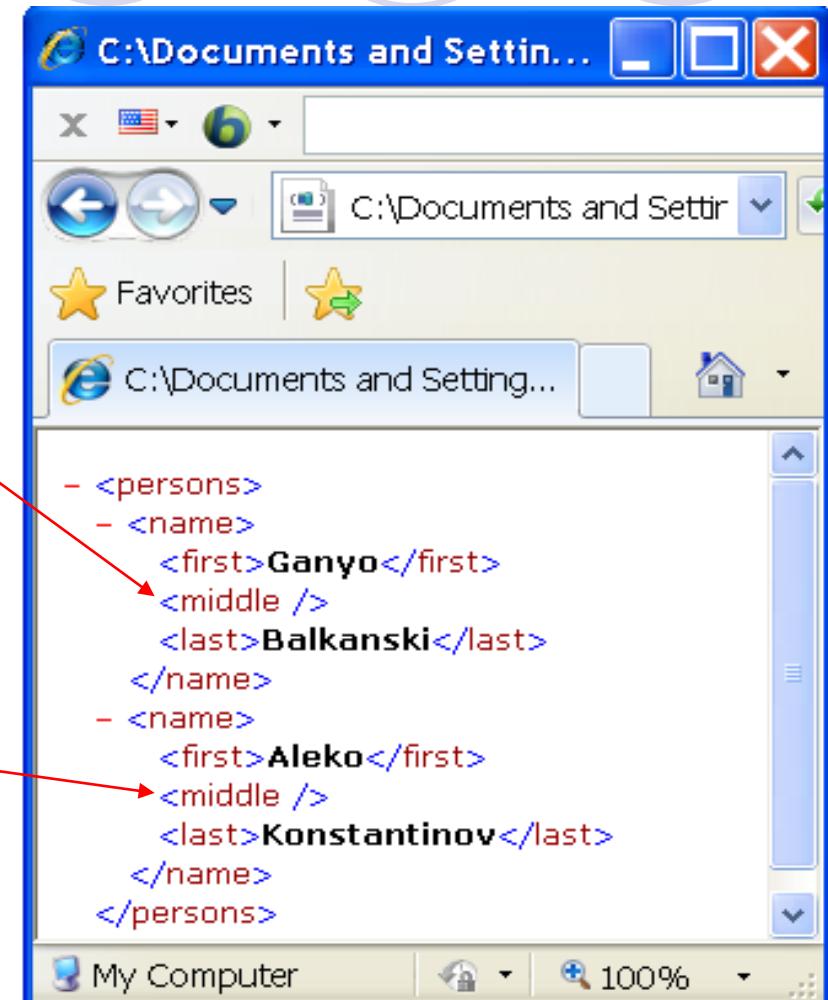
Видове XML елементи 3/3

Понякога елементите нямат нито PCDATA, нито структурно съдържание. Такива елементи се наричат **празни** и могат да се означат по два начина в XML документите:

- С начален и краен таг както всички останали елементи – такъв е първият елемент `<middle>` в примера по-долу;
- Посредством само-затварящ се (self-closing) таг, какъвто е вторият елемент `<middle>` в примера на следващата страница:

Пример за само-затварящ се (self-closing) таг

- <persons>
- <name >
- <first>Ganyo</first>
- **<middle></middle>**
- <last>Balkanski</last>
- </name>
- <name >
- <first>Aleko</first>
- **<middle />**
- <last>Konstantinov</last>
- </name>
- </persons>



Структура на XML документа

Пролог

```
<?xml version="1.0" encoding="UTF-8"  
standalone="yes"?>  
<?xml:stylesheet type="text/css" href="s.css"?>  
<!DOCTYPE test SYSTEM "test.dtd">  
•Декларации на документ, стилове и тип на документа
```

Екземпляр

- Елементна йерархия

```
<author>  
    <first>Ganyo</first>  
    <last>Balkanski</last>  
</author>
```

Допълнения

- Коментари, CDATA секции и инструкции за обработка

```
<!-- comment -->  
<?myApp status="draft"?>  
<![CDATA[ Press <<<ENTER>>> ]]>
```

Синтактични правила за елементи 1/2

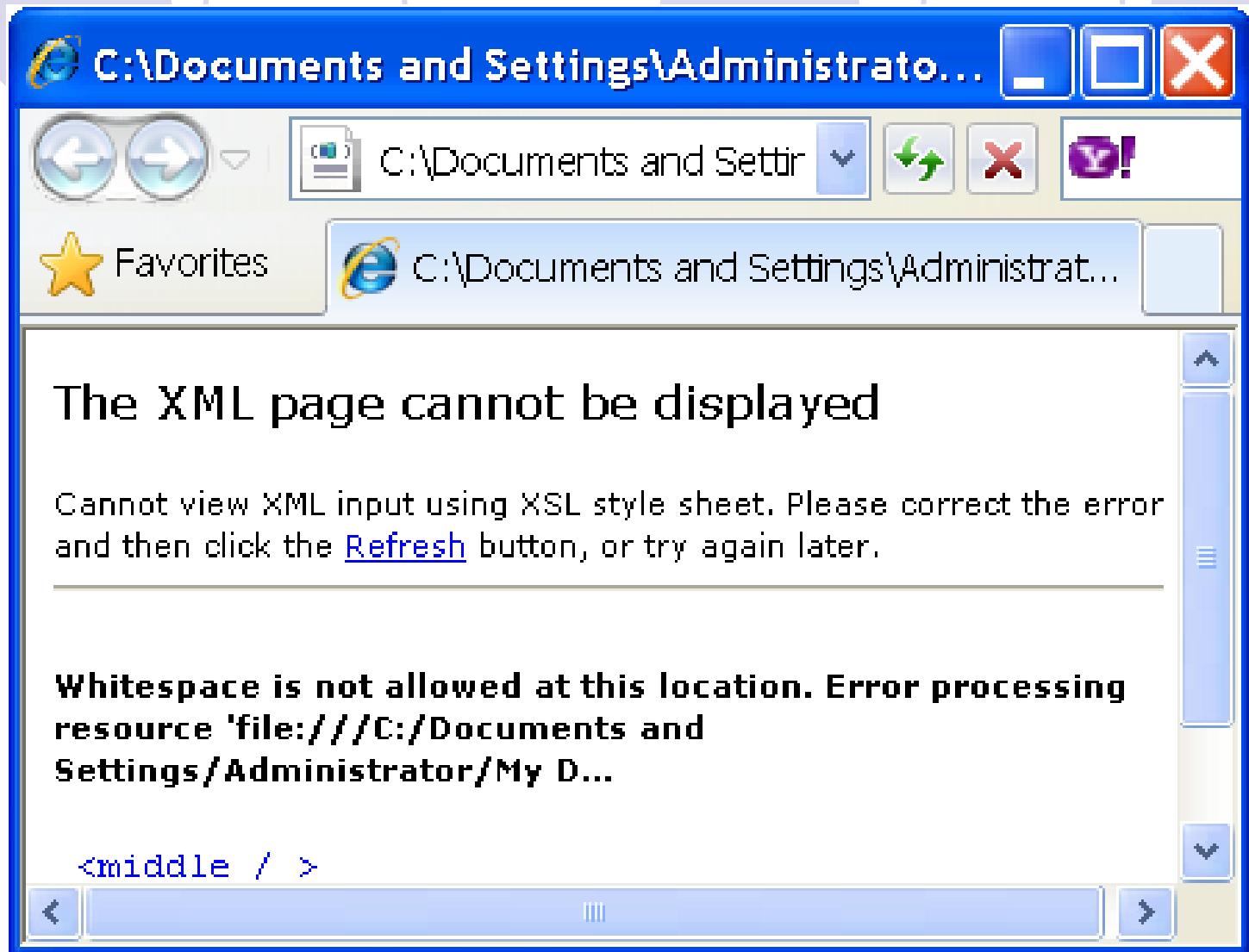
- Имената в XML са чувствителни към регистъра, т.е. XML парсерът прави разлика между използването на малки и големи букви в имената (case-sensitive) – затова `<person>`, `<Person>`, `<PerSon>` и `<PERSON>` са различни начални тагове;
- Имената трябва да започват с малка или главна буква или ‘_’;
- По-нататък, освен букви и ‘_’, в имената могат да бъдат използвани цифри, ‘-’ и ‘.’;
- Имената не могат да започват с думите XML, xml, Xml, xmL и т.н.;
- Празни пространства се допускат само след имената на елементите, т.е. преди знака ‘>’;

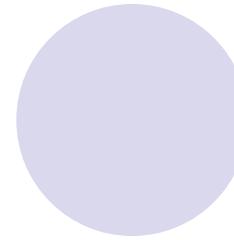
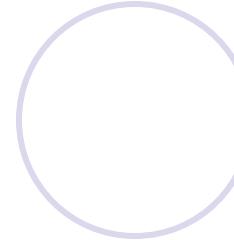
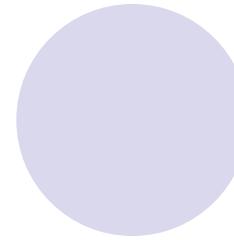
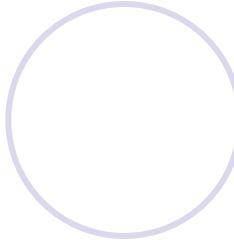
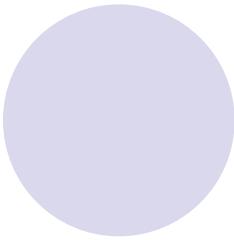
Синтактични правила за елементи 2/2

- Използването в имената на символа за двоеточие ‘:’ е разрешено, резервирано – по-точно, знакът ‘:’, използван в име на таг, отделя име на пространство от имена от локалното име на тага.
- В XML имената не могат да бъдат използвани други символи освен букви (вкл. и от азбуки, различни от латинската), цифри, празно пространство, ‘_’, ‘-’, ‘:’ и ‘;’;
- В XML имената празното пространство може да бъде интервал, нов ред или табулация, и е допустимо да се зададе само преди затварящия разделител '>';
- използването на символите ‘<’ и ‘>’ е резервирано за разделители;
- символите ‘&’ и ‘;’ се използват за указване на рефериране към съдържание посредством единица (entity) - наричана още семантична единица или същност.

Примерни XML имена на тагове

Valid names	Invalid names
<BobSun>	<Bob Sun>
<BobSun2003>	<2003BobSun>
<Bob-Sun>	<xm Tag>
<Bob.Sun>	<my=tag>
<rsume>	<a&b>
</middle >	</ middle>
<i>Question: Why correct?</i>	<i>Question: Why incorrect?</i>





Важно:

Поради чувствителността на XML към регистъра, в един XML документ е допустимо да се използват елементи с различно значение, имащи за начални тагове примерно `<title>` и `<Title>`, но това може да представлява причина за объркване или двусмислие.

Празните пространства (whitespace) се задават в текстовото съдържание посредством интервал (space), нов ред (чрез клавиша *Enter*) или табулация (Tab). Докато при маркиращи езици като HTML и WML поредица от празни пространства се запазва като едно (освен ако не са зададени като), то при XML празните пространства се запазват в PCDATA елементите така, както са въведени от създалеля на документа.

Така например елементът

`<person>Aleko Konstantinov</person>`

ще има различно съдържание от елемента

`<person>Aleko Konstantinov</person>`

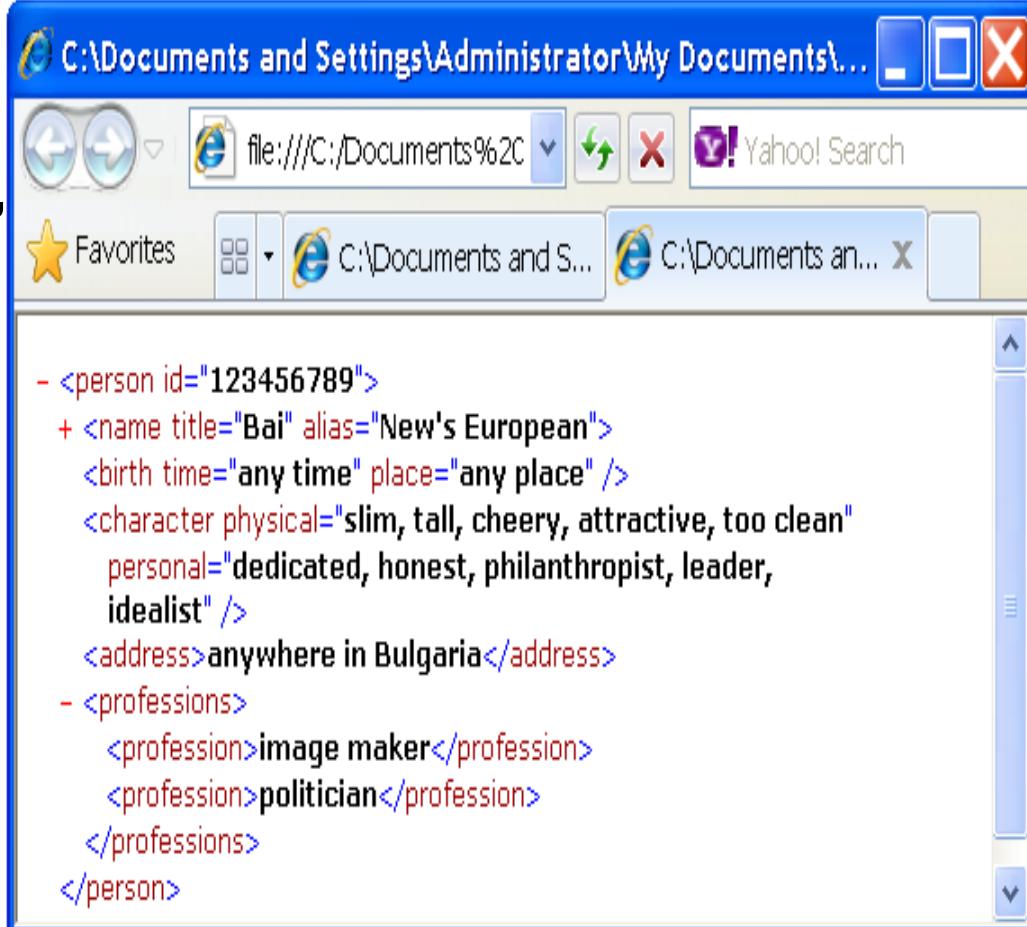
Важно: XML запазва празните пространства.

Синтактични правила за атрибути 1/3

- Всеки атрибут трябва да има стойност, дори и тя да бъде празен стринг;
- Атрибутът **id="123456789"** е добавен към элемента **<person>**, защото той характеризира като идентификатор целият физически индивид, а не само отделни елементи от описанието му;
- За разделител между два и повече атрибути се използва празно пространство, а не запетая;
- Възможно е едновременното използване на кавички и апострофи за ограждане на стойностите на атрибутите на един елемент, както е направено за елемента **<name>**. *Не е допустимо* обаче ограждане на стойност на един атрибут с кавички и апостроф или обратно, например **id="123456789"**.

Синтактични правила за атрибути 2/3

- Независимо от начина на ограждане на стойностите на атрибутите с документа, те се показват в браузер като **оградени с кавички**
- При сгъването в браузер на йерархията на елемент със структурно съдържание, атрибутите продължават да се показват (елемент **<name title='Bai' alias="New's European">** на фигураната)



The screenshot shows a Microsoft Internet Explorer window displaying XML code. The address bar shows 'file:///C:/Documents%20'. The window title is 'C:\Documents and Settings\Administrator\My Documents...'. The toolbar includes Back, Forward, Stop, Refresh, Favorites, and Search buttons. The search bar has 'Yahoo! Search'. Below the toolbar, there are two tabs: 'C:\Documents and S...' and 'C:\Documents an...'. The main content area displays the following XML code:

```
- <person id="123456789">
+ <name title="Bai" alias="New's European">
  <birth time="any time" place="any place" />
  <character physical="slim, tall, cheery, attractive, too clean"
    personal="dedicated, honest, philanthropist, leader,
    idealist" />
  <address>anywhere in Bulgaria</address>
- <professions>
  <profession>image maker</profession>
  <profession>politician</profession>
</professions>
</person>
```

Синтактични правила за атрибути 3/3

- Атрибутите могат да следват произволен ред на подреждане в елемента.
- Това е решено да бъде така, защото те описват характеристики на елемента, които не са структурни и нямат релация за подреждане.

Коментари

- Не са видими за форматираната версия
 - `<!-- this is a comment -->`
- Не се обработват от парсера

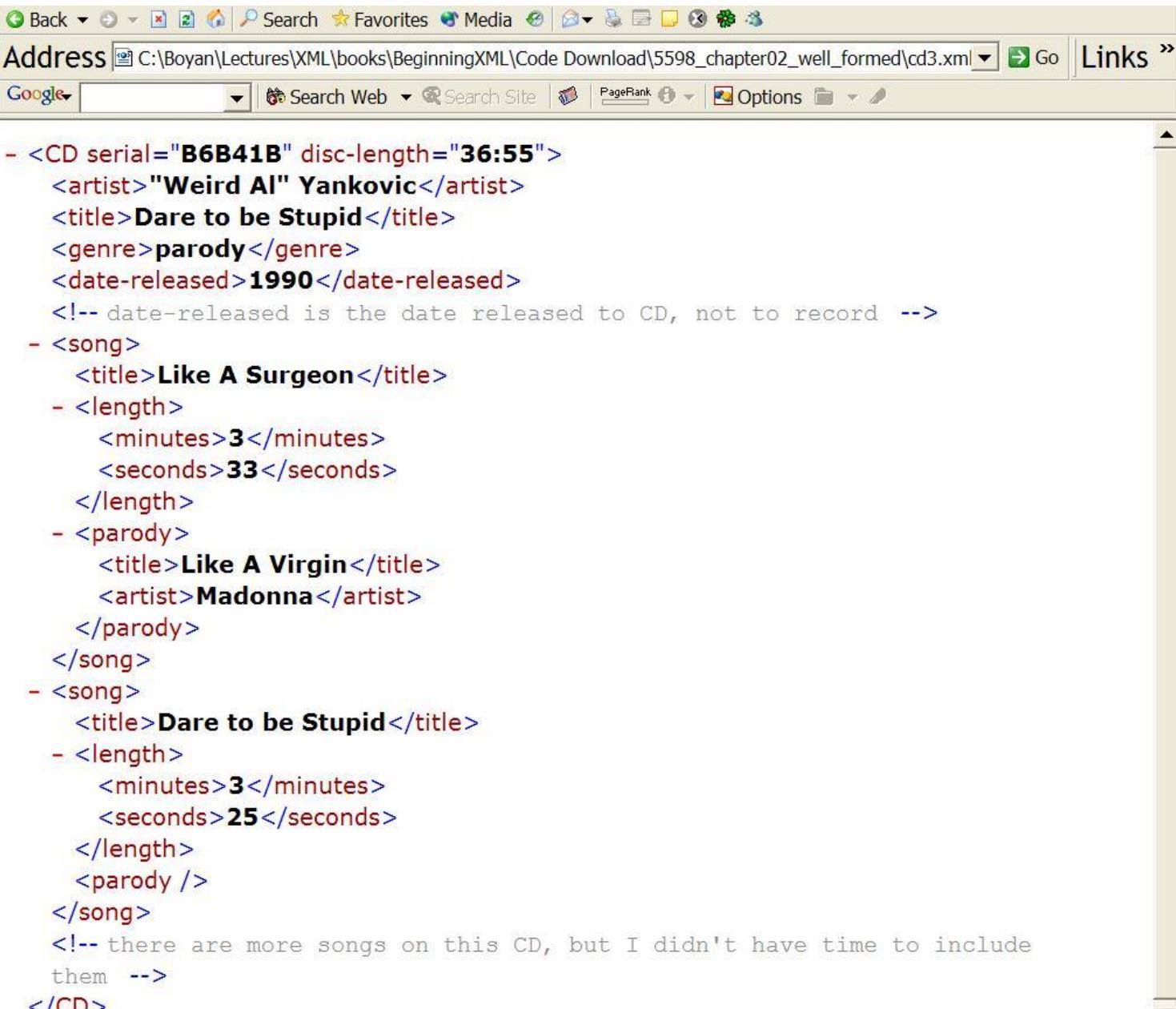


The screenshot shows a Microsoft Internet Explorer window displaying XML code. The XML structure represents a person with various attributes and nested elements, including several comments () that provide descriptive text about the person's characteristics and background.

```
- <person id="123456789">
  - <name title="Bai" alias="New's European">
    <first>Ganyo</first>
    <middle />
    <!-- Who knows the middle name of that famous person? -->
    <last>Balkanski</last>
  </name>
  <birth time="any time" place="any place" />
  - <character physical="slim, tall, cheery, attractive, too clean"
personal="dedicated, honest, philanthropist, leader, idealist">
    <!-- That's why Bai Balkanski turned to be so popular... -->
  </character>
  <address>anywhere in Bulgaria</address>
  - <professions>
    <!-- Initially, he started as a modest trader of rose oil... -->
    - <profession>
      image
      <!-- what a image -->
      maker
    </profession>
    <profession>politician</profession>
  </professions>
</person>
```

XML Добре-структурiran XML

При- мер



The screenshot shows a web browser window with the address bar pointing to a local file: C:\Boyan\Lectures\XML\books\BeginningXML\Code Download\5598_chapter02_well_formed\cd3.xml. The page content displays an XML document structure.

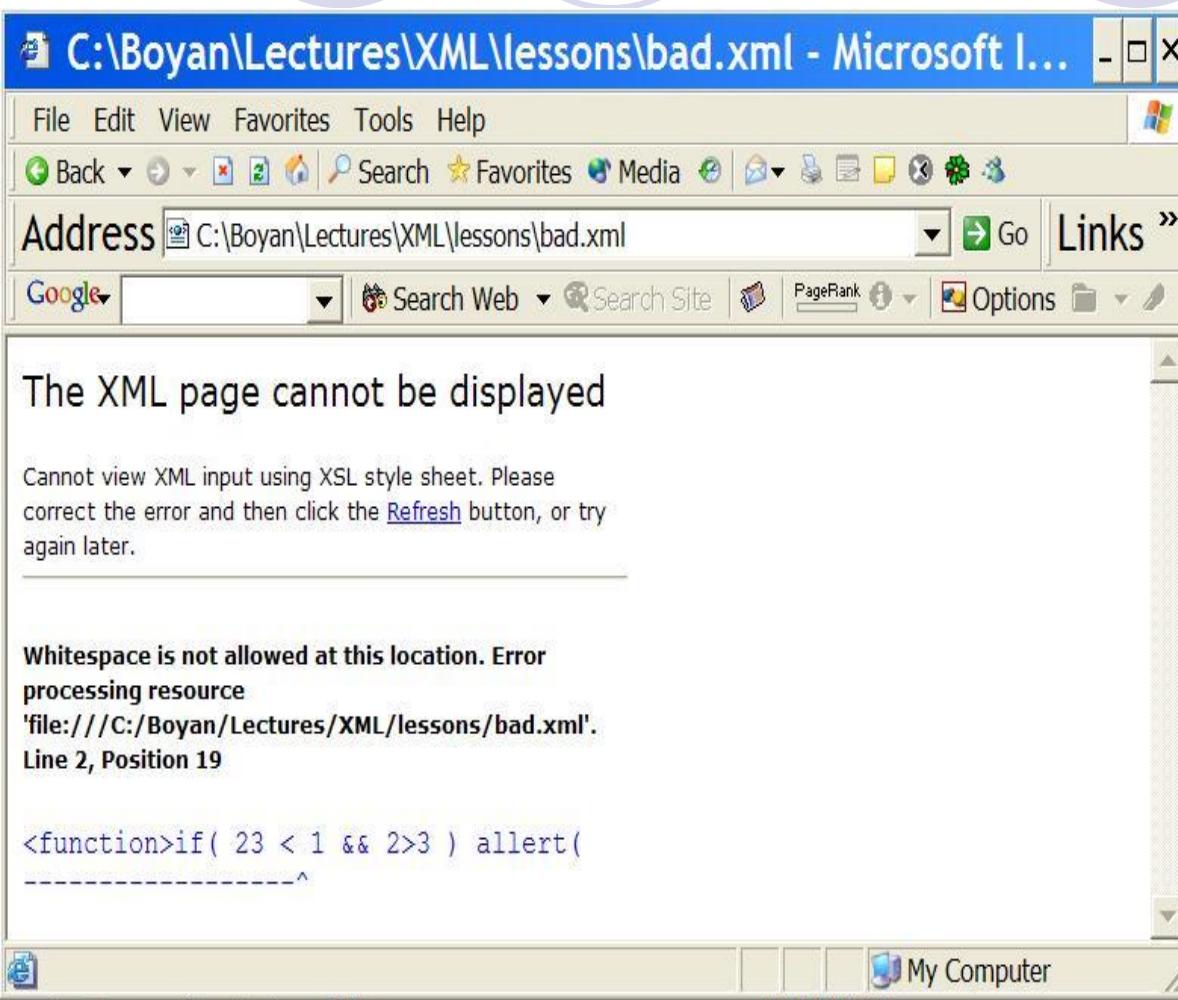
```
- <CD serial="B6B41B" disc-length="36:55">
  <artist>"Weird Al" Yankovic</artist>
  <title>Dare to be Stupid</title>
  <genre>parody</genre>
  <date-released>1990</date-released>
  <!-- date-released is the date released to CD, not to record -->
  - <song>
    <title>Like A Surgeon</title>
    - <length>
      <minutes>3</minutes>
      <seconds>33</seconds>
    </length>
    - <parody>
      <title>Like A Virgin</title>
      <artist>Madonna</artist>
    </parody>
  </song>
  - <song>
    <title>Dare to be Stupid</title>
    - <length>
      <minutes>3</minutes>
      <seconds>25</seconds>
    </length>
    <parody />
  </song>
  <!-- there are more songs on this CD, but I didn't have time to include them -->
</CD>
```

Екранни последователности

- За предефинирани единици:

○ &	&	(ampersand)
○ <	<	(less than)
○ >	>	(greater than)
○ '	'	(apostrophe)
○ "	"	(quotation mark)

Забранени PCDATA символи



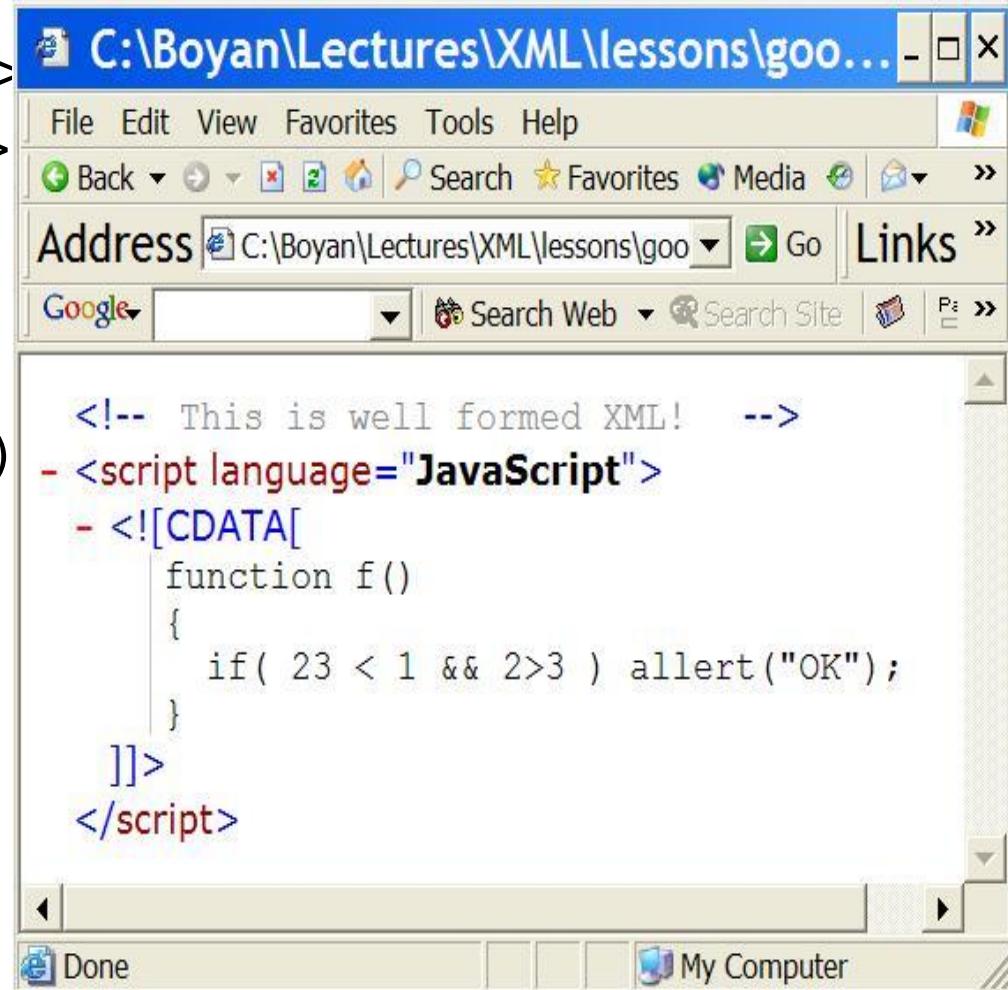
- Пример:
- '<', '&', '>'
●
● <!-- This is not well formed XML! -->
- <function>if(23 < 1 && 2>3) alert("OK");
- </function>

Решение: CDATA секции

- CDATA (Character DATA) идва от SGML
- Започват с **<![CDATA[**
- Завършват с **]]>**
- CDATA съдържанието се игнорира от парсъра и се подава към приложението *as-is*

CDATA пример

- <!-- This is well formed XML! -->
- <script language="JavaScript">
- <![CDATA[
- function f()
- {
- if(23 < 1 && 2>3) alert("OK")
- }
-]]>
- </script>



The screenshot shows a Microsoft Internet Explorer window displaying a well-formed XML document. The XML code includes a CDATA block containing a JavaScript function that alerts "OK" if the condition 23 < 1 && 2>3 is true. The browser's status bar at the bottom indicates "Done" and "My Computer".

```
<!-- This is well formed XML! -->
<script language="JavaScript">
  <![CDATA[
    function f()
    {
      if( 23 < 1 && 2>3 ) alert("OK");
    }
  ]]>
</script>
```

Атрибути

- Атрибутът се задава като наредена двойка име-стойност
- Доставя инфо за елемента
- Стойността се огражда от " или '
- Чувствителност към регистъра
- Видове
 - **value = "Blue Peter" (character data)**
 - **value = "blue" (single token)**
 - **value = "red green blue" (tokens)**
 - enumerated or defaulted (DTD)

Избор между XML елементи и атрибути 1/2

- Ако дадено съдържание представлява неструктурирани или неподредени метаданни за друго, то най-лесно да се представи чрез атрибути. Атрибутите заемат по-малко място и се обработват много лесно от софтуерните приложения посредством интерфейси като DOM, SAX и StAX;
- Ако дадено съдържание представлява структурирани данни или метаданни, то трябва да се представи чрез елементи вместо атрибути. При атрибутите липсва релация за подреждане и затова те не могат да представлят нито линейна, нито йерархична структура;

Избор между XML елементи и атрибути 2/2

- При използване на изброими стойности е удачно да се използват атрибути;
- Ако документът ще се ползва най-вече от хора, за предпочтение е да се използват елементи, понеже структурираната йерархия от елементи е по-четима от множество атрибути, изредени един след друг.

Още по дилемата: Elements or Attributes?

- [1] Does the value have one of an enumeration of values or is the value free-form?
 - [1a] - enumerated values can be name token groups in attributes
 - [1e] - no restrictions on values for the content of elements
- [2] Is the value to be specified, manipulated, organised, consumed by a program or by a human?
 - [2a] - I use attributes for computer-manipulated values
 - [2e] - I use elements for human-manipulated values
- [3] Does the information represent information *about* content, or is the information the content itself?
 - [3a] - I typically put meta-data in attributes
 - [3e] - I typically put content into elements
- [4] Is the information flat or hierarchical?
 - [4a] - attributes are flat and a value has no hierarchy
 - [4e] - elements can be either flat or hierarchical
- [5] Is the information unordered or ordered?
 - [5a] - multiple attribute values in a single start element have no prescribed order
 - [5e] - multiple child elements of an element can be modelled in a prescribed order³⁸

Инструкции за обработка

- Инструкциите за обработка (processing instructions, или съкратено PIs) представляват начин за предаване на скрита информация към дадено приложение.
- Също както коментарите, те не са част от действителното съдържание на документа, но XML процесорите са длъжни да ги взимат под внимание и да ги предават на приложението.
 - Формат- <? ... ?>
 - <?xml version='1.0' ?>
 - <?xml:stylesheet type="text/css" href="sample.css" ?>

XML декларация

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

Кодиране: ISO 8859-1 спрямо UNICODE

- Първи ред от XML документ
 - <?xml version="1.0" **encoding="UTF-8"** ?>
- Не е задължително декларирането на кодиране
- UTF-16 или UTF-8 (Unicode Transformation Format) е кодирането по подразбиране
- Ако кодирането не е указано и то не е нито UTF-8, нито UTF-16 => ERROR!
- UNICODE
 - UTF-8 (включва 7-битовия ASCII)
 - UTF16 (2 байта за всички символи => 65536 chars)
- ISO 8859-1
 - 8 битов код (ANSI)
- Може да се конфигурира в браузъра

UNICODE пример – XML

- `<?xml version="1.0" encoding="UTF-16"?>`
- `<article status="revision">`
- `<title>First Test</title>`
- `<author><first>Susanne</first><last>Dobratz</last></auth or>`
- `<para>This is the next XML document created as test. It shows how <key>formulas</key>like $2+2=4$can be tagged with XML.`
- Testing UNICODE characters: This is UNICODE U00E4 (hexadecimal number): `ä` . and this is UNICODE U0061: `a` and U00A8: `¨` . Even mathematical characters can be encoded in UNICODE:
 - `∀` x `∈` `∃` i.
 - `</para><date>28.08.2002</date>`
 - `</article>`

UNICODE пример – XML в MS IE

C:\Boyan\Lectures\XML\lessons\unicode.xml - Microsoft Internet Exp... -

File Edit View Favorites Tools Help

Back Search Favorites Media PageRank Options

Address C:\Boyan\Lectures\XML\lessons\unicode.xml Go Links

Google Search Web Search Site PageRank Options

```
<?xml version="1.0" encoding="UTF-16" ?>
- <article status="revision">
  <title>First Test</title>
  - <author>
    <first>Susanne</first>
    <last>Dobratz</last>
    </author>
  - <para>
    This is the next XML document created as test. It shows how
    <key>formulas</key>
    like
    <math>2+2=4</math>
    can be tagged with XML. Testing UNICODE characters: This is
    UNICODE U00E4 (hexadecimal number): ä . and this is UNICODE
    U0061: a and U00A8: " . Even mathematical characters can be
    encoded in UNICODE: √ x ∈ ∃ i.
  </para>
  <date>28.08.2002</date>
</article>
```



XML пространства от имена (XML Namespaces)

Конфликти по имена
Пространства от имена
Идентификация
Пространство по подразбиране
Примери

Проблем: дублиране на имена

- Използването на еднакви имена на XML елементи в различен контекст, който не е явно указан, може да доведе до проблеми с дублирани имена.
- Такива колизии могат да възникнат при:
 - смесване в един XML документ на маркирано съдържание от източници с различен контекст, или
 - при работа с един и същ XML документ в различни обкръжения, където имената на елементите се интерпретират по различен начин.



Пример: дублиране на <title>

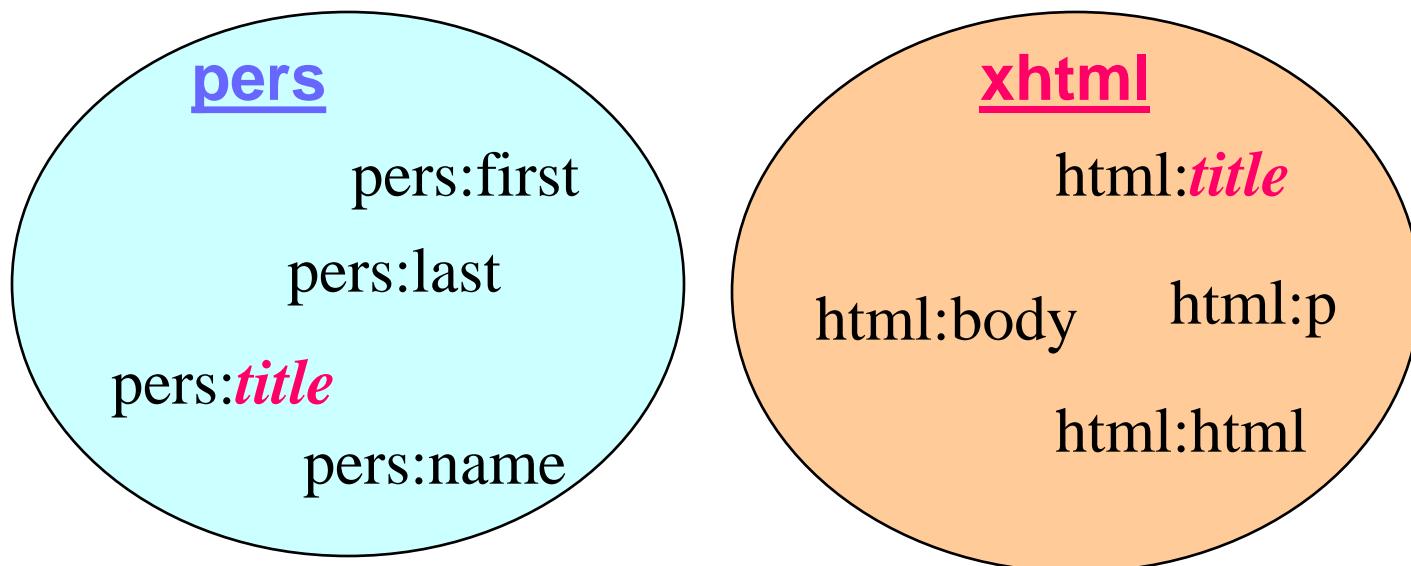
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person>
  <name id="1">
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <resume>
    <html>
      <head> <title>Resume of John Doe</title> </head>
      <body>
        <h1>John Doe</h1>
        <p style="FONT-FAMILY: Arial">
          John's a great guy, you know?
        </p>
      </body>
    </html>
  </resume>
</person>
```

Решение: префиксы (source: *Beginning XML*, by David Hunter et al)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pers:person xmlns:pers="http://sernaferna.com/pers"
               xmlns:html="http://www.w3.org/1999/xhtml">
    <pers:name>
        <pers:title>Sir</pers:title>
        <pers:first>John</pers:first>
        <pers:middle>Fitzgerald</pers:middle>
        <pers:last>Johansen</pers:last>
    </pers:name>
    <pers:position>Vice President of Marketing</pers:position>
    <pers:résumé>
        <html:html>
            <html:head><html:title>Resume of John Doe</html:title></html:head>
            <html:body>
                <html:h1>John Doe</html:h1>
                <html:p>John's a great guy, you know?</html:p>
            </html:body>
        </html:html>
    </pers:résumé>
</pers:person>
```

Идея за пространства от имена (Namespaces)

- Пространствата от имена представляват абстрактна категория за задаване на група (речник) от имена



Квалифицирани имена (Qualified Names)

- Qualified Names (или **QNames**):
 - **QName = namespace_prefix : local_name**
- Виж <http://www.w3.org/TR/1999/REC-xml-names>
- *Но как да се осигури уникалността на имената?*

Уникална идентификация на пространство от имена

- <pers:person xmlns:pers="http://sernaferna.com/pers" xmlns:html="http://www.w3.org/1999/xhtml">
- Решение за осигуряване на уникалността на име на XML пространство от имена - посредством използването на URI (Uniform Resource Identifier)
- URI стандартите задават как име или Уеб ресурс се идентифицира посредством символен низ. Съществуват два типа URI:
 - **URL (Uniform Resource Locator)** – задава уникален адрес на Уеб ресурс и начин на достъп до него
 - **URN (Universal Resource Name)**
 - **prefix:name** (задава кратко уникално име за URL)

URL

URL адресите имат следния формат:

<scheme>://<authority><path>?<query>#<fragment>

Те могат да включват пет съставящи ги компонента:

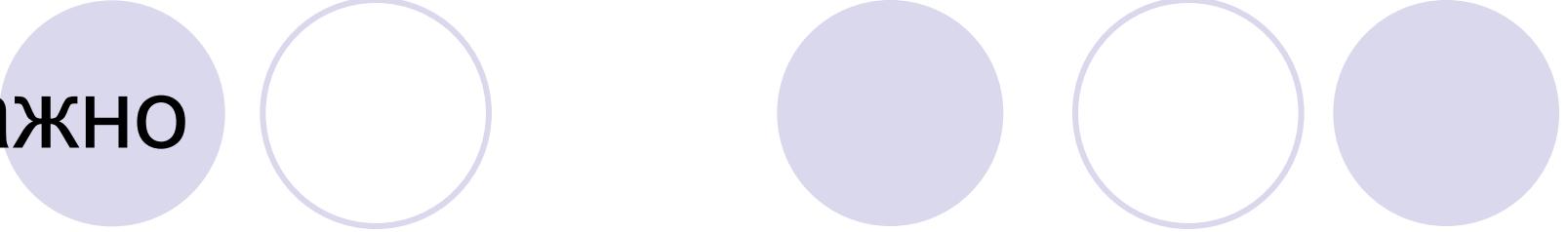
1. Схема, известна още като **протокол** - напр. http
2. **Организация (authority)** - напр. www.biblio.org
3. Път - напр. /java/myApp
4. Заявка (**query string**) - напр. param=123
5. Идентификатор на фрагмент (наречен още секция)

Допълнително, името на организацията може да бъде разделено на потребителска информация, хост и порт.

За URL равен на **http://admin@www.aemon.net:8088/**,

имаме:

- потребителска информация: **admin**
- хост: **www.aemon.net**
- порт: **8088**



Важно

- За осигураване на уникалността на префиксите, означаващи пространства от имена, се използва само URL.
- Не е задължително условие означеният чрез URL ресурс да съществува в Интернет.

Пример

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pers:person xmlns:pers="http://sernaferna.com/pers"
               xmlns:html="http://www.w3.org/1999/xhtml">
    <pers:name>
        <pers:title>Sir</pers:title>
        <pers:first>John</pers:first>
        <pers:middle>Fitzgerald Johansen</pers:middle>
        <pers:last>Doe</pers:last>
    </pers:name>
    <pers:position>Vice President of Marketing</pers:position>
    <pers:résumé>
        <html:html>
            <html:head><html:title>Resume of John Doe</html:title></html:head>
            <html:body>
                <html:h1>John Doe</html:h1>
                <html:p>John's a great guy, you know?</html:p>
            </html:body>
        </html:html>
    </pers:résumé>
</pers:person>
```

Използване на Namespaces

- Дефинират се чрез атрибути:
 - Името на атрибута задава пространството
 - Името на префикса на такъв атрибут винаги е **xmlns**
 - Пространство може да се зададе във всеки елемент
 - ```
<x:html xmlns:x="http://www.w3c.org/TR/REC-html40"
 xmlns:alan="file:/DTD/myDTD.dtd">
 <x:p>An HTML paragraph</x:p>
 <alan:p>My own special p-value markup</alan:p>
 </x:html>
```
- Важат за елементната йерархия, където се дефинират(!)

# Пространство по подразбиране

```
<book
 xmlns="file:/DTD/myDTD.dtd"
 xmlns:X="http://www.w3c.org/TR/REC-html40">
 <X:p>An HTML paragraph</X:p>
 <p>My own special p-value markup</p>
</book>
```

**Важно:** XML пространството от имена се отнася само за юерархията от елементи, за която е дефинирано, а не за всички оставащи до края на документа елементи.

# Пространство по подразбиране – xmlns без префикс

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns="http://sernaferna.com/pers"
 xmlns:html="http://www.w3.org/1999/xhtml">
 <name>
 <title>Sir</title>
 <first>John</first>
 <middle>Fitzgerald Johansen</middle>
 <last>Doe</last>
 </name>
 <position>Vice President of Marketing</position>
 <r sum >
 <html:html>
 <html:head><html:title>Resume of John Doe</html:title></html:head>
 <html:body>
 <html:h1>John Doe</html:h1>
 <html:p>John's a great guy, you know?</html:p>
 </html:body>
 </html:html>
 </r sum >
</person>
```



Default NS

# Пространства и за атрибути

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns="http://sernaferna.com/pers">
 <name id="1">
 <title>Sir</title>
 <first>John</first>
 <middle>Fitzgerald Johansen</middle>
 <last>Doe</last>
 </name>
 <position>Vice President of Marketing</position>
 <r sum >
 <html:html xmlns:html="http://www.w3.org/1999/xhtml">
 <html:head><html:title>Resume of John Doe</html:title></html:head>
 <html:body>
 <html:h1>John Doe</html:h1>
 <html:p html:style="FONT-FAMILY: Arial">
 John's a great guy, you know?
 </html:p>
 </html:body>
 </html:html>
 </r sum >
</person>
```

# Особености на прилагане на пространства за атрибути

- атрибутите нямат самостоятелна идентичност и се отнасят за елемента, в който са дефинирани
- ако елементът е зададен с префикс на дадено пространство от имена, то неговите атрибути могат да бъдат определени с или без използване на този префикс:
  - ако те имат префикс, то те принадлежат на определеното от префикса пространство
  - ако са обаче без префикс, то те не принадлежат на никое пространство, дори и на това, за което е определен елементът на атрибутите

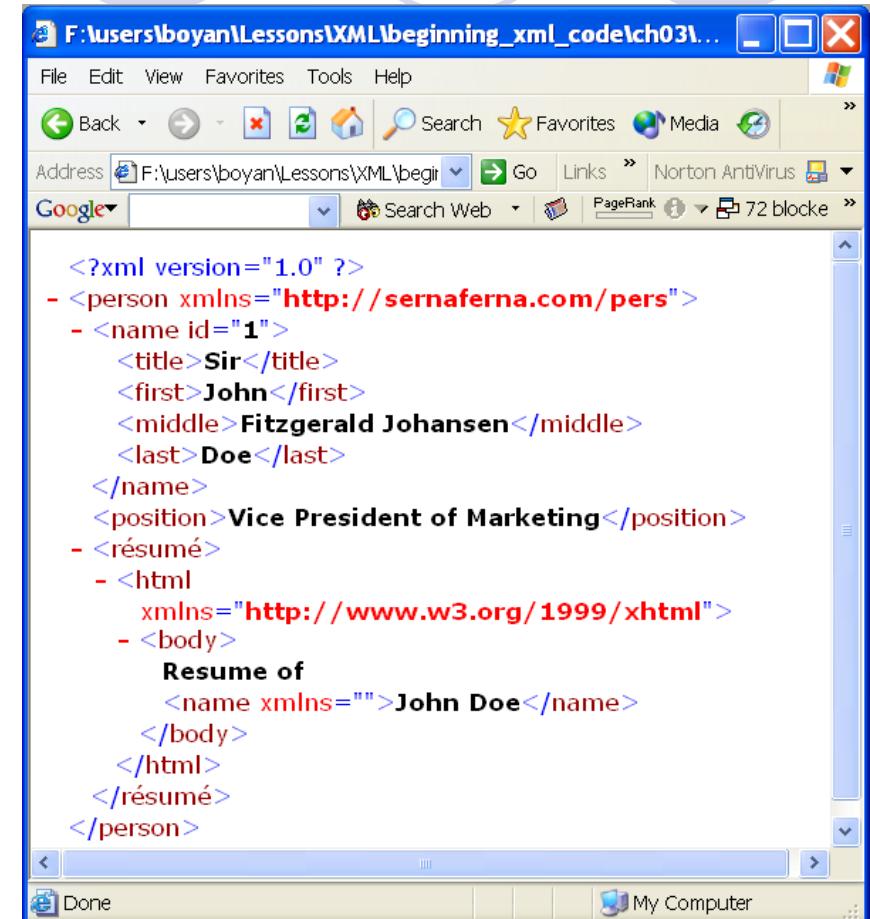
```
<?xml version='1.0'?>
<!--http://www.w3.org is bound to both ref1 and default
namespace-->
<wellFormed xmlns:ref1="http://www.w3.org"
 xmlns= "http://www.w3.org"
 xmlns:ref2="http://www.hmmm.bg">
 <good a="1" b="2" />
 <good a="1" ref1:a="2"/>
 <good ref1:a="1" ref2:a="2"/>
</wellFormed>
```

XML документът е добре конструиран, понеже няма повторения на един и същ атрибут в рамките на начален маркер на елемент. Атрибутите **a** и **ref:a** са различни, понеже първият не принадлежи на никое пространство – дори и на това по подразбиране, а вторият се отнася към пространството с префикс **ref**.

# Канцелиране на пространство по подразбиране

- <?xml version="1.0"?>
- <person  
  xmlns="http://sernaferna.com/pers">
- <name id="1">
- <title>Sir</title>
- <first>John</first>
- <middle>Fitzgerald Johansen</middle>
- <last>Doe</last>
- </name>
- <position>Vice President of  
  Marketing</position>
- <r  sum  >
- <html  
  xmlns="http://www.w3.org/1999/xhtml">
- <body>
- Resume of <name xmlns=""> John  
  Doe</name >
- </body>
- </html>
- </r  sum  >
- XML</person>

xmlns==“ ?”



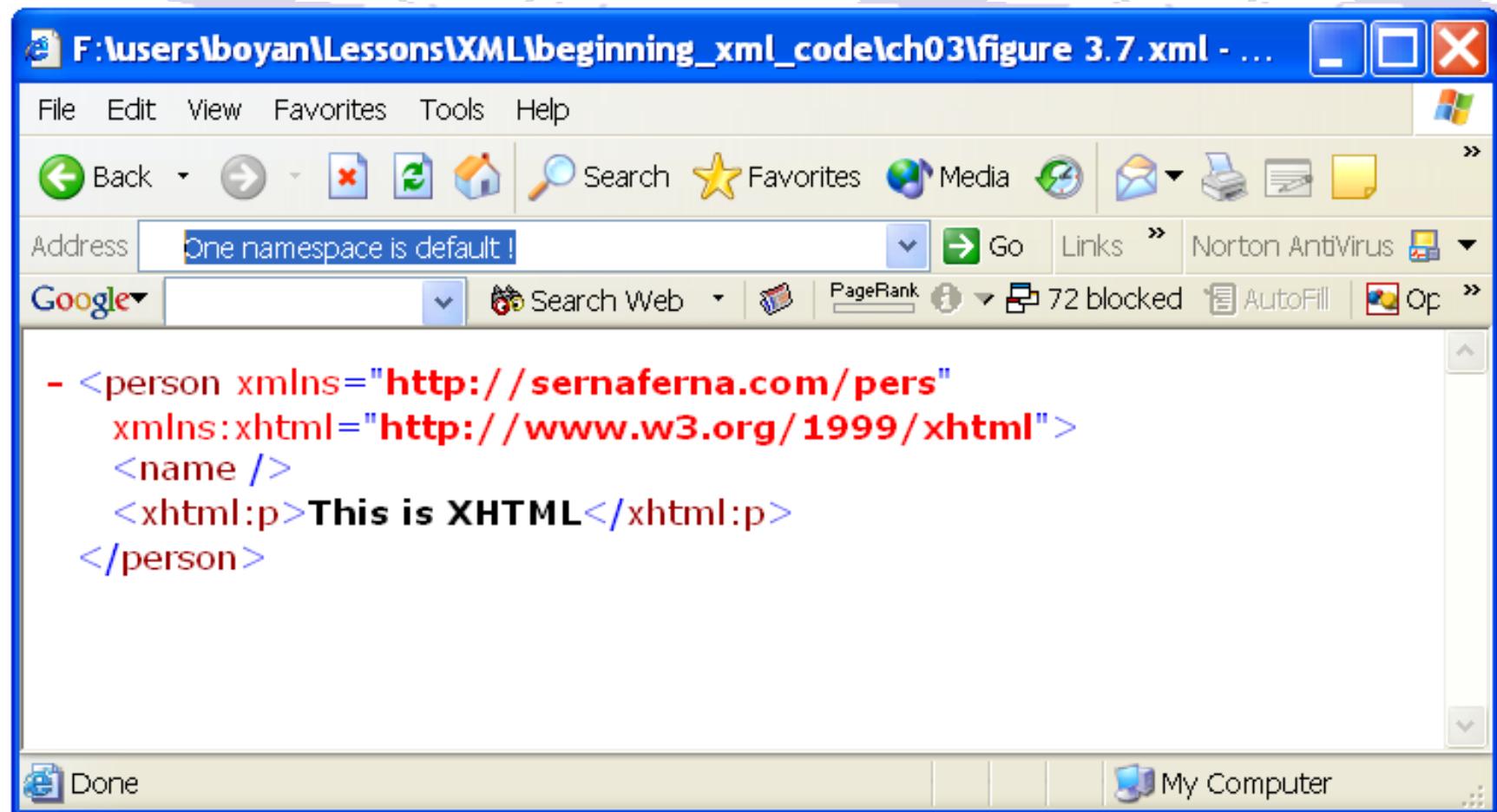
# 3 различни нотации – 1/3

The screenshot shows a Microsoft Internet Explorer window with the following details:

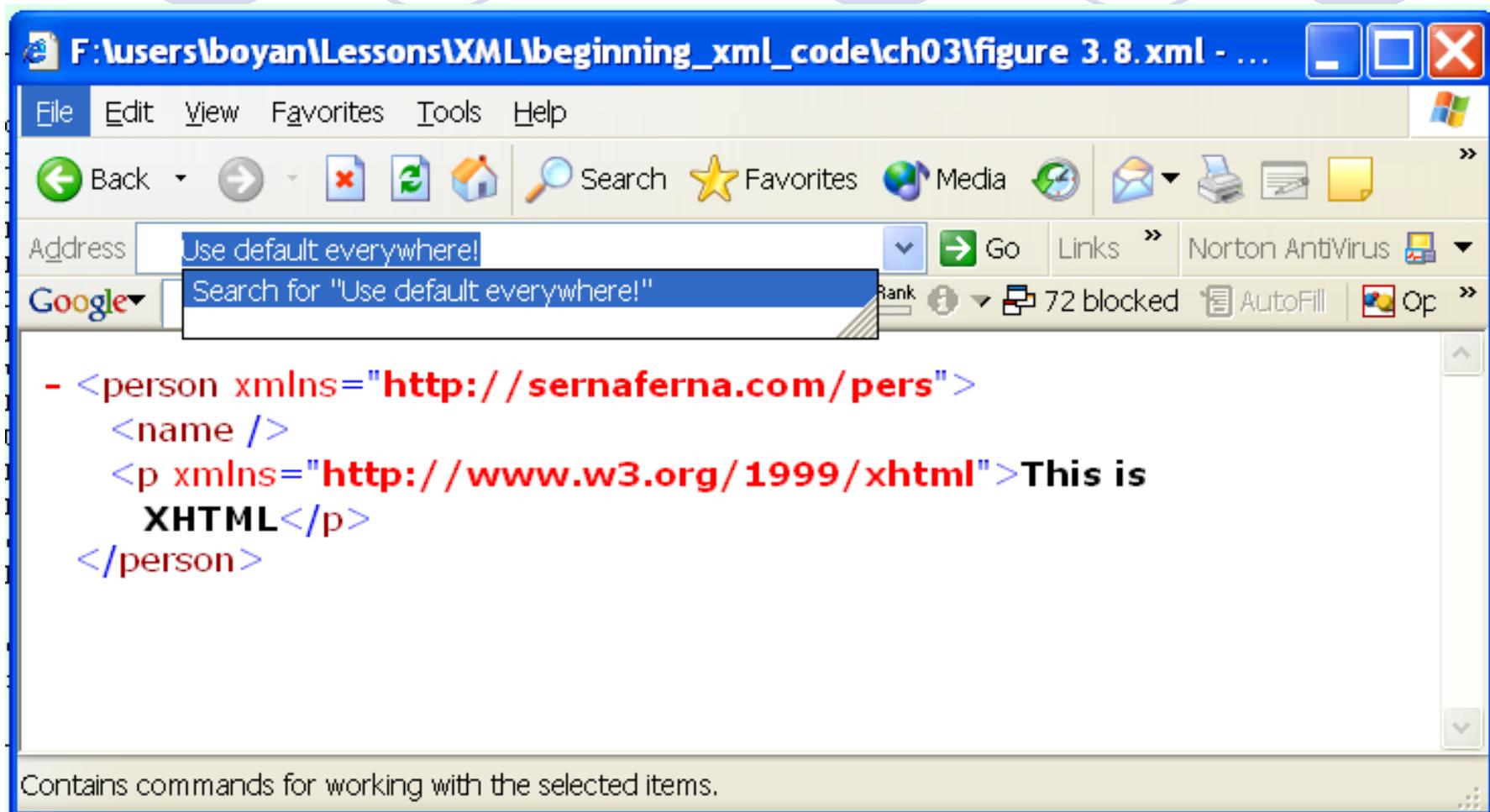
- Title Bar:** F:\users\boyan\Lessons\XML\beginning\_xml\_code\ch03\figure 3.6.xml...
- Menu Bar:** File, Edit, View, Favorites, Tools, Help.
- Toolbar:** Back, Forward, Stop, Refresh, Home, Search, Favorites, Media, Mail, Print, Help.
- Address Bar:** Address: We can qualify every name! Go, Links, Norton AntiVirus.
- Search Bar:** Google, Search Web, PageRank, 72 blocked, AutoFill.
- Content Area:** Displays the following XML code:

```
- <pers:person xmlns:pers="http://sernaferna.com/pers"
 xmlns:xhtml="http://www.w3.org/1999/xhtml">
 <pers:name />
 <xhtml:p>This is XHTML</xhtml:p>
</pers:person>
```
- Status Bar:** Done, My Computer.

# 3 различни нотации – 2/3



# 3 различни нотации – 3/3



The screenshot shows a Microsoft Internet Explorer window displaying an XML document. The title bar reads "F:\users\boyan\Lessons\XML\beginning\_xml\_code\ch03\figure 3.8.xml - ...". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The toolbar contains icons for Back, Forward, Stop, Refresh, Home, Search, Favorites, Media, Mail, Print, and others. The address bar shows "Use default everywhere!" and a search bar below it with the text "Search for 'Use default everywhere!'". The main content area displays the following XML code:

```
- <person xmlns="http://sernaferna.com/pers">
 <name />
 <p xmlns="http://www.w3.org/1999/xhtml">This is
 XHTML</p>
</person>
```

A context menu is open at the bottom of the XML code, with the status bar indicating "Contains commands for working with the selected items."

## За допълнителен прочит...

- Namespaces in XML 1.0 (Third Edition) – W3C Recommendation, 8 December 2009:

<http://www.w3.org/TR/REC-xml-names/>

# XML валидиране чрез Document Type Definition (DTD)



Цели  
Структура на DTD  
Синтаксис  
Особености  
Единици  
Примери

# XML синтаксис накратко

- Elements
  - XML tags for markup
- Attributes
  - Tuple information of elements
- Declarations
  - Instructions to XML processor
- Processing Instructions
  - Instructions to external applications

# Дефиниция на типа на документа (DTD)

- DTD дефиницията задава шаблон за маркиране на XML документ
- Форматът на DTD е наследен от SGML, като значително е опростен
- Както при SGML, така и XML DTD използва формална граматика за описание на структурата и типа на съдържанието на XML документа
- DTD осигурява начин за проверка на неговата структура и съдържание – т.е. за **валидация**

# DTD и XML schema

- DTD (Document Type Definition) и XML schema или XSD (XML Schema Definition) задават правила, съгласно които се определят имената на елементите и атрибутите, тяхната последователност, честота на срещане и др.
- DTD използва по-стегнат и кратък синтаксис в сравнение с XML Schema, но за сметка на това XML Schema предоставя по-богат набор от средства за по-строго дефиниране на структурата на XML и освен това нейните правила се задават в XML формат.

# Валиден XML документ

- Всеки отделен документ, отговарящ на даден документен тип, е документен екземпляр (инстанция) на типа. Такъв документ представлява валиден документ за този документен тип.
- Всеки валиден документ е добре конструиран, но обратното не е задължително вярно.
- Всички XML парсери проверяват дали входния документ е **добре конструиран** XML документ.
- Парсери, които извършват още и проверка за определяне дали съдържанието на XML документите е валидно спрямо зададен тип на документа, се наричат **валидиращи** парсери.

# Валидация

- Валидацията е времеемък процес, но често спестява много проблеми на външните приложения и се извършва от специализиран процесор (валидиращ парсер)
- DTD описанието е споделено от валидизирация парсер за XML документите-екземпляри (инстанции) на този документен тип – т.е. използва се многократно само едно описание

# Същност на DTD

DTD документите описват правилата за структуриране на документа, изграждащите го елементи, възможните типове атрибути и стойностите им по подразбиране. По-конкретно, те задават:

- *Какви имена на елементи и техни атрибути могат да бъдат използвани в документа;*
- *Какви са типовете на елементите и техните атрибути;*
- *Каква е юрархията на документа;*
- *Какви видове единици се ползват в него.*

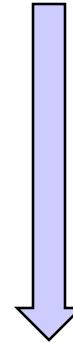
# Рефериране към вътрешно DTD

- <?xml version="1.0" encoding="windows-1251" ?>
- <!DOCTYPE **recipe** [
  - <!ELEMENT recipe (category,author,title,body)>
  - <!ELEMENT category (#PCDATA)>
  - <!ELEMENT author (#PCDATA)>
  - <!ELEMENT title (#PCDATA)>
  - <!ELEMENT body (#PCDATA)>
- ]>
- <**recipe**>
- <category>Cakes</category>
- <author>Ralica</author>
- <title>Chocolate cake</title>
- <body>Products: ....
  - **Way of preparation:....**
  - **Result: Very delicious!**
- </body>
- XML </**recipe**>

*Допустимо е само  
едно вътрешно  
DTD описание*

# Използване на външна DTD дефиниция

- <?xml version="1.0" encoding="windows-1251" ?>
- <!DOCTYPE recipe SYSTEM "example-DTD-2.dtd">
- <recipe>
- <category>Cakes</category>
- <author>Ralica</author>
- <title>Chocolate cake</title>
- <body>Products: ....  
        Way of preparation:....  
        Result: Very delicious!  
    </body>
- </recipe>



```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT recipe
 (category,author,title,body)>
 <!ELEMENT category (#PCDATA)>
 <!ELEMENT author (#PCDATA)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
 <!ENTITY footer "www.recipe.com">
 <!ATTLIST author profession CDATA
#REQUIRED>
```

# Външни DTD дефиниции

- <?xml version="1.0"?>
- <!DOCTYPE Rolodex SYSTEM "rolodex.dtd">
- . . . { *XML document instance goes here* }

**SYSTEM (local file, URL) or PUBLIC (for public catalogues) descriptors**

**PUBLIC – a catalog entry in Formal Public Identifiers format (FPI)**

# Комбинация от двата подхода

- с дефиниран DTD както в отделен документ, така и в самия XML. В този случай вътрешната DTD може да предефирира само ENTITY и ATTLIST на външната дефиниция. Пример:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE recipe SYSTEM "example-DTD-2.dtd" [
 <!ENTITY footer "www.myrecipe.com">
 <!ATTLIST author title CDATA #REQUIRED>]>
<recipe>
 <category>Cakes</category>
 <author title="eee" profession="musician"> &footer;
 </author>
 <title>Chocolate cake</title>
 <body>Products:
 Way of preparation:....
 </body>
 <comment> Very delicious!</comment>
</recipe>
```

# DTD декларации

- Инструкции за XML процесора
- Формат - <! ... > or <! ... [<! ... >]>
  - Document type - <!DOCTYPE ... >
  - Character data - <! [CDATA[ ... ] ]>
  - Entities - <!ENTITY ... >
  - Notation - <!NOTATION ... >
  - Element - <!ELEMENT ... >
  - Attributes - <!ATTLIST ... >
  - Conditional sect.'s <! [INCLUDE [...] ]> and  
<! [IGNORE [...] ]>

# Дефиниране на елементи в DTD

Ако елементът съдържа само текст:

```
<!ELEMENT element_name (#PCDATA)>
```

Ако е празен:

```
<!ELEMENT element_name EMPTY>
```

Ако е с елементно съдържание:

```
<!ELEMENT element_name (child_1, child_2)>
```

Валиден XML съгласно тази DTD дефиниция е:

```
<element_name>
 <child_1> </child_1>
 <child_2> </child_2>
</element_name>
```

# Избор на елементно съдържание

Под-елементите могат да бъдат разделени и със знака '|', който има значение на логическия оператор *OR*:

```
<!ELEMENT element_name
((child_1, child_3) | (child_2, child_4))>
```

# Честота на срещане на елементи

- ? – със значение нула или веднъж,
- \* - със значение нула или повече пъти,
- + - веднъж или повече пъти.
- Тези *оператори за честота* се поставят след съответния елемент / елементи така:
  - **<!ELEMENT element-name**
  - **(child\_1\*, (( child\_2, child\_3?) | child\_4+)\*,...)>**
  - Ако липсва такъв оператор след даден елемент или множество от елементи, то те трябва да се срещнат в съответния XML точно веднъж.

# Елементи с произволно съдържание

- **<!ELEMENT element-name ANY>**
- Съдържанието му може да бъде произволно – да бъде празен или не, да съдържа текст или не, да има смесено съдържание или различни под-елементи в произволен ред и т.н.

# Моделни групи

- Дефинират съдържанието на елементите
  - <!ELEMENT person (name, e-mail\*)>
- Задават йерархията на елемент
  - <!ELEMENT name (fname, surname)>  
    <!ELEMENT fname (#PCDATA)>  
    <!ELEMENT surname (#PCDATA)>  
    <!ELEMENT e-mail (#PCDATA)>
- Управляват организацията на елементите
  - Sequence connector - ',' - (A, B, C) [then]
  - Choice connector - '|' - (A | B | C) [or]



# Смесено съдържание

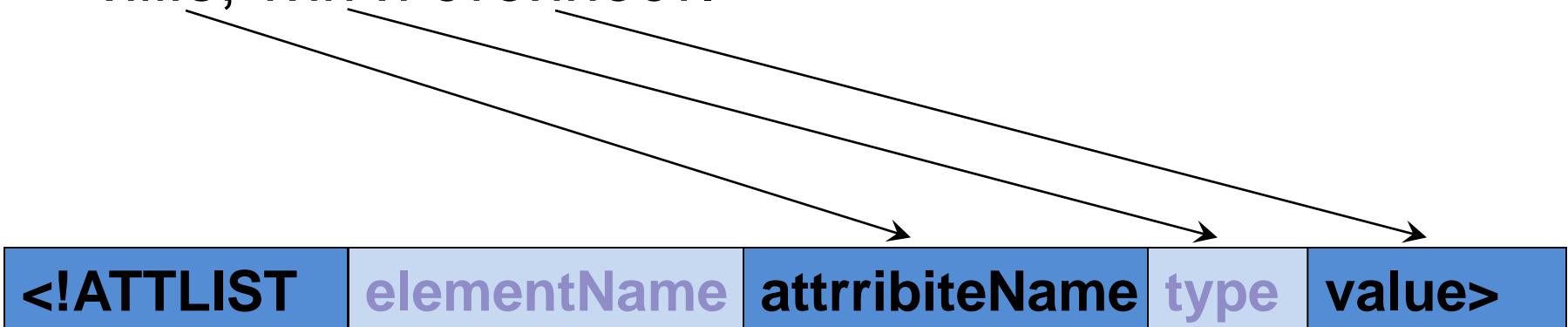
- `<!ELEMENT descr (#PCDATA | a | b)*>`
- *#PCDATA трябва да предхожда останалата част от дефиницията*
- Използваме '*choice*' за разделяне на елементите

# Грануларността на XML структурата се отразява в DTD

- <PERSON>  
    <NAME>John Smith</NAME>  
    </PERSON>
- <PERSON>  
    <FORENAME>John</FORENAME>  
    <SURNAME>Smith</SURNAME>  
    </PERSON>

# Деклариране на атрибути в DTD

- Декларацията на атрибут:
- ключова дума **ATTLIST**
- име на елемент, към който е свързан атрибута;
- списък на деклариирани атрибути
- За всеки атрибут задължително се задава име, тип и стойност.



# Още за атрибути

- Декларация на множество атрибути:
- **<!ATTLIST myElementName**
- **attrName1 TYPE VALUE**
- **attrName2 TYPE VALUE**
- **attrName3 TYPE VALUE>**
- **<!ATTLIST myElementName**
- **attrName4 TYPE VALUE>**
- Дефинираните пространства от имена, напр.  
**xmlns:tasks = "http://aemon.net/project/tasks"**, също се третират като атрибути.
- **Атрибутите без префикс на пространство от имена не се отнасят към пространството от имена по подразбиране.**

# Име и тип на атрибут

- Име на атрибут

- `<!ATTLIST tag name type default>`
- `<!ATTLIST tag first_attr ...`  
`secon_attr ...`  
`third_attr ... >`

- Тип на атрибут

**CDATA**

**NMTOKEN**

**NMTOKENS**

**ENTITY**

**ENTITIES**

**ID**

**IDREF**

**IDREFS**

**NOTATION**

**name group**

# Повече за типовете

- CDATA (non-parsed)
  - Character data
- NMTOKEN
  - Single token
- NMTOKENS
  - Multiple tokens
- ENTITY
  - Attribute is entity ref
- ENTITIES
  - Multiple entity ref's
- ID
  - Unique ID
- IDREF
  - Match to given unique ID defined elsewhere in the XML.
- IDREFS
  - Match to multiple ID's
- NOTATION
  - Describe non-XML data
- Name group
  - (enumeration) – restricted list

has to start with a letter or ':' or '\_'. The rest of the characters must be numbers, letters ':', '\_', '-', or ''

# Примери за типове на атрибути

- CDATA

- name = "<Tom Jones>"

- NMTOKEN

- color="red"

- NMTOKENS

- values="12 15 34"

- ENTITY

- photo="MyPic"

- ENTITIES

- photos="pic1 pic2"

- ID

- ID = "P09567"

- IDREF

- IDREF="P09567"

- IDREFS

- IDREFS="A01 A02"

- NOTATION

- FORMAT="TeX"

- Name group

- coord="X"

# Дефиниции на типове на атрибути

- CDATA

- <!ATTLIST person name CDATA ... >

- NMTOKEN (XML

NameChar symbols only)

- <!ATTLIST mug color NMTOKEN ... >

- NMTOKENS

- <!ATTLIST temp val's NMTOKENS ... >

- ENTITY

- <!ATTLIST person photo ENTITY ... >

- ENTITIES

- <!ATTLIST album photos ENTITIES ...>

- ID

- <!ATTLIST person id ID ... >

- IDREF

- <!ATTLIST person father IDREF ... >

- IDREFS

- <!ATTLIST person children IDREFS ... >

- NOTATION

- <!ATTLIST image format NOTATION (TeX|TIFF) ...>

- Name group (enum)

- <!ATTLIST point title\_point (Mr|Ms|Mrs|Rev|Dr) ... >

# Указания за стойност на атрибут

- **#REQUIRED**
- **#IMPLIED**
- "default"
- **#FIXED**

Трябва да бъде зададен  
Може да бъде зададен  
Стойност по подразбиране  
ако не е зададен  
Само една разрешена  
стойност

Синтаксис:

```
"<!ATTLIST tag name type default>"
<!ATTLIST seqlist sepchar NMTOKEN #REQUIRED
 type (alpha | num) "num"
 ver CDATA #FIXED "1.0"
```

```
<?xml version="1.0"?>
<!DOCTYPE students [
 <!ELEMENT students (student*, groups_of_students)>
 <!ELEMENT student (name, number, university)>
 <!ELEMENT name EMPTY>
 <!ELEMENT number EMPTY>
 <!ELEMENT university (#PCDATA)>
 <!ELEMENT groups_of_students (group*)>
 <!ELEMENT group (#PCDATA)>
 <!ATTLIST number student_no ID #REQUIRED>
 <!ATTLIST name first CDATA #IMPLIED>
 <!ATTLIST name last CDATA #REQUIRED>
 <!ATTLIST group studentNum IDREF #REQUIRED>
 <!ATTLIST group status (important|normal) "important">
 <!ATTLIST university logo ENTITY #IMPLIED>
 <!ATTLIST university format NOTATION (jpg|gif) #IMPLIED>
 <!ENTITY fmi SYSTEM "www.uni-sofia.bg/logo.gif" NDATA jpg>
 <!ENTITY oxf SYSTEM "www.ox.ac.uk/display_images/logo.gif"
 NDATA gif>
 <!NOTATION jpg PUBLIC "jpg viewer">
 <!NOTATION gif PUBLIC "gif viewer">
]>
<students>
 <student>
 <name last="Deyanov"/>
 <number student_no="MNI5567"/>
 <university logo="fmi" format="jpg">Sofia University</university>
 </student>
 <student>
 <name first="Mitko" last="Bombev"/>
 <number student_no="MNI7890"/>
 <university logo="oxf">Oxford</university>
 </student>
 <groups_of_students>
 <group studentNum="MNI5567" status="important">IT</group>
 <group studentNum="MNI7890" status="normal">IT</group>
 </groups_of_students>
</students>
```

# Псевдо-атрибути

- Имат вида **xml:pseudoAttrName**
- Стойността на псевдо-атрибутите трябва да бъде предефирирана, като напр. тази на **xml:lang** или **xml:space**, които съответно се използват за дефиниране на код на език на съдържанието на элемента съгласно ISO-639 и за запазване или не на допълнителни интервали в съдържанието на элемента.
- Друг псевдо-атрибут е **xml:id**, който задава условие за уникална стойност, без това да е необходимо да се декларира в DTD.
- xml:base** анотира елемент с базов URI

# Пример

```
<поем xml:space="default">
 <author>
 <givenName> Димчо </givenName>
 <familyName>Дебелянов</familyName>
 </author>
 <verse xml:space="preserve">
 <line> И ето скръб </line>
 <line>крила над мен привежда</line>
 <signature xml:space="default">
 Д. Дебелянов</signature>
 </verse>
</поем>
```

# Интервали в атрибутите

- Въпреки че XML процесорите запазват всички интервали (бели полета) в **съдържанието на елемента**, те често го нормализират в **стойностите на атрибутите**.
- Табулатии, пренос на нов ред, както и празни пространства се отчитат като един интервал.
- Ако документът е с дефиниран DTD, това подрязване ще се извърши за всички атрибути, които не са от тип CDATA.

# Декларация на коментари

- Коментарите не са част от съдържанието на документа
  - `<!-- A comment -->`
- Забранено е ползването на '--' в коментар
- Коментар не може да включва други декларации в себе си

# Декларация на символна секция

- За документа:
  - Press <<<ENTER>>>
- Декларираме:
  - <! [CDATA [Press <<<ENTER>>> ] ]>
- Дискусия: как да представим в CDATA стринга "]]>" ?

# Символна секция със стринга "]]>"

```
<! [CDATA[]]] > <! [CDATA[>]] >
```

]]>

# Използване на символни секции при inline <script> и <style> елементи в XHTML документи

```
<script type="text/javascript">
//<![CDATA[
document.write("<");
//]]>
</script>
<style type="text/css">
/*<![CDATA[*/
body { background-image: url("111.png?width=300&height=300") }
/*]]>*/
</style>
```

# Дискусия

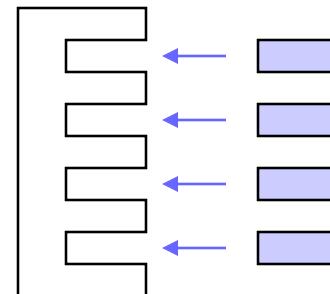
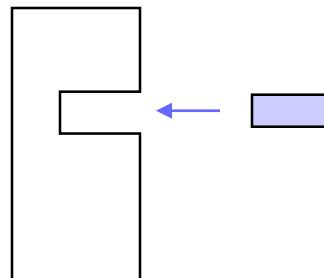
- Каква е разликата при интерпретирането на двета реда по-долу:
- **<sender>Емил Боев</sender>**
- и
- **&lt;sender&gt;Емил Боев&lt;/sender&gt;**

# Инструкции за обработка (Processing Instructions)

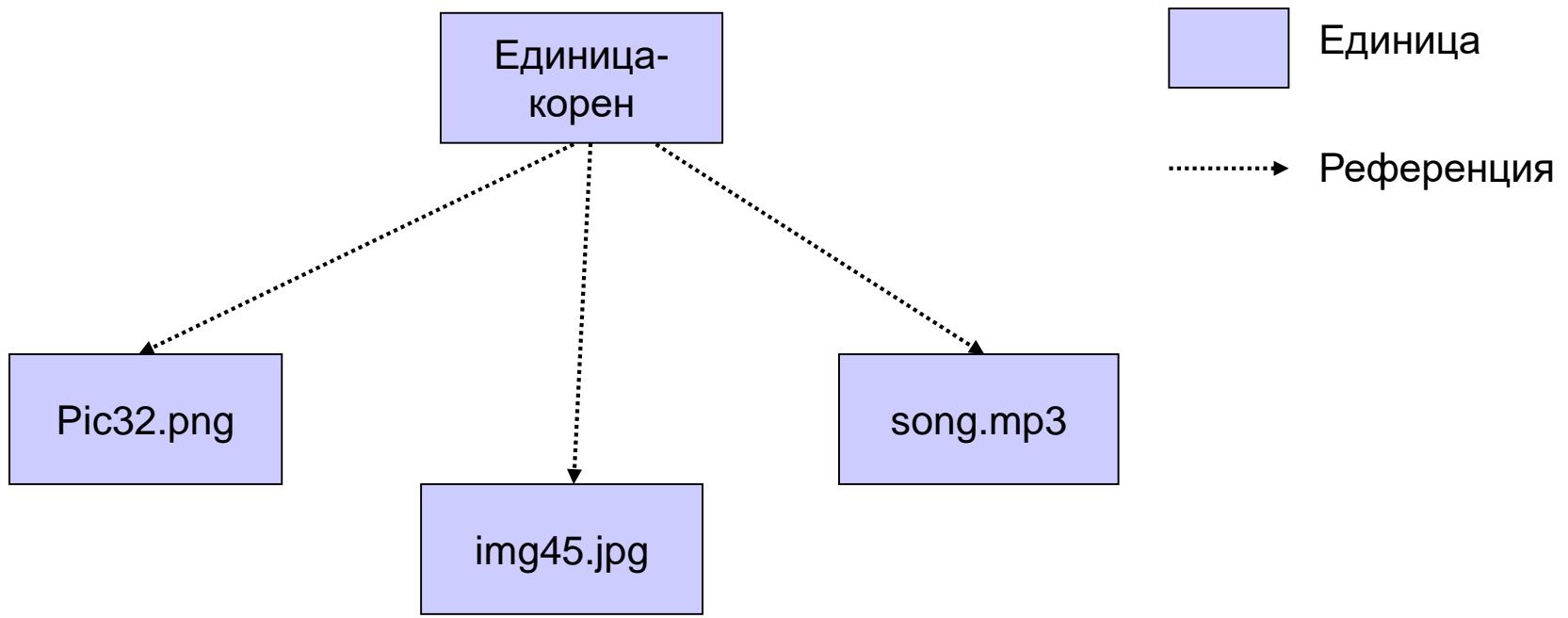
- Задават информация за външни приложения
- Processing Instructions
  - Формат: <? ... ?>
  - XML PI пример: <?xml version='1.0' ?>
    - XML декларацията всъщност е инструкция за обработка

# Единици (Entities)

- XML документът може да бъде разпределен сред голям брой файлове. Всяка част от организирането на информацията се нарича единица (*entity*).
- Единицата има име, да я идентифицира. Тя се дефинира чрез специална **декларация (ENTITY)** и се използва през **референция**, която я указва.
- В един XML документ, **ENTITY** референциите се използват като указател към текст или външен ресурс.



# Единици (XML Entities)



# Кога да използваме единици

- Единиците подобряват четимостта на XML документите:
  - Повторението на един и същи текст на много места се заменя с кратък запис
  - Могат да имат различно представяне
  - Разделянето на документа го прави по-лесно управляем
  - Може да се използват и други, не-XML формати

# Типове единици

## ● Общи (General)

- Декларирани в DTD;  
Реферирали в XML  
документите

## ● Параметрични

- Декларирани в DTD;
- Реферирали  
единствено в  
декларациите на DTD;

## ● Вътрешни (Internal)

- Съхраняват се в XML  
документа
- Текстово съдържание

## ● Външни (External)

- Съхраняват се извън  
XML документа
- Текстово или двоично  
съдържание
- Могат да групират  
други единици

# Общи единици

- Декларирали в 'Document Type Declaration'
  - <!DOCTYPE My\_XML\_Doc [  
    !ENTITY name "replacement">  
]>
- Използват се в XML - пример:
  - В DTD: <!ENTITY xml "eXtensible Markup Language">
  - В XML: The &xml; includes entities
  - Резултат: The eXtensible Markup Language  
includes entities

# Предефинирани общи единици

Могат да бъдат използвани без да е необходимо да се специфицират. Те са:

- **&apos;** - представя знака апостроф (' ) – резервиран заедно с (" ) за ограждане на стойности на атрибути;
- **&quot;** - представя знака кавички (" ) - резервиран заедно с ( ' ) за ограждане на стойности на атрибути;
- **&amp;** - представя знака амперсанд (&)-резервиран за задаване на единици;
- **&gt;** - представя знака по-голямо (>) - резервиран заедно с (<) за ограждане на елементи;
- **&lt;** - представя знака по-малко (<) - резервиран заедно с (>) за ограждане на елементи.

# Параметрични (Parameter Entities)

- Декларирали се в 'Document Type Declaration'
  - ```
<!DOCTYPE My_XML_Doc [  
    <!ENTITY % name "replacement">  
]>
```
- Използват се в DTD:
 - ```
<!ENTITY % param "(para | list)">
```
  - ```
<!ELEMENT section (%param;)*>
```
 - Вместо
 - ```
<!ELEMENT section (para | list)*>
```

# Вътрешни единици

- Реферират текст, който е дефиниран в DTD документа. Дефиницията им започва с ключовата дума **ENTITY** следва името на единицата и накрая нейната стойност заградена в кавички.
- **<?xml version="1.0" standalone="yes" ?>**
- **<!DOCTYPE name [**
- **<!ELEMENT name (#PCDATA)>**
- **<!ENTITY myFirstName "Elen">**
- **<!ENTITY myLastName "Lenon">**
- **]>**
- **<name>&myFirstName; &myLastName;</name>**

# Външни (External Entities)

Съдържат XML съдържание – реферираат към данни, които един XML процесор трябва да може да обработи

- А. **частни** – при дефиниция се използва ключовата дума **SYSTEM**. Те са предназначени и достъпни за определена група от хора:
- **<!ENTITY entity\_name SYSTEM "URI">**
- Б. **публични** - използват ключовата дума **PUBLIC**. Те са налични в Интернет и могат да бъдат използвани от всеки:
- **<!ENTITY entity\_name PUBLIC "PUBLIC\_ID" "URI">**

# Външна частна единица

- <?xml version="1.0" standalone="no" ?>
- <!DOCTYPE entityExample [
- <!ELEMENT entityExample (#PCDATA)>
- <!ENTITY entityData SYSTEM "example.txt">
- ]>
- <entityExample>&entityData;</entityExample>

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE entityExample (View Source for full doctype...)>
<entityExample>text load from external file via entity.</entityExample>
```



# Външна публична единица

- **<!ENTITY entity\_name PUBLIC "PUBLIC\_ID" "URI">**
  - **entity\_name** е името, с което съответната единица ще бъде използвана в XML документа, **PUBLIC\_ID** може да се използва от XML процесора, за да генерира алтернативен URL адрес и ако той не може да бъде намерен, то се използва дефинирания адрес в **URI**.
- **<?xml version="1.0" standalone="no" ?>**
- **<!DOCTYPE entityExample [**
- **<!ELEMENT entityExample (#PCDATA)>**
- **<!ENTITY entityData PUBLIC "-//W3C//TEXT entity//BG"**  
**"http://www.w3.org/xmlspec/entity.xml" >**
- **]>**
- **<entityExample>&entityData;</entityExample>**

# Външни единици, съдържащи не- XML съдържание

- За вграждане на не-XML данни (напр. графика), използваме външно ***unparsed entity***
- Реферират към данни, които XML процесора не трябва да обработва
- Могат да бъдат частни и публични
- След ключовата дума *NDATA* следвана от име на задължително дефинирана в DTD документа нотация:
- **<!ENTITY entityName SYSTEM "URI" NDATA notation\_name>**
- **<!ENTITY entityName PUBLIC "PUBLIC\_ID" "URI" NDATA notation\_name>**

# Външни единици, съдържащи не- XML съдържание - пример

```
<?xml version="1.0"?>
<!DOCTYPE company [
 <!ELEMENT company (logo) >
 <!ELEMENT logo EMPTY>
 <!ATTLIST logo url ENTITY #REQUIRED>
 <!ENTITY company_logo SYSTEM " logo.gif" NCDATA GIFGIF PUBLIC "image/gif ">
]>
<company>
 <logo url="company_logo"/>
</company>
```

# Нотация (Notation)

- Нотацията се използва, за да специфицира различни от XML данни като например файлове от тип image/gif, image/jpeg и др.
- Специфичното при този тип атрибут е, че при декларация трябва да бъдат изброени стойностите, които атрибутът може да приема и всяка една от тях трябва да бъде име на декларирана в DTD документа нотация.

# Декларация на нотация (Notation)

- Описва external non-XML entity
- Използва се с **NDATA**

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ENTITY turing_getting_off_bus
 SYSTEM "http://www.tur.org.uk/aa.jpg"
 NDATA jpeg>
<!ELEMENT image EMPTY>
<!ATTLIST image source ENTITY #REQUIRED>

<image source="turing_getting_off_bus"/>
```

- Друг начин за това е да се ползва XLink,  
подобно на задаването в HTML

# Ограничения при единиците 1/2

- За общите текстови единици
  - Не могат да се използват рекурсивно
  - Могат да се ползват в елементно съдържание
    - <para> ... &ent; ... </para>
  - Допустими са в атриутно съдържание
    - <para name="&ent;"> ... </para>
  - Домустими са във вътрешни единици
    - <!ENTITY cod "&ent;">
  - Не могат да се ползват в други части на DTD

# Ограничения при единиците 2/2

- Двоично съдържание

- Ако съдържанието не е XML, такава единица не може да се ползва като референция

- Error - `<!ELEMENT sec (para | &photo; )>`
    - Error - `<para> &photo; </para>`

- Двоичните единици могат да бъдат само атрибут от тип ENTITY

- `<!ENTITY photo SYSTEM "photo.tif" NDATA TIFF>`  
...  
~~`<!ELEMENT pic (#PCDATA)>`~~  
~~`<!ATTLIST pic name ENTITY #REQUIRED>`~~

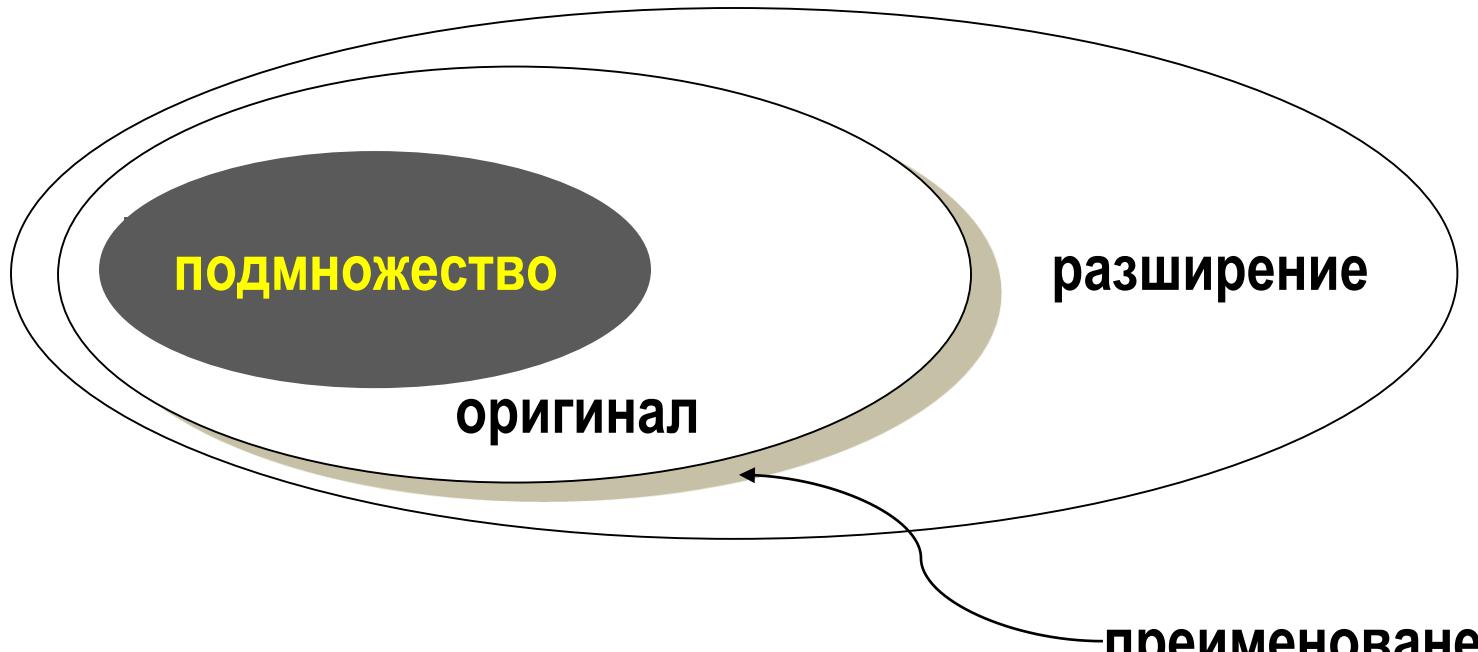
# Условни секции 1/2

- Единиците от параметричен тип могат успешно да бъдат използвани и при т. нар. условно DTD. То дава възможност в един DTD документ да има *условни* части, една от които е означена с ключовата дума **INCLUDE**, а останалите с **IGNORE**. В този случай от всички условни части на DTD за XML документа ще бъде валидна тази означена с **INCLUDE**.
  - `<! [INCLUDE [ ... ]]>`
  - `<! [IGNORE [ ... ]]>`
- Включване на декларации:
  - `<!ENTITY % variant "INCLUDE">`  
  
`<! [%variant; [`  
`<!ENTITY % Text "#PCDATA|temp")>`  
`]]>`

## Условни секции 2/2

- <!ENTITY % bigDTD "IGNORE">
- <!ENTITY % smallDTD "INCLUDE">
- :
- <![ %bigDTD; [
- <!ENTITY % blocks "para|excerpt|epigraph">
- ]]>
- <![ %smallDTD; [
- <!ENTITY % blocks "para|excerpt">
- ]]>
- :
- <![ %bigDTD; [
- <!ELEMENT epigraph (#PCDATA)>
- ]]>

# Проектиране на DTD с цел многократно използване



Оригинална дефиниция на атрибут:

```
<!ATTLIST document status CDATA #IMPLIED >
```

Подмножество:

```
<!ATTLIST document status (draft|final) #IMPLIED>
```

XML валидирани чрез DTD

XML

56

# Техники за управление на персонализация на дефинициите 1/2

- Работа с модулни DTD. Принципите (критериите) на Meyer за изграждане на модулни системи са приложими и тук
- Персонализирием модели на съдържанието - чрез начини за поставяне на контейнери в елементните декларации и в списъците атрибути
- Условно маркиране декларации - как да се използват маркираните секции с цел условно включване на декларации за маркиране

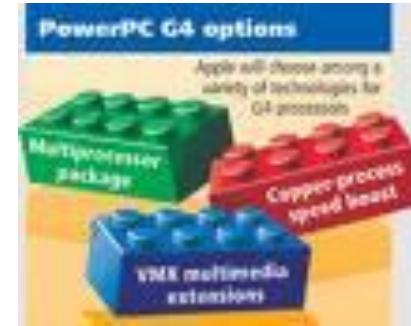
# Техники за управление на персонализация на дефинициите 2/2

- Работа с персонализирани имена за маркиране - чрез техники за персонализиране на имената на елементите и други имена, напр:

```
<!ENTITY % title "title">
<!ELEMENT %title (#PCDATA)>
<!ATTLIST %title id ID #IMPLIED >
...
<!ELEMENT div ((%title), para+, subdiv*)>
```

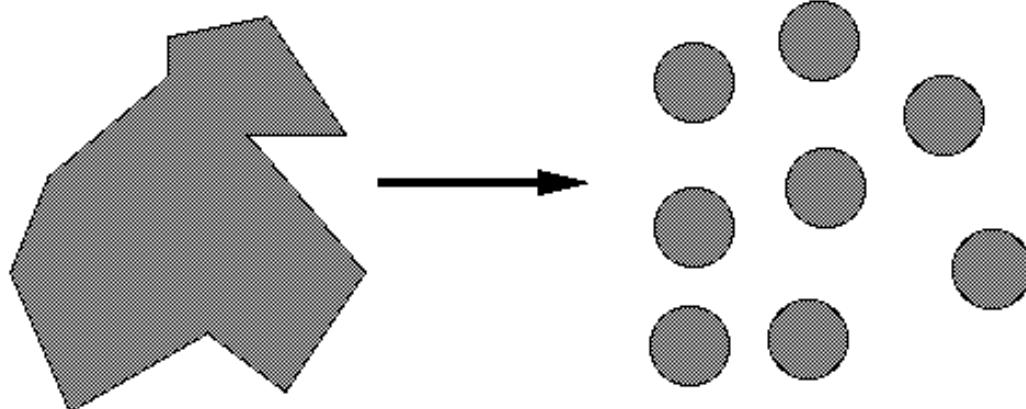
# Критерии на Meyer за оценка на системната модулност (модулни DTD)

- Декомпозируемост
- Композируемост
- Разбираемост
- Непрекъснатост  
(континюитет)
- Протекция



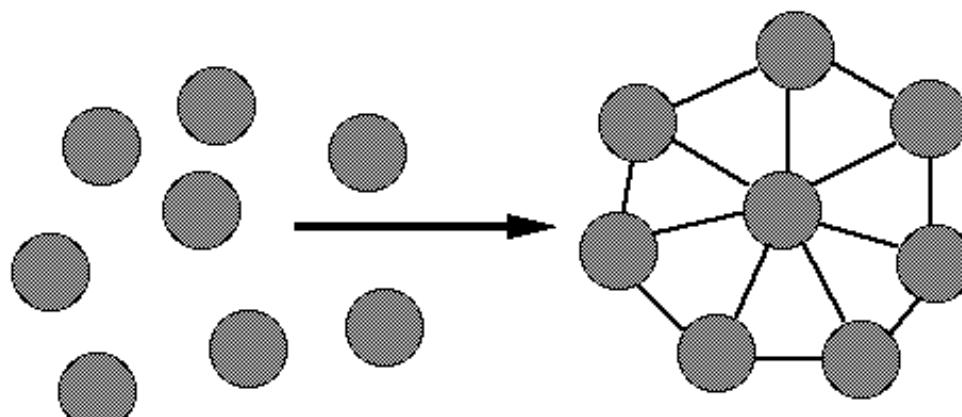
# Декомпозируемост (Decomposability)

- Идея: проблемът да се разложи на по-малки подпроблеми, които могат да бъдат решени отделно
- Пример: Top-Down дизайн
- Контрапример: Initialization модул



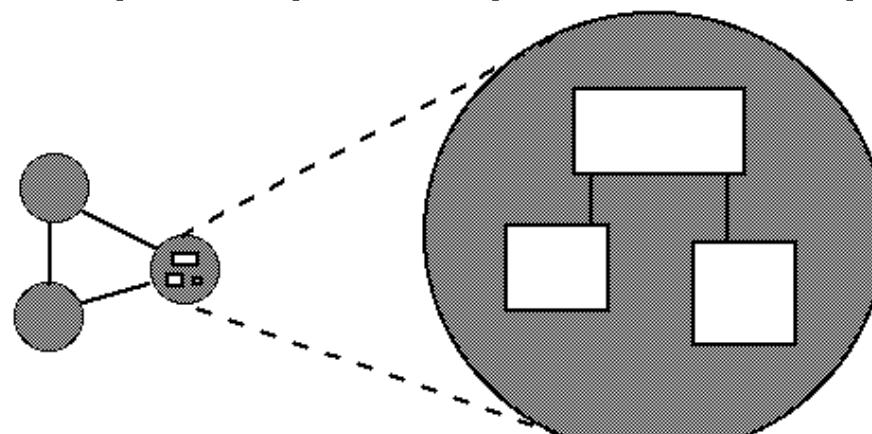
# Композируемост

- Идея: да се комбинират свободно модули за създаване на нови системи
- Пример: Math libraries, Unix command & pipes
- Контрапример: ?



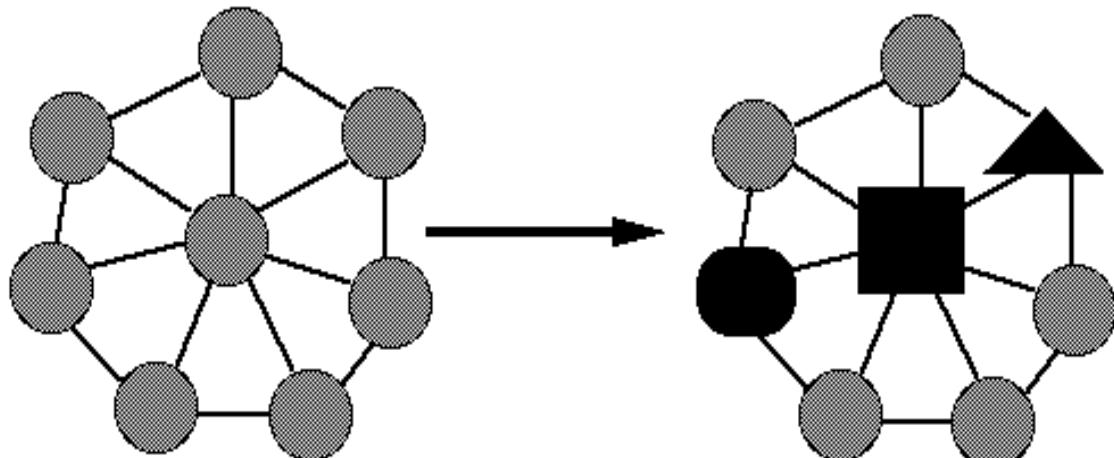
# Разбираемост

- Идея: отделните модули да бъдат разбираеми за читателя
- Пример: ?
- Контрапример: Sequential Dependencies



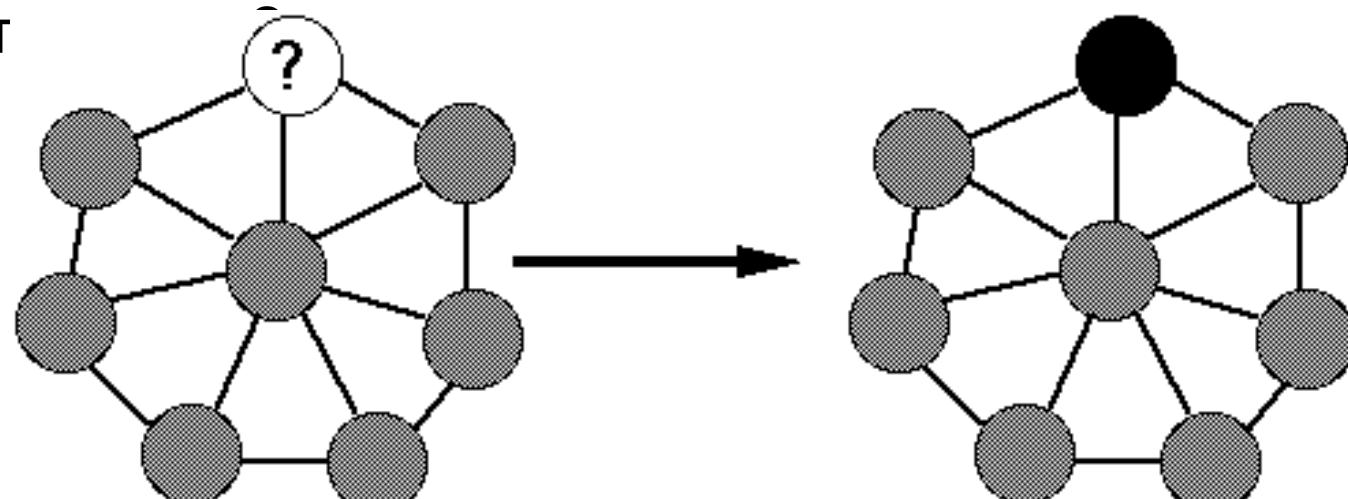
# Непрекъснатост (континюитет)

- Идея: малката промяна в спецификацията да има за резултат:
  - промени в само няколко модула
  - да не засяга архитектурата
- Пример: константи -> const MaxSize = 100
- Контрапример: ?



# Протекция

- Идея: влиянието на ненормално състояние по време на изпълнение се ограничава до няколко модули
- Пример: Validating input
- Контраг



# Два важни принципа за разработка на софтуер

KISS: Keep it *simple* & *stupid*

Поддържа:

- Разбираемост
- Композируемост
- Декомпозируемост

*Small* is Beautiful

*Def.: Upper bound for average size (lines of code) of an operation*

[Lorenz'93](#):

Smalltalk – 8

C++ - 24

Поддържа:

- Декомпозируемост
- Композируемост
- Разбираемост

# Web based e-Learning (free!)

- **W3 School: DTD School**
  - <http://www.w3schools.com/dtd/>
- **Introduction to DTD**
  - An introduction to the XML DTD, and why you should use it.
- **DTD - XML Building Blocks**
  - The XML building blocks that can be defined in a DTD.
- **DTD Elements**
  - How to define the legal elements of an XML document using DTD.
- **DTD Attributes**
  - How to define the legal attributes of XML elements using DTD.
- **DTD Entities**
  - How to define XML entities using DTD.
- **DTD Validation**
  - How to test for DTD errors while loading XML documents.
- **DTD Examples**
  - Some real world DTD examples.

# XML валидиране чрез XML Schema



Цели  
Структура на XML Schema  
Синтаксис  
Особености  
Примери

# Дефиниция на типа на документа (DTD)

- DTD дефиницията задава шаблон за маркиране на XML документ
- Форматът на DTD е наследен от SGML, като значително е опростен
- Както при SGML, така и XML DTD използва формална граматика за описание на структурата и типа на съдържанието на XML документа
- DTD осигурява начин за проверка на неговата структура и съдържание – т.е. за **валидация**

# XML Schema

- XML Schema е специализиран XML-базиран език за описание на XML документи
- XML Schema е на практика XML документ
- Всички правила, които важат за XML – за затваряне на таговете, за влагане и т.н., са в сила и за XML Schema
- XML Schema добавя допълнителни ограничения към правилата на XML - елементите в един XSD документ имат точно определени имена и са с точно определено значение

# DTD и XML schema

- DTD (Document Type Definition) и XML schema или XSD (XML Schema Definition) задават правила, съгласно които се определят имената на елементите и атрибутите, тяхната последователност, честота на срещане и др.
- DTD използва по-стегнат и кратък синтаксис в сравнение с XML Schema, но за сметка на това XML Schema предоставя по-богат набор от средства за по-строго дефиниране на структурата на XML и освен това нейните правила се задават в XML формат

# Валиден XML документ

- Всеки отделен документ, отговарящ на даден документен тип, е документен екземпляр (инстанция) на типа. Такъв документ представлява валиден документ за този документен тип.
- Всеки валиден документ е добре конструиран, но обратното не е задължително вярно.
- Всички XML парсери проверяват дали входния документ е добре конструиран XML документ.
- Парсери, които извършват още и проверка за определяне дали съдържанието на XML документите е валидно спрямо зададен тип на документа, се наричат валидиращи парсери.

# Валидация

- Валидацията е времеемък процес, но често спестява много проблеми на външните приложения и се извършва от специализиран процесор (валидиращ парсер)
- DTD описанието (ако е външно!) е споделено от валидизиращия парсер за XML документите - екземпляри на този документен тип – т.е. използва се многократно само едно описание

# Валидиране чрез XML схеми

- XML schema е XML документ, който описва структурата на друг XML документ.
- Тя е наследник на DTD, но предоставя много по-богати възможности за по-прецизна спецификация на XML структура.
- Има множество XML schema езици, като най-разпространените са:
  - **XML Schema** или наричан още XSD (XML Schema Definition), който се препоръчва от W3C консорциума, и
  - **RELAX NG**, дефиниран от OASIS.
- XML schema с малко **s** е общо име за такъв тип езици, а XML Schema с голямо **S** е конкретен XML schema език).

# Валидиране чрез XML схеми (сравнено с DTD) 1/3

- XML Schema използват XML-базиран синтаксис, докато DTD има специфичен синтаксис повлиян от SGML DTD.
- DTD използва по-стегнат и компактен синтаксис в сравнение с XML Schema.
- XML Schema дава възможност да се създават типове данни, които след това да се използват при специфицирането на елементи и атрибути. Върху тези типове данни могат да бъдат специфицирани различни ограничения като например минимална/максимална стойност, изброимо множество от стойности, стойности отговарящи на определен регулярен израз и др.

# Валидиране чрез XML схеми (сравнено с DTD) 2/3

- С DTD за разлика от XML Schema честотата на срещане на даден елемент не може да се зададе с произволно цяло число, а може да приема само стойностите – нула, едно, безкрайност.
- В DTD даден елемент не може да бъде асоцииран със избирам списък от стойности (например така:  
**<!ELEMENT score ("bad"|"good"|"high")>**), а това е валидно само за атрибутите (напр. **<!ATTLIST score value ("bad"|"good"|"high")>**). В XML Schema това е възможно, като се дефинира такъв тип данни и след това съответният елемент се асоциира с него.

# Валидиране чрез XML схеми (сравнено с DTD) 3/3

- XML Schema поддържа пространства от имена, докато DTD не осигурява такава поддръжка.
- В DTD моделът на съдържание на един елемент се определя изцяло от неговото име и така не може да се използва в различен контекст в зависимост от това къде се намира.
- В XML Schema всеки под-елемент на даден елемент може да се специфицира локално и така отделните под-елементи с едно и също име ще имат различен модел на съдържание.

# И все пак... ползваме DTD:

- ако е необходимо да бъде предефинирана дефиниция на даден елемент (например с използване на ключовите думи **INCLUDE** и **IGNORE**);
- когато съдържанието на елементите е най-вече текстово и не се налага дефинирането на собствени типове и налагането на ограничения;
- XML Schema, както и другите schema езици не предоставят заместващ механизъм за деклариране на **DTD entities**;
- ако е от значение големината на файла, дефиниращ структурата на XML документа.

# Програмни средства за работа със схема

- XML Schema-aware Parser
  - Xerces-J
  - Oracle XML Schema Processor
- XML Schema Validator (XSV, online)
- DTD to Schema Conversion Tools
- XML Schema Editor
  - Extensibility's XML Authority

# Възможности на XML Schema

- Структурни:
  - Пространства от имена
  - Интеграция на примитивни и комплексни типове данни
  - Ограничаване на даден тип данни
  - Наследяване
  - Интегритет
- Типове данни:
  - integers, dates, ... (както в езиците за прогр.)
  - user-defined (ограничаване на свойства)
- Използване:
  - Разлики с MS Schema: *XDR namespace (but not XSD), shown by URN (but not URL), ...*

# Използване на пространства от имена в XML Schema документ

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.myFirstXMLSchema.com"
 xmlns="http://www.myFirstXMLSchema.com"
 elementFormDefault="qualified">
 <xs:element name="country">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="total_area" type="xs:decimal"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

# Префиксът xs

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.myFirstXMLSchema.com"
 xmlns="http://www.myFirstXMLSchema.com"
 elementFormDefault="qualified">
 <xs:element name="country">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="total_area" type="xs:decimal"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

**xmlns:xs = http://www.w3.org/2001/XMLSchema**,  
указва че елементите и типовете данни на схемата  
идват от пространството от имена  
**http://www.w3.org/2001/XMLSchema** и те трябва да  
бъдат с префикс **xs**, т.е. префиксът на директивите на  
схема езика е **xs**. Така директивите на схема езика  
трябва да бъдат използвани по следния начин  
**xs:element**, **xs:attribute**, **xs:complexType** и т.н.

# Нуждата от targetNamespace

- Аргументът **targetNamespace** определя пространството от имена, за което тази схема е предназначена, а атрибутът **xmlns** определя, че точно **targetNamespace** ще бъде пространството от имена по подразбиране. За тази цел двета атрибути трябва да имат еднаква стойност.
- Пространството от имена дефинирано от **targetNamespace** няма префикс, но това не е задължително и може да има произволен такъв. Така с използване на **targetNamespace** когато дефинираме собствени типове данни те ще бъдат част от пространството специфицирано от него.
- Аргументът **elementFormDefault** декларира, че всички имена на елементи от XML екземплярите на тази схема трябва да бъдат асоцииирани с пространство от имена:
  - или чрез префикс,
  - или с пространството от имена по подразбиране.

# XML документ, асоцииран с предишната схема

```
<?xml version="1.0"?>
<xx:country xmlns:xx="http://www.myFirstXMLSchema.com"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.myFirstXMLSchema.com country.xsd">
 <xx:name>Bulgaria</xx:name>
 <xx:total_area>110993.7</xx:total_area>
</xx:country>
```

# XML Schema и xmlns

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.somewhere.org/BookCatalogue"
 xmlns:cat="http://www.somewhere.org/BookCatalogue">
 <element name="BookCatalogue">
 <annotation>
 <info>A book catalogue contains zero or more books</info>
 </annotation>
 <complexType>
 <sequence>
 <element ref="cat:Book" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
 </complexType>
 </element>
 <element name="Book">
 <annotation>
 <info>A Book has a Title, Author, Date, ISBN, and a Publisher</info>
 </annotation>
 <complexType>
 <sequence>
 <element ref="cat:Title"/>
 <element ref="cat:Author"/>
 <element ref="cat:Date"/>
 <element ref="cat:ISBN"/>
 <element ref="cat:Publisher"/>
 </sequence>
 </complexType>
 </element>
 <element name="Title" type="string"/>
 <element name="Author" type="string"/>
 <element name="Date" type="string"/>
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
</schema>
```

Референция  
(ref) – за пре-  
използване  
на типове

Елементите на  
схемата са с NS  
по подразбиране

Пространството  
cat е  
дeфинираното от  
схемата (target  
Namespace).

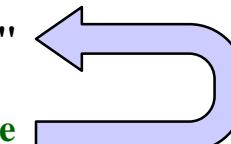
targetNameSpace  
съвпада с cat.

ref само към  
Global types –  
те са директни  
наследници на  
<schema>

# Рефериране на схема в XML документ-екземпляр

NS по  
подразбиране е  
targetNamespace

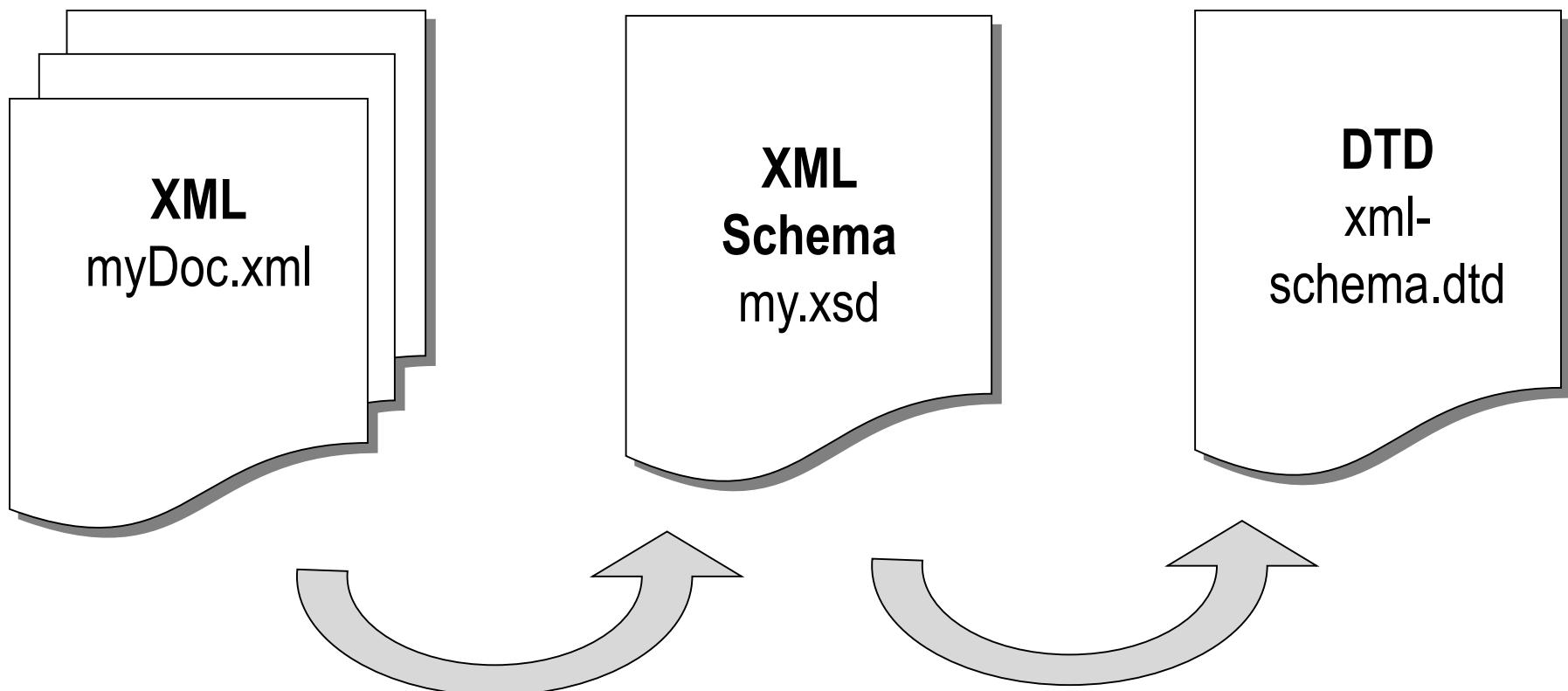
```
<?xml version="1.0"?>
<BookCatalogue xmlns ="http://www.somewhere.org/BookCatalogue"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.somewhere.org/BookCatalogue
 http://www.somewhere.org/BookCatalogue1.xsd">
 <Book>
 <Title>My Life and Times</Title>
 <Author>Paul McCartney</Author>
 <Date>July, 1998</Date>
 <ISBN>94303-12021-43892</ISBN>
 <Publisher>McMillin Publishing</Publisher>
 </Book>
 ...
</BookCatalogue>
```



schemaLocation е двойката {NS, XSD файл}

В елемента BookCatalogue декларираме, че атрибута **schemaLocation** идва от пространството XML Schema Instance namespace (xsi). Стойността на schemaLocation е двойката (namespace, URI към схемата). Когато XML парсерът обработва този XML документ, той използва двойката в schemaLocation, за да извлече файла със схемата (BookCatalogue.xsd) и да провери дали нейният targetNamespace отговаря на пространството в schemaLocation. **Пространството по подразбиране указва, че всички XML елементи идват от същото NS.**

# Валидиране на документ чрез XML Schema



# XML Schema спецификации

- Part 0: Primer

  - въведение

- Part 1: Structures

  - <http://www.w3.org/TR/xmlschema-1/>

  - структури

  - ограничения

- Part 2: Data types

  - типове данни за елементи и атрибути

  - <http://www.w3.org/TR/xmlschema-2/>

# Част 1: Структури

- **Type Definitions** `<simpleType> <complexType>`  
`<element> <group> <all> <choice> <sequence>`  
`<attribute> <attributeGroup>`
- **Attribute Declarations** `<attribute>`  
`<simpleType>`
- **Element Declarations** `<element>`  
`<simpleType> <complexType>`
- **Attribute Group Definitions** `<attributeGroup>`  
`<attribute> <attributeGroup>`
- **Model Group Definitions** `<group>`  
`<element> <group> <all> <choice> <sequence>`
- **Notation Declarations** `<notation>`
- **Annotations** `<annotation>`  
`<appinfo> <documentation>`

# Структура на DTD спрямо схема

- DTD
- ```
<!ELEMENT  
      e1  
((e2,e3?) +| e4)>
```
- Schema
 - ```
<element name="e1">
 <complexType>
 <choice>
 <sequence
 maxOccurs="unbounded">
 <element ref="e2"/>
 <element ref="e3"
 minOccurs="0"/>
 </sequence>
 <element ref="e4"/>
 </choice>
 </complexType>
</element>
```

# Интегритет по референции и уникалност

*Дефинираме ограничения (Constraints)  
чрез XPath изрази*

- <unique>
- <key>
- <keyref>
- <selector>
- <field>

# Част 2: Типове данни: <simpleType>

- Пространство на стойности (Value Space)
  - Множество стойности за даден тип данни
  - Дефинира аксиоматично примитивните типове
  - Изброими
  - Ограничими
  - Комбинации от стойности в списък (list)
  - Свойства (*cardinality, equality, ordered, ...*)
- Лексическо пространство
  - Освен value space, всеки тип данни има и lexical space.
  - Дефиниция: „A lexical space is the set of valid literals for a datatype.(e.g. 100 and 1.0E2 denote same value)“

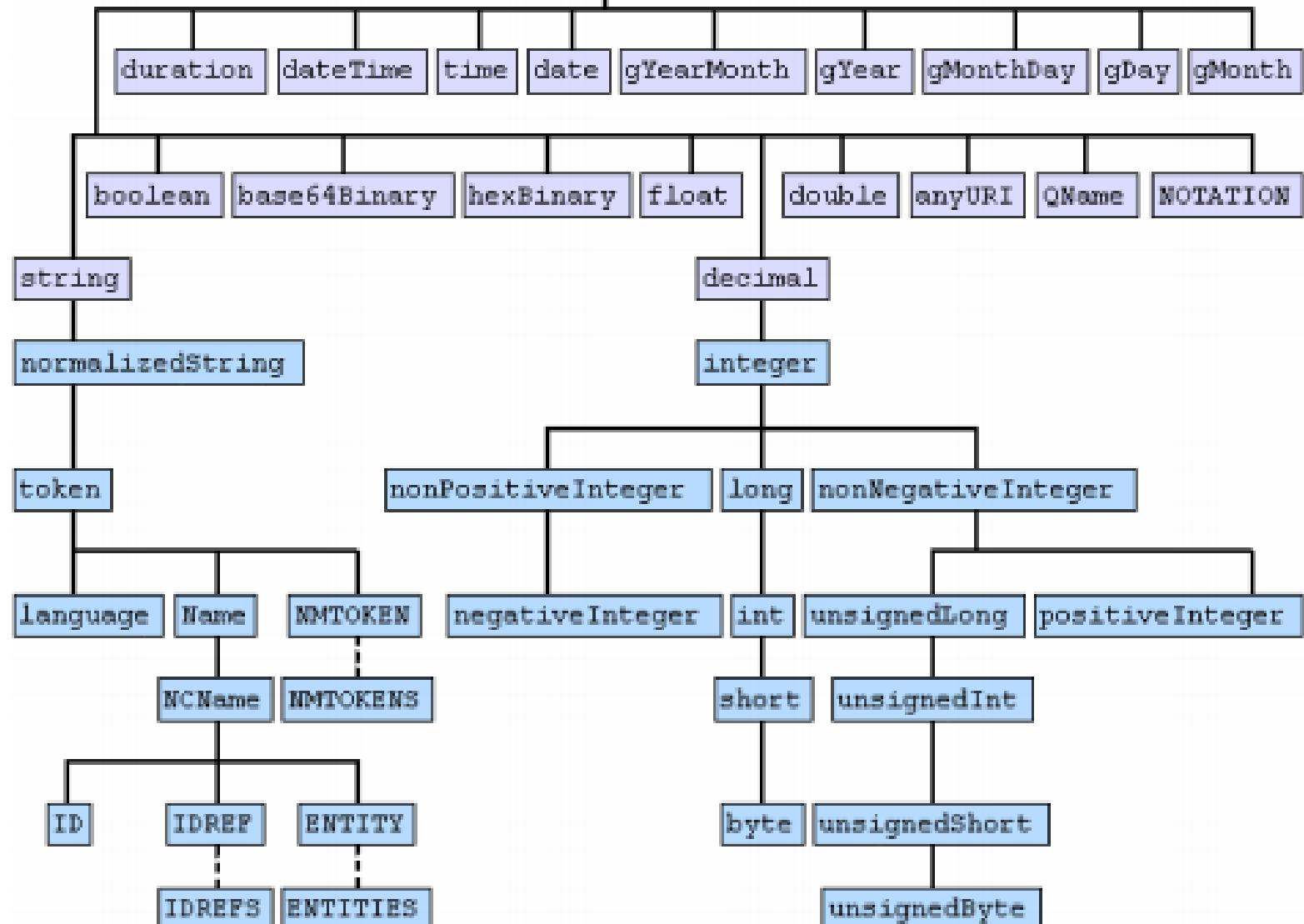
# Дефиниране на елементи от прост тип

- Елементите от прост тип в една XML Schema са елементите, които са от даден предефириран тип, и нямат атрибути и под-елементи:

```
<xs:element name="element_name" type="type_name"/>
```

- Атрибутът **name** задава името на елемента,
- Атрибутът **type** - името на предефинирания тип и може да приема една от стойностите: **float, double, decimal, integer, nonPositiveInteger, nonNegativeInteger, negativeInteger, positiveinteger, long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte, xs:time, dateTime, date, time, gDay, gMonth, gYear, gYearMonth, gMonthDay, duration, string, normal, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, anyURI, QName, NOTATION, hexBinary, base64Binary, boolean.**

## ПРОСТ ТИП



Вградени примитивни типове



Вградени производни типове



получаване с restriction



получаване с list

# Създаване на нови прости типове

Създаваме нови прости типове чрез Елементът `<xs:simpleType>` - на база на съществуващите, като обединяваме типове или като добавим ограничения над тях:

- Обединение (union) - чрез елемента `<xs:union />`:

```
<xs:simpleType name="intORfloat">
 <xs:union memberTypes="xs:integer xs:float" />
</xs:simpleType>
```

- Списък от елементи от даден прост тип, разделени с интервали - чрез елемента `<xs:list />`:

```
<xs:simpleType name="intList">
 <xs:list itemType="xs:integer" />
</xs:simpleType>
```

- Ограничение (restriction) – на следващите слайдове

# Ограничения върху стойността на елемент от прост тип

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.xmlschema.com"
 xmlns="http://www.xmlschema.com"
 elementFormDefault="qualified">
 <xs:element name="country">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="Bulgaria"/>
 <xs:enumeration value="Greece"/>
 <xs:enumeration value="Romania"/>
 </xs:restriction>
 </xs:simpleType>
 </xs:element>
 <xs:element name="area" type="xs:decimal"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

Новият тип наследява **xs:string**, и то с ограничения!

```
<country>
 <name>Bulgaria</name>
 <area>110879</area>
</country>
```

# Простите типове могат да дефинират списък от стойности

- <xsd:element name="cities" type="xsd:countryCities ">
- <xsd:simpleType name="countryCities">
- **<xsd:list>**
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
- <xsd:enumeration value="Sofia"/>
- <xsd:enumeration value="Plovdiv"/>
- <xsd:enumeration value="Varna"/>
- </xsd:restriction>
- </xsd:simpleType>
- **</xsd:list>**
- </xsd:simpleType>
- </xsd:element>
  
- Примерна инстанция:
- **<cities>Sofia Plovdiv</cities >**

XML валидиране чрез XML Schema

# Фасети (асекти)

- Дефиниция (<http://www.w3.org/TR/xmlschema-2/#facets>)
  - Фасет е определящ аспект на пространството от стойности за даден тип.
  - Фасетите на даден тип данни служат да се разграничават онези аспекти на този тип, които се различават от други типове данни.
- Видове фасети
  - Фундаментални фасети (дефинират типа)
  - Ограничаващи фасети (ограничават стойностите на типа)

# Фундаментални фасети

- Equal
  - Важи за всички типове данни
- Order
  - За някои типове
- Bounds
  - upper bound и lower bound
- Cardinality
  - finite, infinite
- Numeric
  - yes или no

# Ограничаващи фасети

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive /  
maxExclusive
- minInclusive /  
minExclusive
- precision
- scale
- encoding
- duration
- period

# Примитивни спрямо дериватни типове

## • А. Примитивни типове

- string
- boolean
- float
- double
- decimal
- timeDuration
- recurringDuration
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName

Съществуват *ab initio*

## • В. Дериватни типове

### ○ Чрез ограничения

- С използване на фасети

```
<simpleType name="sku"
 base="xsd:string">
 <pattern
 value="\d{3}-[A-D]{4}" />
 </simpleType>
```

- Чрез списъци →

# Вградени спрямо производни типове

- Вградени типове
  - А. примитивни
  - В. производни
    - language
    - IDREFS
    - long
    - int
    - short
    - positiveInteger
    - time
    - month
    - recurringDay
    - ...
- Потребителски типове
  - Само производни

# Атомарни спрямо списъчни типове

- Атомарни

- Неделими стойности

- ```
<simpleType  
name="ShoeSize"  
base="xsd:decimal"/>
```

- ```
<element name="shoe"
type="ShoeSize"/>
```

- ```
<shoe>10.5</shoe>
```

- Списъци

- Последователност от атомарни стойности

- ```
<simpleType
name="ShoeSizeS"
base="ShoeSize"
derivedBy="list"/>
```

- ```
<element name="shoes"  
type="ShoeSizes"/>
```

- ```
<shoes>8 10
10.5</shoes>
```

# Създаване на потребителски тип данни

```
<datatype name="PhoneNumber" source="string">
 <length value="8"/>
 <pattern value="\d{3}-\d{4}"/>
</datatype>
```

Новият тип 'PhoneNumber' се задава от осем-символен шаблон, съдържащ : ddd-dddd, където 'd' представля цифра ('digit').

# Фасети за тип данни Integer

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

```
<datatype name= "EarthSurfaceElevation"
source="integer">
 <minInclusive value="-1246"/>
 <maxInclusive value='29028'/>
</datatype>
```

# Начин за използване на фасети

```
<datatype name= "name" source= "source">
 <facet value= "value"/>
 <facet value= "value"/>
 ...
</datatype>
```

## Facets:

- minInclusive
- maxInclusive
- minExclusive
- maxExclusive
- length
- minlength
- maxlength
- pattern
- enumeration

...

## Sources:

- string
- boolean
- float
- double
- decimal
- timeInstant
- timeDuration
- recurringInstant
- binary
- uri

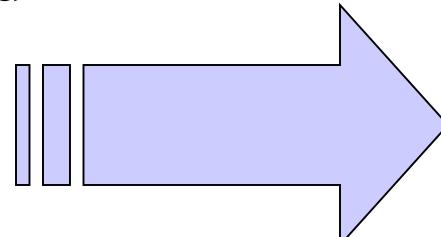
...

# Регулярни изрази 1/2

- Стойността на pattern facet е регулярен израз:

## *Regular Expression*

- Chapter \d
- a\*b
- [xyz]b
- a?b
- a+b
- [a-c]x



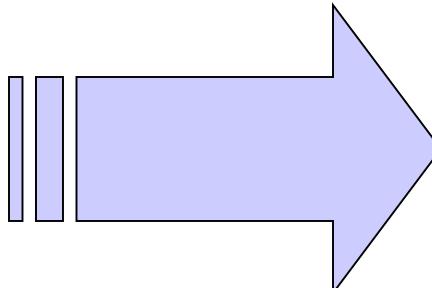
## *Example*

- Chapter 1
- b, ab, aab, aaab, ...
- xb, yb, zb
- b, ab
- ab, aab, aaab, ...
- ax, bx, cx

# Регулярни изрази 2/2

## • Регулярен израз

- [a-c]x
- [-ac]x
- [ac-]x
- [^0-9]x
- \Dx
- Chapter\s\d
- (ho){2} there
- (ho\s){2} there
- .abc
- (a|b)+x



## • Пример

- ax, bx, cx
- x, ax, cx
- ax, cx, -x
- any non-digit char followed by x
- any non-digit char followed by x
- Chapter followed by blank followed by digit
- hoho there
- ho ho there
- any char followed by abc
- ax, bx, aax, bbx, abx, bax, ...

# Регулярен израз за 0..255

**[1-9]?[0-9] | 1[0-9][0-9] |2[0-4][0-9] | 25[0-5]**

The regular expression is divided into four segments by vertical lines under each character group: [1-9]?[0-9], 1[0-9][0-9], 2[0-4][0-9], and 25[0-5]. Below these segments are the corresponding numerical ranges: 0 to 99, 100 to 199, 200 to 249, and 250 to 255.

# IP тип данни

```
<datatype name="IP" source="string">
 <pattern value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
 ([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])">
 <annotation>
 <info>
 Datatype for representing IP addresses. Examples,
 129.83.64.255, 64.128.2.71, etc.
 This datatype restricts each field of the IP address
 to have a value between zero and 255, i.e.,
 [0-255].[0-255].[0-255].[0-255]
 Note: in the value attribute (above) the regular
 expression has been split over two lines. This is
 for readability purposes only. In practice the R.E.
 would all be on one line.
 </info>
 </annotation>
</pattern>
</datatype>
```

# Сложни типове

Сложни елементите са тези, асоциирани с т.нар. сложен тип и могат да бъдат празни или да съдържат под-елементи и атрибути. За целта съответният сложен тип трябва да бъде дефиниран и именован извън елемента или да бъде специфициран като под-елемент.

```
<xs:element name="city">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="population" type="xs:integer"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

# Сложен тип, дефиниран извън елемента

Сложен тип, дефиниран извън елемента, подобрява четимостта на XML Schema документа:

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
 <xs:sequence>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="age" type="xs:integer"/>
 </xs:sequence>
</xs:complexType>
```

# Видове елементи от сложен тип

- празни елементи – те са без съдържание, под-елементи и атрибути подобно на html элемента **<br/>**.
- елементи с атрибути – те имат атрибути и могат да имат също текстово съдържание, но нямат под-елементи.
- елементи съдържащи под-елементи - те имат под-елементи и могат да имат също текстово съдържание, но нямат атрибути.
- елементи с текстово съдържание и атрибути – те имат както атрибути, така и под-елементи, а също така може да имат текстово съдържание.

# Елементи от тип **simpleContent**

- XML Schema елементът с име **simpleContent** задава разширения или ограничения върху:
  - complex type, който е само текст
  - simple type
- XML елементът от тип **simpleContent** не съдържа елементи, но може да съдържа атрибути

# Елементи с атрибути

```
<xs:element name="population">
 <xs:complexType>
 <xs:simpleContent>
 <xs:extension base="xs:integer">
 <xs:attribute name="country" type="xs:string"
 use="required"/>
 <xs:attribute name="year" type="xs:integer"/>
 </xs:extension>
 </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

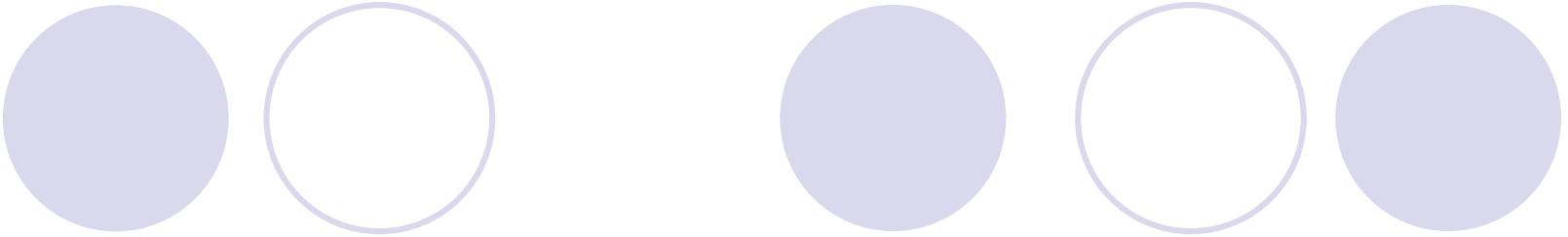
XML документът ще има елемент **population**, който ще изглежда така:

```
<population country="bulgaria" year="2011"> 7500441
</population>
```

# Производни типове

- Производни типове ("derived types") се създават чрез наследяване по два начина:
  - derive by **extension**: разширяване на родителския тип с повече елементи
  - derive by **restriction**: ограничаване на родителския тип посредством:
    - **по-ограничен обхват на стойности (range of values), и/или**
    - **по-ограничен брой на повторения.**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.somewhere.org/BookCatalogue"
 xmlns:cat="http://www.somewhere.org/BookCatalogue">
 <complexType name="Publication">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs=" unbounded "/>
 <element name="Date" type="date"/>
 </complexType>
 <complexType name="Book" source="cat:Publication" derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 </complexType>
 <element name="BookCatalogue">
 <complexType>
 <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded"
type="cat:Book" />
 </complexType>
 </element>
</schema>
```



```
<complexType name="Publication">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs=" unbounded " />
 <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
</complexType>
```

Като резултат:

*Елементите от тип Book ще имат 5 наследника – Title, Author, Date, ISBN, and Publisher.*

# Използване на дериватни типове

- Ако декларираме даден тип да бъде *Publication*,
- тогава в XML документа-екземпляр неговото съдържание може да бъде
- или *Publication*, или *Book*
- (понеже *Book* е *Publication*).

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
 <!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.somewhere.org/BookCatalogue"
 xmlns:cat="http://www.somewhere.org/BookCatalogue">
 <complexType name="Publication">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs=" unbounded "/>
 <element name="Date" type="date"/>
 </complexType>
 <complexType name="Book" source="cat:Publication"
 derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 </complexType>
 <element name="BookCatalogue">
 <complexType>
 <element name="CatalogueEntry" minOccurs="0"
 maxOccurs="unbounded" type="cat:Publication"/>
 </complexType>
 </element>
</schema>
```

BookCatalogue6.xsd

```
<?xml version="1.0"?>
<Catalogue xmlns ="http://www.somewhere.org/Catalogue"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.somewhere.org/Catalogue
 http://www.somewhere.org/Catalogue/BookCatalogue6.xsd">
 <CatalogueEntry>
 <Title>Staying Young Forever</Title>
 <Author>Karin Granstrom Jordan, M.D.</Author>
 <Date>December, 1999</Date>
 </CatalogueEntry>
 <CatalogueEntry xsi:type="Book">
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34319-4</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
 </CatalogueEntry>
 <CatalogueEntry xsi:type="Book">
 <Title>The First and Last Freedom</Title>
 <Author>J. Krishnamurti</Author>
 <Date>1954</Date>
 <ISBN>0-06-064831-7</ISBN>
 <Publisher>Harper & Row</Publisher>
 </CatalogueEntry>
</Catalogue>
```

The diagram illustrates the hierarchical structure of the XML document. At the top level is a light blue cloud shape containing the word "Publication". Below it is another light blue cloud shape containing the word "Book". A vertical line connects the "Publication" cloud to the "Book" cloud. To the left of the "Publication" cloud, there is a vertical line connecting the "Catalogue" element to the "Publication" element. There are also three small blue circles scattered around the clouds.

BookCatalogue2.xml

```
<CatalogueEntry xsi:type="Book">
 <Title>Illusions The Adventures of a Reluctant Messiah
 </Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34319-4</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>
```

- Нека елементът CatalogueEntry е от тип Publication. Book е дериват на Publication. Следователно, Book е Publication. Ето защо съдържанието на CatalogueEntry може да е Book.
- За да укажем, че съдържанието е от дериватен тип, трябва да зададем от кой точно дериватен тип е то (така избягваме възможни нееднозначности!)
- Атрибутът '**type**' идва от XML Schema Instance (**xsi namespace**)

# Деривация чрез рестрикция

```
<complexType name="Publication">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs="unbounded"/>
 <element name="Date" type="date"/>
</complexType>
<complexType name="SingleAuthorPublication" source="cat:Publication"
derivedBy="restriction">
 <restrictions>
 <element name="Author" type="string" maxOccurs="1"/>
 </restrictions>
</complexType>
```

Елементите от типа SingleAuthorPublication ще имат 3 дъщерни елемента – Title, Author, and Date.

Трябва да съществува точно един елемент Author.

# Деривация чрез рестрикция – простирайт тип *restriction*

```
<attribute name="Title">
 <simpleType>
 <restriction base="string">
 <enumeration value="Sir"/>
 <enumeration value="Dr."/>
 <enumeration value="Mr."/>
 </restriction>
 </ simpleType >
</attribute >
```

# Ограничаване на деривациите

Можем да създадем нов тип и да:

- Забраним всички деривации от него, или
- Забраним само рестрикциите от него, или
- Забраним само разширенията от него.

```
<type name="Publication" final="#all" ...>
```

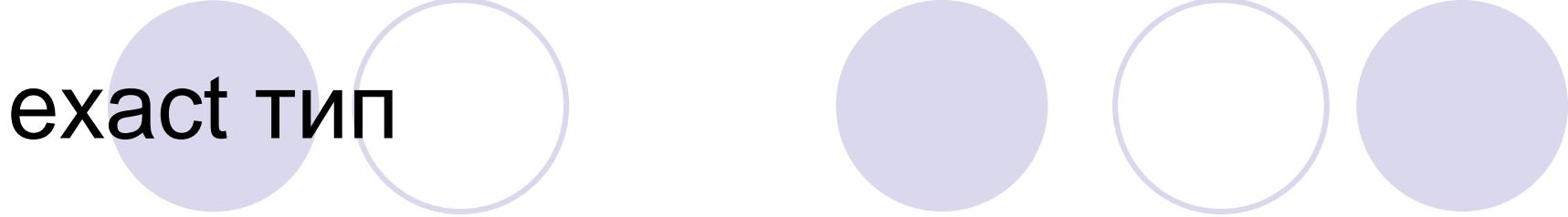
This type cannot be extended nor restricted

```
<type name="Publication" final="restriction" ...>
```

This type cannot be restricted

```
<type name="Publication" final="extended" ...>
```

This type cannot be extended



**exact тип**

- Даден тип е **exact**, ако:
  - Други типове могат да са негови деривати.
  - В документа-екземпляр на схемата дериватните типове не могат да се ползват вместо този exact тип.

Схема:

```

<complexType name="Publication">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs="unbounded"/>
 <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
</complexType>
<element name="Catalogue">
 <complexType>
 <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded" type="cat:Publication"/>
 </complexType>
</element>

```

**Това разрешава елементи от тип Publication, както и техните дериватни типове като например Book, да се ползват като деца на Catalogue, напр.**

Документ-екземпляр:

```

<CatalogueEntry xsi:type="Book">
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34319-4</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>

```

Схема:

```
<complexType name="Publication" exact="extension">
 <element name="Title" type="string" maxOccurs="unbounded"/>
 <element name="Author" type="string" maxOccurs="unbounded"/>
 <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
</complexType>
<element name="Catalogue">
 <complexType>
 <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded" type="cat:Publication"/>
 </complexType>
</element>
```

*Това забранява използването на дериватни типове, разширяващи Publication, като деца на CatalogueEntry, напр.*

Екземпляр:

```
<CatalogueEntry xsi:type="Book">
 <Title>Illusions of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34311-5</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>
```

# exact типове

- **exact="extension"**

- Забранява **използването** на дериватни типове чрез разширение вместо дадения тип в документ – екземпляр на схемата

- **exact="restriction"**

- Забранява **използването** на дериватни типове чрез рестрикция вместо дадения тип в документ – екземпляр на схемата

- **exact="#all"**

- Забранява **използването** на всякакви дериватни типове вместо дадения тип

# Еквивалентност

- Често в ежедневното общуване изразяваме нещо по няколко начина:
  - В Бостън думите "T" (от Tube) и "subway" са взаимозаменяеми и означават метро:
    - "we took the T into town"
    - "we took the subway into town".
  - Така "T" и "subway" са еквивалентни
- Свойството еквивалентност (equivalence) може да се представи в XML Schema.
  - Желаем да декларираме еквивалентност между елемента "subway" и елемента "T", така че в документите-екземпляри на схемата да можем да ползваме "subway" или "T", според предпочтенията ни.

# equivClass

- Създаваме елемента **subway** (наричан *exemplar*) и след него други елементи, които са еквивалентни (equivalent) на екземпляра.

```
<element name="subway" type="string"/>
<element name="T" equivClass="boston:subway"
 type="string"/>
```

- **subway** : exemplar
- **T** : equivalent.

*Можем да използваме екземпляра и  
еквивалентите му взаимозаменямо.*

Schema:

```
<element name="subway" type="string"/>
<element name="T" equivClass="boston:subway" type="string"/>
<element name="transportation">
 <simpleType>
 <element ref="boston:subway"/>
 </simpleType>
</element>
```

Instance doc:

```
<transportation>
 <subway>Red Line</subway>
</transportation>
```

Alternative  
Instance doc:

```
<transportation>
 <T>Red Line</T>
</transportation>
```

Заб.: **тиปът на всеки еквивалентен елемент трябва да бъде същият като типа на елемента-екземпляр, или негов дериват.**

# Подпомагане на оперативния обмен на данни (Interoperability) 1/2

- Софтуерни приложения от различни приложни области (domains) трябва да си комуникират ефективно независимо от различията помежду им
- equivClass е начална стъпка в реализацията на оперативния обмен на данни (interoperability).
  - Експлицитна еквивалентност между елементи

# Подпомагане на оперативния обмен на данни (Interoperability) 2/2

- Софтуерно приложение очаква да открие елемента `<subway>`.
- В даден XML документ-екземпляр то открива елемента `<T>`.
- Приложението прочита декларациите в XML Schema и установява, че `<T>` е еквивалент на `<subway>` => то приема този XML документ, макар че той е с различен речник (*vocabulary*) от очаквания.
- Добавяме и нов елемент към схемата напр.  
`<element name="train" equivClass="boston:subway" type="string"/>`
- Без промяна на приложението, то ще може да обработва XML документи с елемента `<train>`.
- Mike Los предлага термина "**interoperability schema**" с цел да опише тези възможности.

# Abstract елементи

- Даден елемент може да се декларира като абстрактен, напр.:  
`<element name="Publication" type="cat:Pub"  
abstract="true"/>`
- Абстрактният елемент е вид контейнер или шаблон.
- Той не може да присъства в XML документа-екземпляр:
  - Напр., `<Publication>` не може да бъде използван.
- На негово място обаче могат да се ползват елементи, които са му еквивалентни.

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Catalogue"
 xmlns:cat="http://www.somewhere.org/Catalogue">
 <complexType name="Pub">
 <element name="Title" type="string" maxOccurs="*"/>
 <element name="Author" type="string" maxOccurs="*"/>
 <element name="Date" type="date"/>
 </complexType>
 <element name="Publication" type="cat:Pub" abstract="true"/>
 <element name="Book" equivClass="cat:Publication">
 <complexType source="cat:Pub" derivedBy="extension">
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 </complexType>
 </element>
 <element name="Magazine" equivClass="cat:Publication">
 <complexType source="cat:Pub" derivedBy="restriction">
 <restrictions>
 <element name="Author" type="string" maxOccurs="0"/>
 </restrictions>
 </complexType>
 </element>
 <element name="Catalogue">
 <complexType>
 <element ref="cat:Publication" minOccurs="0" maxOccurs="unbounded"/>
 </complexType>
 </element>
</schema>

```

Елементите Book и Magazine са еквивалентни на Publication.

Понеже Publication е абстрактен, то само негови еквиваленти могат да бъдат деца на Catalogue.

# XML документ-екземпляр на предната схема

```
<?xml version="1.0"?>
<Catalogue xmlns = "http://www.somewhere.org/Catalogue"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.somewhere.org/Catalogue
 http://www.somewhere.org/Catalogue/BookCatalogue7.xsd">
 <Magazine>
 <Title>Natural Health</Title>
 <Date>December, 1999</Date>
 </Magazine>
 <Book>
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34319-4</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
 </Book>
 <Book>
 <Title>The First and Last Freedom</Title>
 <Author>J. Krishnamurti</Author>
 <Date>1954</Date>
 <ISBN>0-06-064831-7</ISBN>
 <Publisher>Harper & Row</Publisher>
 </Book>
</Catalogue>
```

# Атрибути

- Използване в BookCatalogue DTD.
- ... и в XML Schema....

```
<!-- A book catalogue contains zero or more books -->
<!ELEMENT BookCatalogue (Book)*>
<!-- A Book has a Title, one or more Authors, a Date, an ISBN, and a Publisher -->
<!ELEMENT Book (Title, Author+, Date, ISBN, Publisher)>
<!-- A Book has three attributes - Category, InStock, and Reviewer. Category must be
either "autobiography", "non-fiction", or "fiction". A value must be supplied for this
attribute whenever a Book element is used within a document. InStock can be either
"true" or "false". If no value is supplied it defaults to "false". Reviewer contains the
name of the reviewer. It defaults to "" if no value is supplied -->
```

## **<!ATTLIST Book**

```
Category (autobiography | non-fiction | fiction) #REQUIRED
InStock (true | false) "false"
Reviewer CDATA "">
```

```
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!-- A Date may have a Month. It must have a Year. -->
<!ELEMENT Date (Month?, Year)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
```

[**Дефиниция:** Ако декларацията не е  
нито #REQUIRED, нито #IMPLIED,  
тогава [AttValue](#) съдържа стойността  
по подразбиране]

BookCatalogue2.**dtd**

```
.....
<element name="BookCatalogue">
 <annotation>
 <info>A book catalogue contains zero or more books</info>
 </annotation>
 <complexType>
 <element ref="cat:Book" minOccurs="0" maxOccurs="*"/>
 </complexType>
</element>
<element name="Book">
 <annotation>
 <info>A Book has a Title, one or more Authors, a Date, an ISBN, and a Publ.</info>
 </annotation>
 <complexType>
 <element ref="cat:Title"/>
 <element ref="cat:Author" maxOccurs="unbounded"/>
 <element ref="cat:Date"/>
 <element ref="cat:ISBN"/>
 <element ref="cat:Publisher"/>
 <attributeGroup ref="BookAttributes"/>
 </complexType>
</element>
```

```
<attributeGroup name="BookAttributes">
 <annotation>
 <info>
 A Book has three attributes - Category, InStock, and Reviewer.
 Category must be either "autobiography", "non-fiction",
 or "fiction". A value must be supplied for this attribute
 whenever a Book element is used within a document. InStock can
 be either "yes" or "no". If no value is supplied it defaults
 to "no". Reviewer contains the name of the reviewer. It defaults
 to "" if no value is supplied.
 </info>
 </annotation>
 <attribute name="Category" minOccurs="1">
 <datatype source="string">
 <enumeration value="autobiography"/>
 <enumeration value="non-fiction"/>
 <enumeration value="fiction"/>
 </datatype>
 </attribute>
 <attribute name="InStock" type="boolean" default="false"/>
 <attribute name="Reviewer" type="string" default="" />
</attributeGroup>
<element name="Title" type="string"/>
<element name="Author" type="string"/>
<element name="Date" type="string"/>
<element name="ISBN" type="string"/>
<element name="Publisher" type="string" />
</schema>
```

```
<attribute name="Category" minOccurs="1">
 <datatype source="string">
 <enumeration value="autobiography"/>
 <enumeration value="non-fiction"/>
 <enumeration value="fiction"/>
 </datatype>
</attribute>
```

“По подразбиране стойността на  
maxOccurs е 1. За minOccurs="1"  
=> такъв атрибут е **REQUIRED**.”

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/BookCatalogue"
 xmlns:cat="http://www.somewhere.org/BookCatalogue">
 <element name="BookCatalogue">
 <complexType>
 <element name="Book" minOccurs="0" maxOccurs="*"/>
 <complexType>
 <element name="Title" type="string"/>
 <element name="Author" type="string"/>
 <element name="Date" type="string"/>
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 <attribute name="Category" minOccurs="1">
 <datatype source="string">
 <enumeration value="autobiography"/>
 <enumeration value="non-fiction"/>
 <enumeration value="fiction"/>
 </datatype>
 </attribute>
 <attribute name="InStock" type="boolean" default="false"/>
 <attribute name="Reviewer" type="string" default="" />
 </complexType>
 </element>
 </complexType>
</element>
</schema>

```

*Алтернативна схема*

# Още за атрибути

- ДЕКЛАРАЦИИТЕ НА АТРИБУТИТЕ ВИНАГИ СА СЛЕД ТЕЗИ НА ЕЛЕМЕНТИТЕ, ЗА КОИТО ТЕ СЕ ДЕКЛАРИРАТ.

# Още примери:

<http://www.zvon.org/xxl/XMLSchemaTutorial/Output/highlights.html>

Let's say we want to define a group of common attributes, which will be reused. The root element is named "root", it must contain the "aaa" and "bbb" elements, and these elements must have attributes "x" and "y".

## Valid document

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <aaa x="1" y="2"/>
 <bbb x="3" y="4"/>
</root>
```

## Invalid document

Attribute "x" is missing.

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <aaa y="2"/>
 <bbb x="3" y="4"/>
</root>
```

## Invalid document

Attribute "z" is not allowed.

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <aaa x="1" y="2" z="2"/>
 <bbb x="3" y="4"/>
</root>
```

## Correct XML Schema (correct\_0.xsd)

Do not forget to add the attribute "use" set to "required" when declaring attribute (default value is "optional").

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="root">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element name="aaa" minOccurs="1" maxOccurs="1">
 <xsd:complexType>
 <xsd:attributeGroup ref="myAttrs"/>
 </xsd:complexType>
 </xsd:element>
 <xsd:element name="bbb" minOccurs="1" maxOccurs="1">
 <xsd:complexType>
 <xsd:attributeGroup ref="myAttrs"/>
 </xsd:complexType>
 </xsd:element>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>

 <xsd:attributeGroup name="myAttrs">
 <xsd:attribute name="x" type="xsd:integer" use="required"/>
 <xsd:attribute name="y" type="xsd:integer" use="required"/>
 </xsd:attributeGroup>
</xsd:schema>
```

# Елемент group

- Елементът **group** ни позволява да групирате заедно декларации на елементи.
- Заб.: елементът group разрешава многократно използване на елементни декларации, но без атрибути в тях.

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd" [
<!ATTLIST schema xmlns:cat CDATA #IMPLIED]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/BookCatalogue"
 xmlns:cat="http://www.somewhere.org/BookCatalogue">
<element name="BookCatalogue">
 <complexType>
 <element name="Book" minOccurs="0" maxOccurs="*"
 <complexType>
 <group ref="cat:BookElements"/>
 <attribute name="Category" minOccurs="1">
 <datatype source="string">
 <enumeration value="autobiography"/>
 <enumeration value="non-fiction"/>
 <enumeration value="fiction"/>
 </datatype>
 </attribute>
 <attribute name="InStock" type="boolean" default="false"/>
 <attribute name="Reviewer" type="string" default="" />
 </complexType>
 </element>
 </complexType>
</element>
<group name="BookElements" order="seq">
 <element name="Title" type="string"/>
 <element name="Author" type="string"/>
 <element name="Date" type="string"/>
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
</group>
</schema>

```

# Задаване на алтернативи

DTD: <!ELEMENT signature (name | (name, date))>

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:sig CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples"
 xmlns:sig="http://www.somewhere.org/Examples">
 <element name="signature">
 <complexType>
 <group order="choice">
 <element ref="sig:name"/>
 <group order="seq">
 <element ref="sig:name"/>
 <element name="date" type="date"/>
 </group>
 </group>
 </complexType>
 </element>
 <element name="name" type="string"/>
</schema>
```

XML Schema:

# Задаване на повторения

DTD:

```
<!ELEMENT binary-digit (zero | one)+>
```

XML Schema:

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd" [
<!ATTLIST schema xmlns:bi CDATA #IMPLIED]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples"
 xmlns:bi="http://www.somewhere.org/Examples">
 <element name="binary-digit">
 <complexType>
 <group order="choice" maxOccurs="unbounded">
 <element name="zero" type="bi:zero-digit"/>
 <element name="one" type="bi:one-digit"/>
 </group>
 </complexType>
 </element>
 <datatype name="zero-digit" source="string">
 <enumeration value="0"/>
 </datatype>
 <datatype name="one-digit" source="string">
 <enumeration value="1"/>
 </datatype>
</schema>
```

# Задаване на произволно редуване (Any Order)

Проблем: елементът Book да съдържа Author, Title, Date, ISBN, and Publisher,  
*в any order* (как беше в *DTD?!*).



XML Schema:

***order="all"*** -> Book трябва да съдържа петте елемента, но в произволен ред.  
Заб.: *minOccurs, maxOccurs* с **ФИКСИРАНИ СЪС СТОЙНОСТ "1"**.

# Пример

Елементът Book да съдържа Author, Title, Date, ISBN, and Publisher,  
*в any order*

XML Schema:

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples">
 <element name="Book">
 <type>
 <group order="all">
 <element name="Title" type="string"/>
 <element name="Author" type="string"/>
 <element name="Date" type="date"/>
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 </group>
 </type>
 </element>
</schema>
```

# Празен елемент

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples">
 <element name="image">
 <simpleType content="empty">
 <attribute name="href" type="uri"/>
 </simpleType>
 </element>
</schema>
```

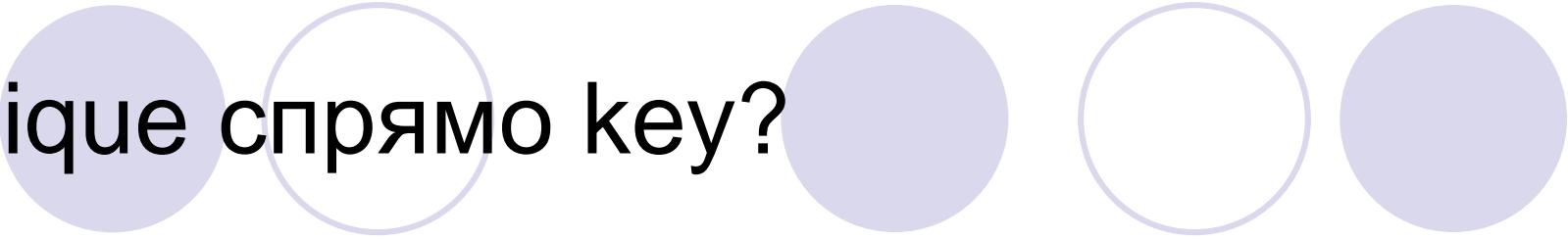
Schema:

Instance  
doc:

```
<image href="http://www.xfront.org/Rog.gif"/>
```

# Уникалност

- DTD предоставя атриутния тип данни **ID** с цел деклариране на уникалност (дадена *ID* атриутна стойност трябва да бъде уникална за целия документ и XML парсерът следи за това).
- В XML Schema можем да дефинираме много повече уникалност (*uniqueness*) на съдържанието:
  1. уникално съдържание на елемент.
  2. non-ID атрибути да бъдат уникални.
  3. комбинация между елементи и атрубити да бъде уникална.
  4. разлика между *unique* и *key*.
  5. обхват в документа, за който нещо е *unique*.



# *unique* спрямо *key*?

- Key: елемент или атрибут (или комбинация от тях), който е деклариран като ключ (*key*), трябва:
  - Винаги да е наличен (*minOccurs* да е поголямо от 0)
  - Да не може да е null (тоест *nullable="false"*)
  - Да е уникален (*unique*)
- Key предполага *unique*, но *unique* не предполага непременно key

```

<?xml version="1.0"?>
<Library xmlns="http://www.somewhere.org/Library">
 <BookCatalogue>
 <Book Category = "fiction" InStock = "yes" Reviewer = "John Doe">
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN>0-440-34319-4</ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
 </Book>
 <Book Category = "non-fiction" InStock = "yes" Reviewer = "John Doe">
 <Title>The First and Last Freedom</Title>
 <Author>J. Krishnamurti</Author>
 <Date>1954</Date>
 <ISBN>0-06-064831-7</ISBN>
 <Publisher>Harper & Row</Publisher>
 </Book>
 </BookCatalogue>
 <CheckoutRegister>
 <Person>Roger Costello</Person>
 <Book titleRef="Illusions The Adventures of a Reluctant Messiah"
 categoryRef="fiction"/>
 </CheckoutRegister>
</Library>

```

+ = key

+ = keyref

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:lib CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Library"
 xmlns:lib="http://www.somewhere.org/Library">
 <element name="Library">
 <complexType>
 <element name="BookCatalogue">
 <complexType>
 <element name="Book" minOccurs="0" maxOccurs="unbounded">
 <complexType>
 <element name="Title" type="string"/>
 <element name="Author" type="string"/>
 <element name="Date" type="string"/>
 <element name="ISBN" type="string"/>
 <element name="Publisher" type="string"/>
 <attribute name="Category" minOccurs="1">
 <datatype source="string">
 <enumeration value="autobiography"/>
 <enumeration value="non-fiction"/>
 <enumeration value="fiction"/>
 </datatype>
 </attribute>
 <attribute name="InStock" type="boolean" default="false"/>
 <attribute name="Reviewer" type="string" default="" />
 </complexType>
 </element>
 </complexType>
 </element>
 </complexType>
 </element>
</schema>
```

The default value for both the minOccurs and the maxOccurs attributes is 1.

```
<element name="CheckoutRegister">
 <type>
 <element name="Person" type="string"/>
 <element name="Book">
 <type content="empty">
 <attribute name="titleRef" type="string"/>
 <attribute name="categoryRef" type="string"/>
 </type>
 </element>
 </type>
</element>
</type>
</element>
<key name="bookKey">
 <selector>Library/BookCatalogue/Book</selector>
 <field>Title</field>
 <field>@Category</field>
</key>
<keyref name="bookRef" refer="bookKey">
 <selector>Library/CheckoutRegister/Book</selector>
 <field>@titleRef</field>
 <field>@categoryRef</field>
</keyref>
</schema>
```

# Дефиниране на ключ (Key)

```
<key name="bookKey">
 <selector>Library/BookCatalogue/Book</selector>
 <field>Title</field>
 <field>@Category</field>
</key>
```

“Комбинацията от съдържанието на елемента Title плюс стойността на атрибута Category ще бъде уникална за целия XML документ-екземпляр. Елементът Title и атрибутът Category са към элемента Book, зададен като селектор чрез XPath израз.”

Заб.: можем да задаваме едно и повече полета (field), като ключът е комбинацията от всички тях в зададения ред.

# Дефиниране на референция (Reference) към ключ (Key)

```
<keyref name="bookRef" refer="bookKey">
 <selector>Library/CheckoutRegister/Book</selector>
 <field>@titleRef</field>
 <field>@categoryRef</field>
<keyref>
```

“Два атрибути - *titleRef* и *categoryRef*, заедно са референция към ключа, дефиниран от *bookKey*. Те принадлежат към селектора, дефиниран чрез Xpath израз.”

Note: полетата на keyref трябва да съответстват на типа и на позицията на тези в ключа.

# Задаване на уникалност (Uniqueness)

```
<unique name="bookUnique">
 <selector>Library/BookCatalogue/Book</selector>
 <field>Title</field>
 <field>@Category</field>
</unique>
```

# Обхват на уникалността в XML Schema

- Елементите key/keyref/unique могат да се намират където и да е в схемата.
- Местоположението им определя обхвата на уникалността.
- В примера, key/keyref елементите се намират на top-level (в прекия наследник на элемента schema). Затова уникалността им е за целия документ.
- Ако обаче разположим елементите key/keyref като деца на элемента Book, то уникалността им ще е само за елементите Book в документа-екземпляр.

# any елемент

- Елементът **any** задава какъв да е well-formed XML – произволен брой елементи от което и да е пространство от имена

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
 <xsd:element name="root">
 <xsd:complexType>
 <xsd:sequence minOccurs="1" maxOccurs="1">
 <xsd:any namespace="#any" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

The free-form element can contain any Well-Formed XML (WFXML).

```
<root>
 <comment source="BB">This is great!</comment>
</root>
```

# anyAttribute

- Елементът anyAttribute задава какъв да е атрибут от кое и да е пространство от имена:

```
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
 <xsd:element name="root"xsd:complexTypexsd:anyAttribute namespace="#any"/>xsd:complexType>
 </xsd:element>
 </xsd:schema>
```

```
<root comment="This is great"/>
```

# (Почти) any елемент / атрибут

`<any namespace="#other"/>`

- allows any well-formed XML element, provided the element is in another namespace than the one we're defining.

`<any namespace="http://www.somewhere.com"/>`

- allows any well-formed XML element, provided it's from the specified namespace.

`<any namespace="#targetNamespace"/>`

- allows any well-formed XML element, provided it's from the namespace that we're defining.

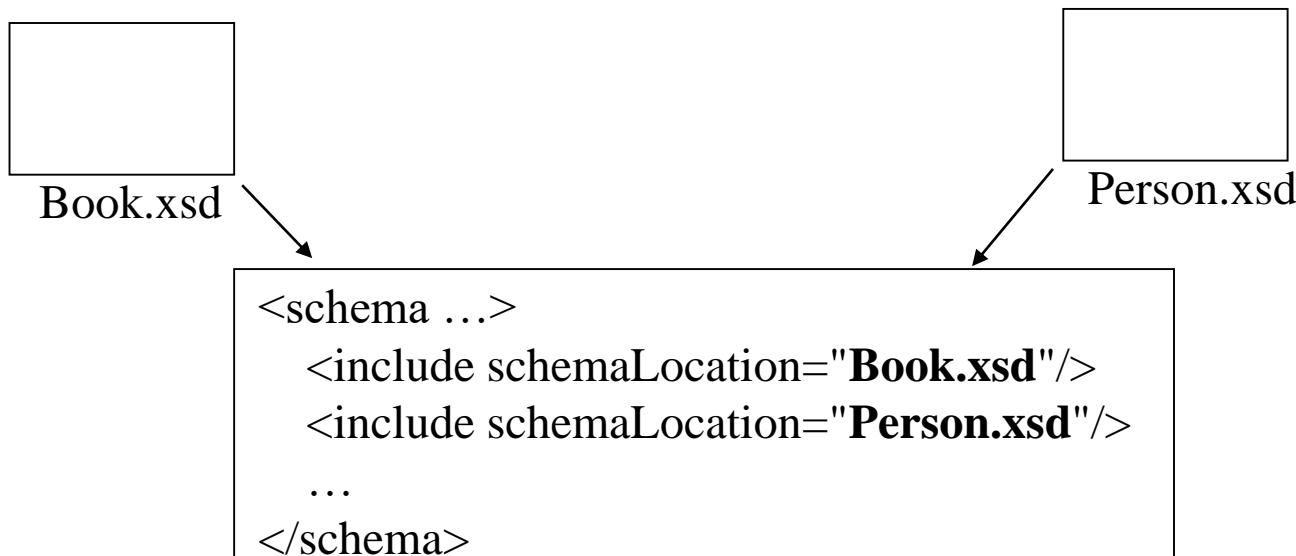
## Валидиране на документ с две схеми

```
<Library xmlns:book="http://www.somewhere.org/Examples"
 xmlns:person="http://www.somewhere-else.org/Person"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation=
 "http://www.somewhere.org/Examples
 http://www.somewhere.org/ Examples/Book.xsd
 http://www.somewhere-else.org/Person
 http://www.somewhere-else.org/Person/Person.xsd">
 <BookCatalogue>
 <book:Book>
 <book:Title>Illusions The Adventures of a Reluctant Messiah</book:Title>
 <book:Author>Richard Bach</book:Author>
 <book:Date>1977</book:Date>
 <book:ISBN>0-440-34319-4</book:ISBN>
 <book:Publisher>Dell Publishing Co.</book:Publisher>
 </book:Book>
 <book:Book>
 <book:Title>The First and Last Freedom</book:Title>
 <book:Author>J. Krishnamurti</book:Author>
 <book:Date>1954</book:Date>
 <book:ISBN>0-06-064831-7</book:ISBN>
 <book:Publisher>Harper & Row</book:Publisher>
 </book:Book>
 </BookCatalogue>
 <Employees>
 <person:Employee>
 <person:Name>John Doe</person:Name>
 <person:SSN>123-45-6789</person:SSN>
 </person:Employee>
 <person:Employee>
 <person:Name>Sally Smith</person:Name>
 <person:SSN>000-11-2345</person:SSN>
 </person:Employee>
 </Employees>
</Library>
```

Library.xml

# Съставяне на схема от няколко схеми

- Елементът ***include*** задава включване в дадена схема на дефиниции от няколко схеми
  - Те трябва да имат едно и също пространство от имена
  - Внасят дефинициите директно в съдържащата ги схема



```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:lib CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples"
 xmlns:lib="http://www.somewhere.org/Examples">
 <include schemaLocation="http://www.somewhere.org/Examples/Book.xsd"/>
 <include schemaLocation="http://www.somewhere.org/Examples/Person.xsd"/>
 <element name="Library">
 <type>
 <element name="BookCatalogue">
 <type>
 <element ref="lib:Book" minOccurs="0" maxOccurs="*"/>
 </type>
 </element>
 <element name="Employees">
 <type>
 <element name="lib:Employee" minOccurs="0" maxOccurs="*"/>
 </type>
 </element>
 </type>
 </element>
</schema>
```

# Елемент `import`

- Елементът ***import*** разрешава да се реферират елементи от друго пространство от имена

```
<schema xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples"
 xmlns:html="http://www.w3.org/1999/XHTML">
 <import namespace="http://www.w3.org/1999/XHTML"
 schemaLocation="http://www.w3.org/XHTML/xhtml.xsd"/>
 ...
 <element ref="html:table">
 ...
</schema>
```

# Декларация или дефиниция

- Използваме декларации (*declarations*) за това, което ще се съдържа в XML документа-екземпляр.
- Използваме дефиниции (*definitions*) за задаване на нови типове и групи – те се използват в схемата

Declarations:

- element declarations
- attribute declarations

Definitions:

- type definitions
- attribute group definitions
- model group definitions

# xsd:ENTITY 1/2

- XML Schema, както и другите schema езици не предоставят заместващ механизъм за деклариране на **DTD entities**
- Типът **xsd:ENTITY** представя референция към unparsed entity и се ползва за включване на документи, които не са в XML формат.
- Стойността на xsd:ENTITY трябва да бъде **NCName** (non-colonized name, или обратното на **QName**)
- Стойността на xsd:ENTITY трябва да бъде **unparsed entity** от DTD дефиницията за документа-екземпляр.

# xsd:ENTITY 2/2

## Schema:

```
<xs:element name="picture">
 <xs:complexType>
 <xs:attribute name="location"
 type="xs:ENTITY"/>
 </xs:complexType>
</xs:element>
<!--...-->
```

## Instance:

```
<!DOCTYPE catalog SYSTEM "catalog.dtd" [
 <!NOTATION jpeg SYSTEM "JPG">
 <!ENTITY prod557 SYSTEM "prod557.jpg"
 NDATA jpeg>
 <!ENTITY prod563 SYSTEM "prod563.jpg"
 NDATA jpeg>]>
<catalog>
 <product>
 <number>557</number>
 <picture location="prod557"/>
 </product>
 <product>
 <number>563</number>
 <picture location="prod563"/>
 </product>
</catalog>
```

# Управление на версии на схемата

- За документиране

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.somewhere.org/Examples"
 version="1.0">
 ...
</schema>
```

# <simple/complex-type> или <datatype>?

- Кога да използваме simple/complex-type и datatype?
  - Използваме <simple/complex-type> елементи за деклариране на типове на елементи и/или атрибути
  - Използваме datatype елемента за деклариране на произведен типове (на базата на string, integer, ...)
- XML Schema елементът с име simpleContent задава разширения или ограничения върху:
  - complex type, който е само текст
  - simple type
- XML елемент от тип simpleContent няма под-елементи

# null съдържание на елемент

- Empty спрямо null:
  - empty: елемент от тип content="empty" е празен - не може да има съдържание.
  - null: този елемент в документа екземпляр може да бъде недефиниран – тогава xsi:null атрибутът му ще е 'true'

XML Schema:

```
<element name="PersonName">
 <complexType><sequence>
 <element name="forename" type="NMTOKEN"/>
 <element name="middle" type="NMTOKEN" nullable="true"/>
 <element name="surname" type="NMTOKEN"/>
 </sequence></complexType>
</element>
```

XML instance document:

```
<PersonName>
 <forename>John</forename>
 <middle xsi:null="true"/>
 <surname>Doe</surname>
</PersonName>
```

еъ XML Schema

Съдържанието на middle може да е NMTOKEN или да не е дефинирано!

# Заключение

- DTD (Document Type Definition) и XML schema или XSD (XML Schema Definition) задават правила, съгласно които се определят имената на елементите и атрибутите, тяхната последователност, честота на срещане и др.
- DTD използва по-стегнат и кратък синтаксис в сравнение с XML Schema, но за сметка на това XML Schema предоставя по-богат набор от средства за по-строго дефиниране на структурата на XML и освен това нейните правила се задават в XML формат.

# Въведение в XSLT

(eXtensible Stylesheet Language for Transformations),  
XPath и XQuery

Преглед на XSL  
Употреба  
Основи на Xpath  
Синтаксис  
Локации и оси  
Xquery  
Примери



# Стилови множества (Style Sheet)

- **CSS - Cascading Style Sheet Specification**

- Предоставя прост синтаксис за добавяне на стилове към елементи (в HTML браузъри)

- **DSSSL - Document Style and Semantics Specification Language**

- Международен SGML стандарт за стилове и конвертиране на документи

- **XSL - Extensible Stylesheet Language**

- Комбинира черти на DSSSL и CSS, използвайки XML синтаксис

# XSL-FO и XSLT

XSL се състои от две части:

- **Extensible Stylesheet Language – Formatting Objects (XSL-FO)** – език за описание на форматирането на данните в XML документ с цел представянето им на различни медии (напр. экран, принтер или мултимедия);
- **Extensible Stylesheet Language for Transformation (XSLT)** – за трансформиране на XML документи с помощта на различни стилове и функции. Най-често се използва за конвертиране на документ от XML формат към документ в HTML формат, обикновен текстов файл или пък например друг XML документ. Полезен е, когато искаме да разделим презентационния слой на едно приложение от модела на данните му.

Входен XML  
документ  
(начално дърво)

XSLT  
декларации

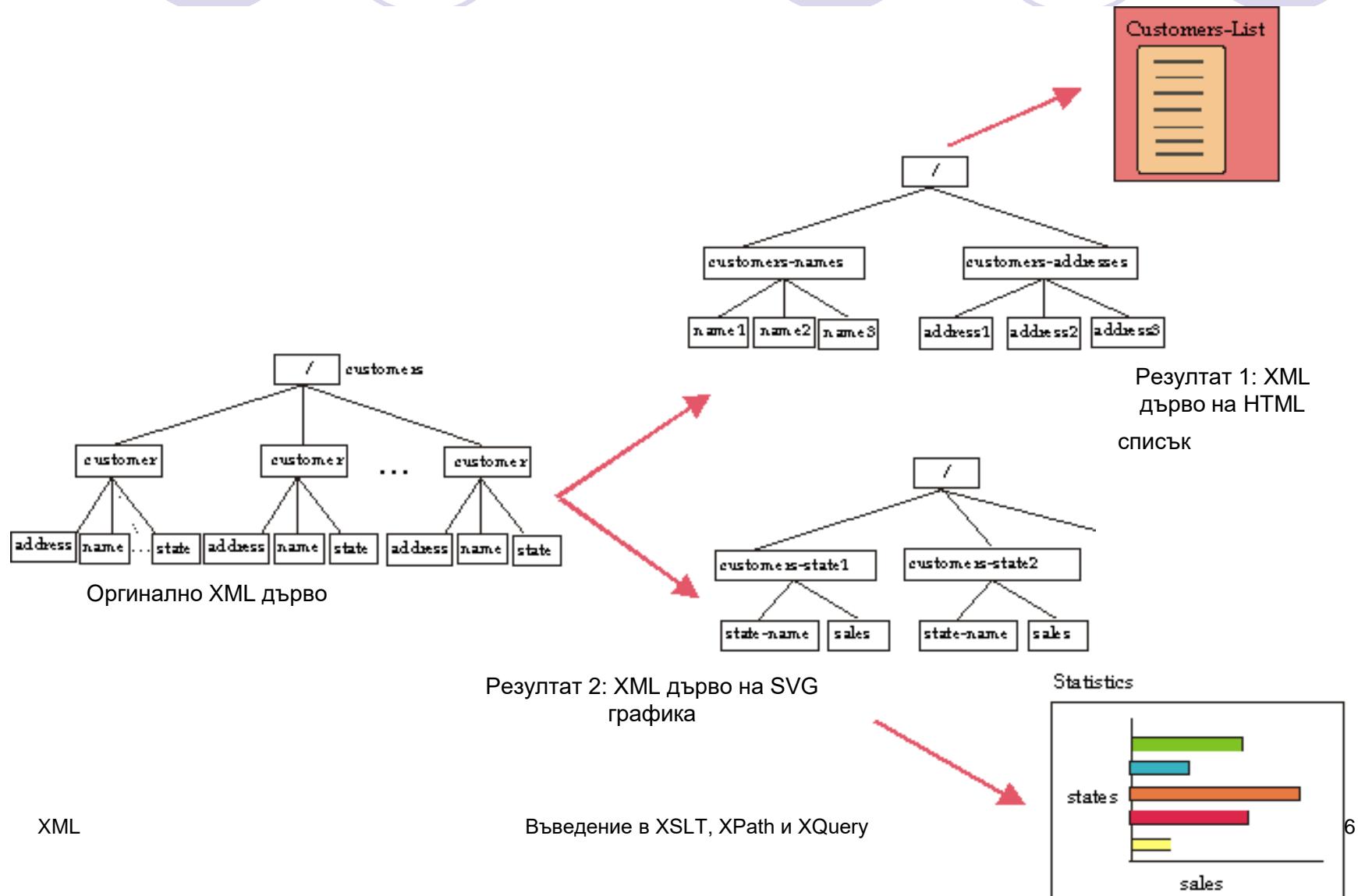
**XSLT процесор**

Изходен текстов  
документ  
(крайно дърво)

# Възможности на XSL/XSLT

- Добавяне на префиксен или суфиксен текст към съществуващо съдържание
- Структурни промени на входното съдържание, като създаване, отстраняване, редактиране, пренареждане и сортиране на XML елементи
- Многократно използване на елементно или атрибутно съдържание на друго място в документа
- Трансформиране на данни между XML формати
- Определяне на XSL форматиращи обекти и на други средства за представяне на съдържанието в дадена медиа (напр. CSS), с цел да се прилагат към даден елемент

# Използване на XSL/XSLT



# Прост пример

- **Файл data.xml:**

```
<?xml version="1.0"?>
<?xmlstylesheet type="text/xsl" href="render.xsl"?>
<message>Hmmm...</message>
```

- **Файл с дефиниция на трансформация render.xsl:**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <!-- one rule, to transform the input root (/) -->
 <xsl:template match="/">
 <html><body>
 <h1><xsl:value-of select="message"/></h1>
 </body></html>
 </xsl:template>
</xsl:stylesheet>
```

# Файл с разширение .xsl

- Всеки XSLT документ има .xsl разширение
- XSLT документ започва с:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/
 XSL/Transform">
```
- Може да съдържа шаблони, като напр.:

```
<xsl:template match="/" ... </xsl:template>
```
- Завършва с:

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

# Намиране на текста message

- Шаблонът `<xsl:template match="/">` задава селектиране на целия входен документ, т.е. на *root* възела на XML дървото
- Вместо това,
  - `<xsl:value-of select="message"/>` селектира преките наследници на *message*
  - Това са Xpath изрази, както и аналогичните им:
    - `./message`
    - `/message/text()` (`text()` е **XPath функция**)
    - `./message/text()`

# Как работи?

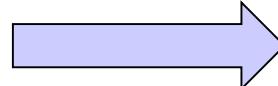
- XSL дефиницията `render.xsl` е:  
`<xsl:template match="/">`  
    `<html><body>`  
    `<h1><xsl:value-of select="message"/></h1>`  
    `</body></html>`  
`</xsl:template>`

- Шаблонът `<xsl:template match="/">` избира корена
- `<html><body>` `<h1>` се записва в изходния файл
- Съдържанието на `message` се записва в изходния файл
- `</h1>` `</body></html>` се записва в изходния файл

Вход:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href='render.xsl'?>
<message>Hmmm...</message>
```

XML



Резултат:

```
<html><body>
 <h1>Hmmm...</h1>
</body></html>
```

# XSLT спецификация

- Налична на <http://www.w3.org/XSL/>
  - Дефинира 34 елемента и техните атрибути
  - Изисква:
    - **Namespace**
    - **XPath**

# XPath и XQuery

Съществуват два популярни декларативни езици + технологии, предназначени за адресиране (локализиране) на определени части и структури от XML документ:

- XPath се използва за адресиране и манипулиране на секции от XML документ:
  - много популярен стандарт от 1999 година насам
  - използва се от останалите XML спецификации XPointer, XQL, XSLT. Локацията се задава чрез URL
  - работи с интернет, интранет и файловата система
- XQuery (от 2007год.) е друг език за описание на заявки към XML документ, но подобно на SQL заявките към релационна база от данни.

# XPath модел

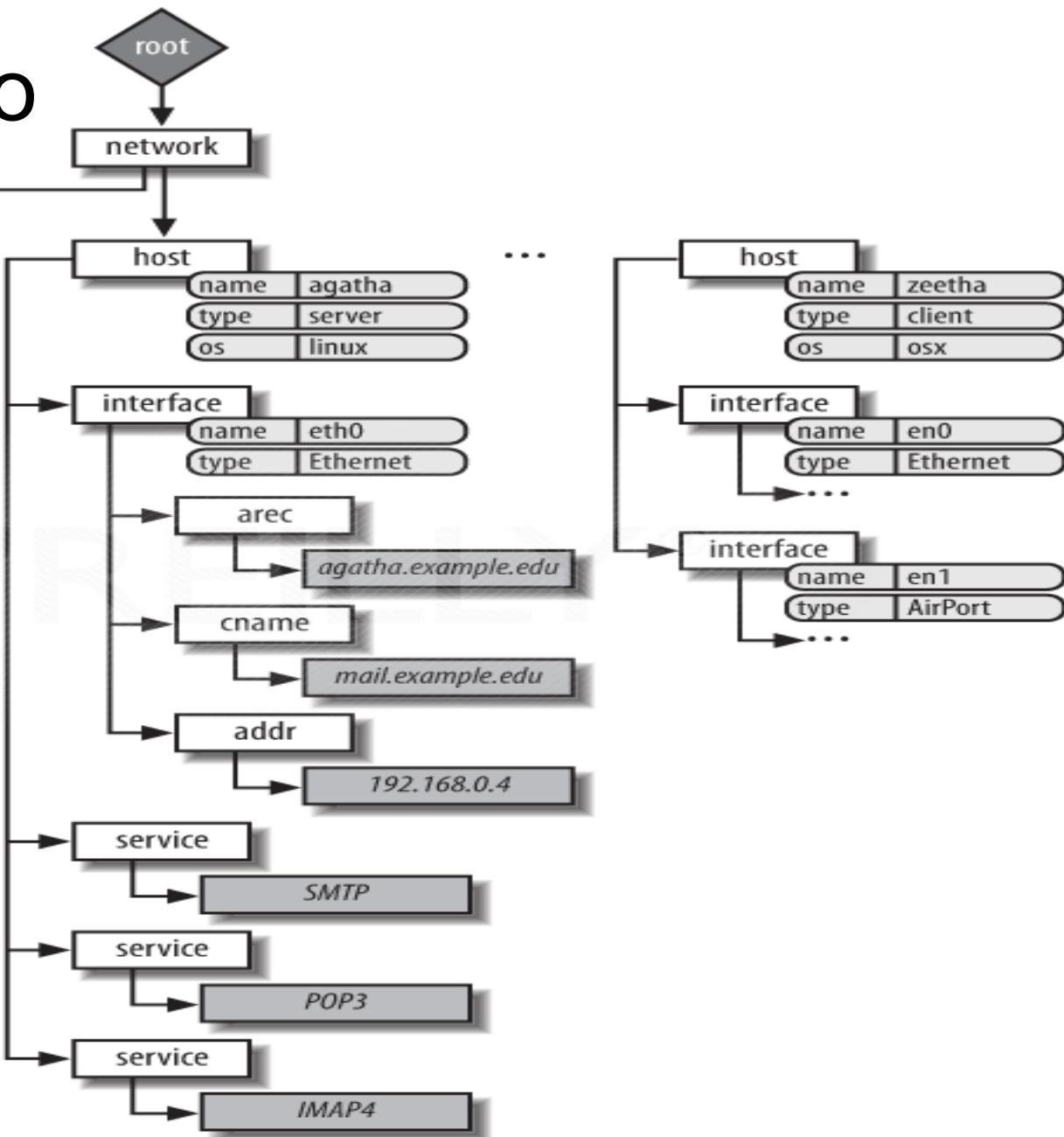
- В XPath 1.0 модела, повечето съставни части от XML документа са представени като възли, свързани с определени отношения.
- Коренът на XPath 1.0 дървото представлява самият документ, а не коренът на документа.
- Всеки елемент в XML документа се представя от елементен възел в дървото.
- Всеки атрибут се представлява от атрибутен възел и по аналогичен начин се представят коментари и инструкции за обработка.
- Текстовият възел представя текстово съдържание на елемент.
- Използваните в документа пространства от имена са представени от възли от тип пространство

# Изходен документ

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
 <description name="Boston">
 This is the configuration of our network in the
 Boston office.
 </description>
 <host name="agatha" type="server" os="linux">
 <interface name="eth0" type="Ethernet">
 <arec>agatha.example.edu</arec>
 <cname>mail.example.edu</cname>
 <addr>192.168.0.4</addr>
 </interface>
 <service>SMTP</service>
 <service>POP3</service>
 <service>IMAP4</service>
 </host>
 <host name="gil" type="server" os="linux">
 <interface name="eth0" type="Ethernet">
 <arec>gil.example.edu</arec>
 <cname>www.example.edu</cname>
 <addr>192.168.0.5</addr>
 </interface>
 <service>HTTP</service>
 <service>HTTPS</service>
 </host>
 <host name="baron" type="server" os="linux">
 <interface name="eth0" type="Ethernet">
 <arec>baron.example.edu</arec>
 <cname>dns.example.edu</cname>
 <cname>ntp.example.edu</cname>
 <cname>ldap.example.edu</cname>
 <addr>192.168.0.6</addr>
 </interface>
 <service>DNS</service>
 <service>NTP</service>
 <service>LDAP</service>
 <service>LDAPS</service>
 </host>
```

```
<host name="mr-tock" type="server" os="openbsd">
 <interface name="fxp0" type="Ethernet">
 <arec>mr-tock.example.edu</arec>
 <cname>fw.example.edu</cname>
 <addr>192.168.0.1</addr>
 </interface>
 <service>firewall</service>
</host>
<host name="krosp" type="client" os="osx">
 <interface name="en0" type="Ethernet">
 <arec>krosp.example.edu</arec>
 <addr>192.168.0.100</addr>
 </interface>
 <interface name="en1" type="AirPort">
 <arec>krosp.wireless.example.edu</arec>
 <addr>192.168.100.100</addr>
 </interface>
</host>
<host name="zeetha" type="client" os="osx">
 <interface name="en0" type="Ethernet">
 <arec>zeetha.example.edu</arec>
 <addr>192.168.0.101</addr>
 </interface>
 <interface name="en1" type="AirPort">
 <arec>zeetha.wireless.example.edu</arec>
 <addr>192.168.100.101</addr>
 </interface>
</host>
</network>
```

# XPath дърво



# Контекстен възел

- XPath процесорът изчислява XPath израз, който задава път от дадена начална точка в дървото, например от корена.
- Чрез изчислението на този път като последователност от свързани възли, процесорът се „придвижва“ по дървото, моделиращо документа, до указания от израза възел.
- Този възел, където за момента се намира XPath процесорът, се нарича **контекстен възел**.
- От него започват всички относителни пътища от този момент до следващото позициониране на процесора в друг контекстен възел.
- Контекстният възел се задава с ‘.’
- **XPath връща множество от възли от дървото, а не XML документ (!)**

# Видове XPath възли

- възел-корен на дървото, сочещ към корена на документа
- възел-елемент - възелът, представляващ корена на документа, е задължителен (възелът с име `network`); други възли от този тип са `host`, `interface` и `service`;

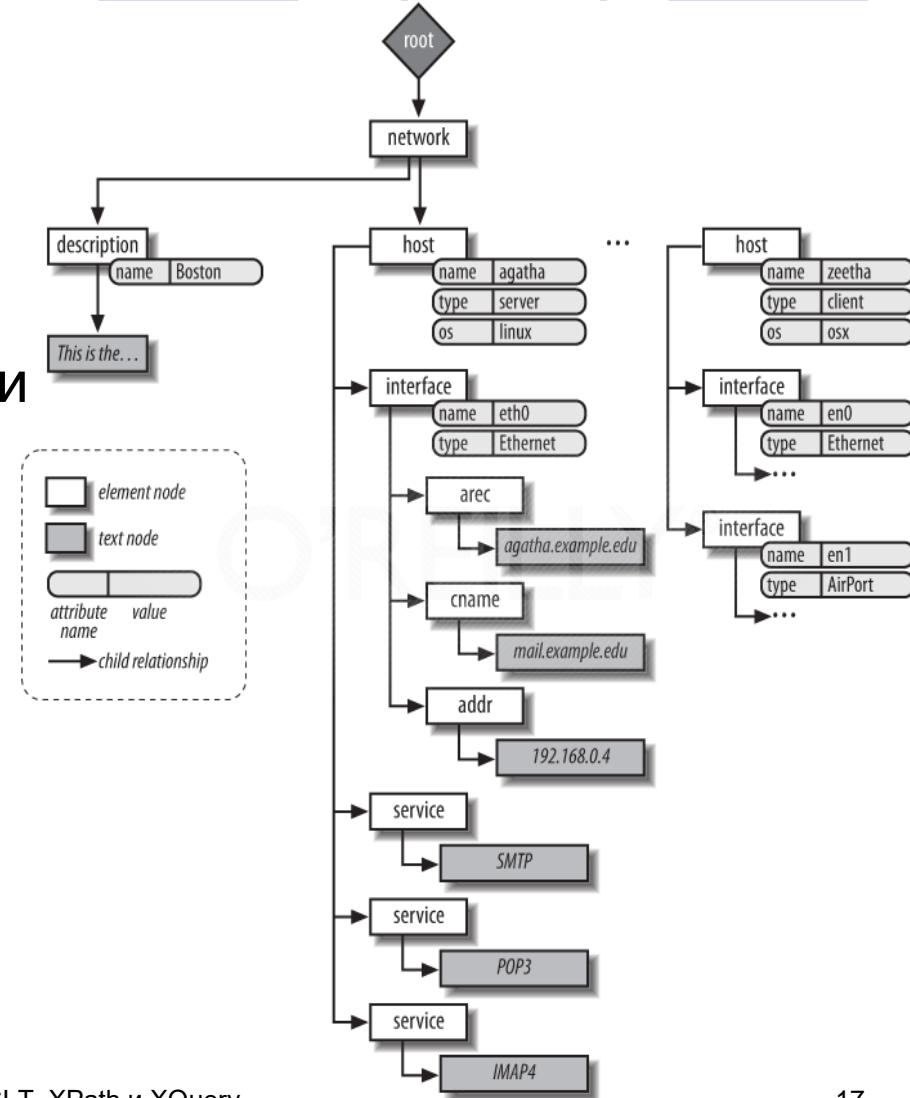
възел-текст - напр. възелът със съдържание "This is the...";

възел-атрибут - като напр. `name`, `type` и `os`;

възел-инструкция за обработка;

възел-декларация на CDATA секция (област от данни);

XML  
възел-коментар.



# Отношения

- Възлите в XPath дървото са свързани с отношения от два типа:
  - отношение на свързване на възел с подреден списък от възли-наследници**
  - отношение на свързване на възел с неподредено множество от други върхове**
- В подреден списък се представят наследниците на даден елемент (понеже те образуват подредена йерархия на елементното съдържание), текстовите възли, инструкциите за обработка и коментарите
- Не са подредени възлите за атрибути и CDATA секциите (областите от данни). Наистина, търсенето на n-ти елемент от йерархията от наследници за даден елемент има смисъл, докато търсенето на n-ти атрибут на елемент не може да дефинира върху неподредени атрибути. Също така, не се използва търсене на част от CDATA секция.

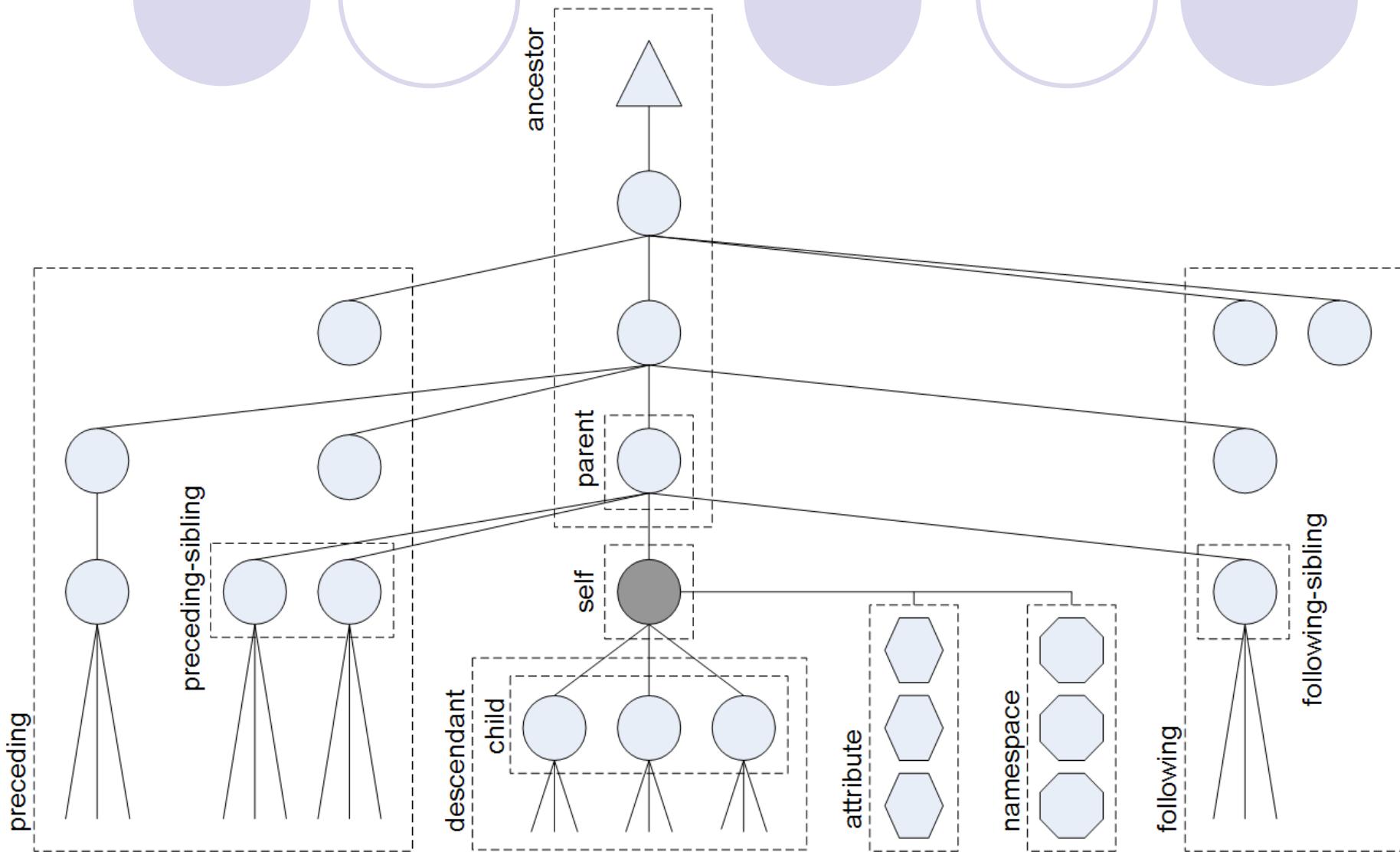
# XPath 1.0 оси 1/3

1. Ос на предшествениците (ancestor axis) - избира предшествениците на контекстния възел в обратен ред на появата им в документа
2. Ос на предшествениците и на самия възел (ancestor-or-self axis) - избира контекстния възел и неговите предшественици в обратен ред на появата им
3. Ос на директните наследници, или ос на децата (child axis) – селектира преките наследници на контекстния възел
4. Ос на атрибутите (attribute axis) – задава всички атрибути на контекстния възел
5. Ос на наследниците (потомците) (descendant axis) – връща всички наследници (преки и непреки) на контекстния възел по ред на появата им в документа. Атрибутите и секциите за данни не се разглеждат като наследници на елементен възел
6. Ос на наследниците и на самия възел (descendant-or-self axis) – връща резултата на оста на наследниците плюс контекстния възел

# XPath 1.0 оси 2/3

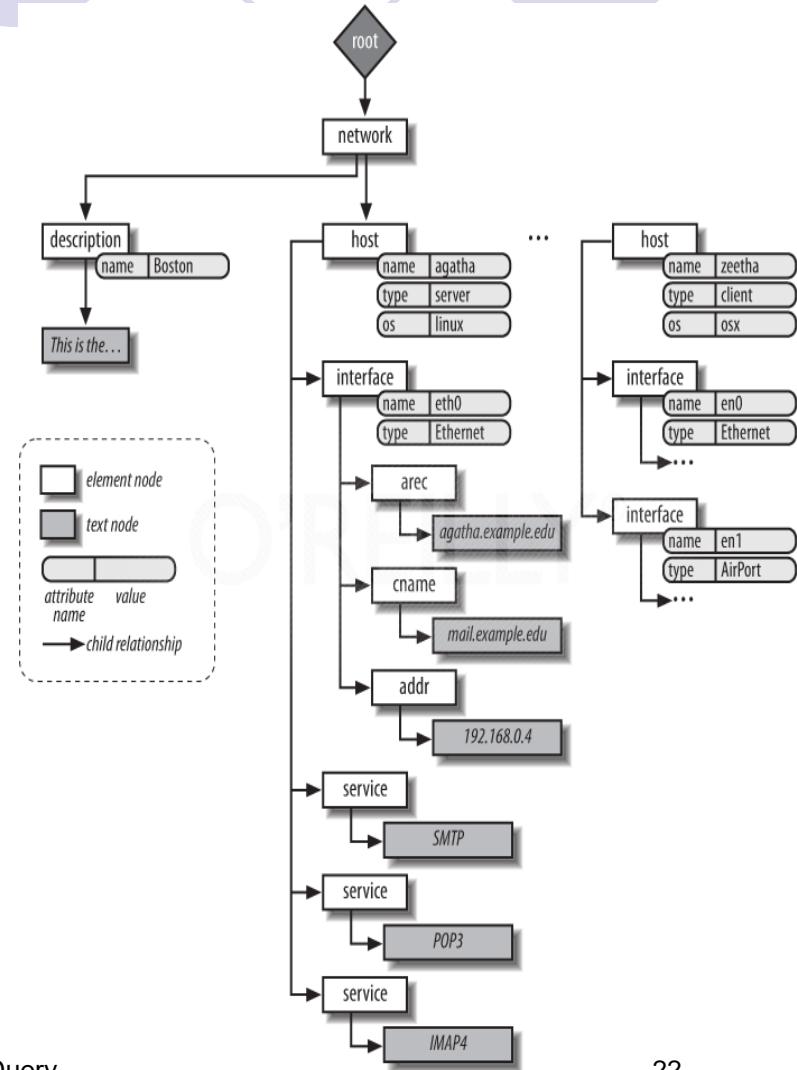
7. Ос на следващите възли (following axis) – селектира всички следващи възли след контекстния възел, без наследниците и атрибутите му, и без пространствата от имена
8. Ос на следващите възли и на самия възел (following-sibling axis) - избира всички братя и сестри на контекстния възел следващи вдясно от него
9. Ос на пространствата от имена (namespace axis) - (не се препоръчва в XPath 2.0)
10. Ос на родителя (parent axis) – връща родителя на контекстния възел
11. Ос на предходните възли (preceding axis) – селектира всички предходни възли спрямо контекстния възел, без наследниците и атрибутите му, и без пространствата от имена
12. Ос на предходните възли и на самия възел (preceding-sibling axis) – избира всички предшестващи братя и сестри на контекстния възел вляво от него
13. Ос на самия възел или собствена ос (self axis)

# XPath 1.0 оси 3/3



# Относителни и абсолютни пътища

- **Относителният път** на местоположение се състои от една или повече стъпки на местоположение (*location step*), разделени със знака '/'.  
This is the...
- **Абсолютният път** на местоположение се състои от знака '/', последван от относителен път.  
**/network/host** е абсолютен път на местоположение и задава възела на първия елемент с име **host** за корена на документа.
- Относителните пътища могат да съдържат в началото си низа '.' - напр. **./interface/cname**



# Разширен синтаксис 1/2

**axis :: node-test [ predicate ] ... [predicate]**

- Ос - индикатор за връзка между текущия възел, избран на предходната стъпка, и избраните от стъпката възли.
- Филтър - тест за възела (*node test*) - задава характеристики на възлите, които ще бъдат избрани при удовлетворяване на условията:
  1. Задаването на името на възел като филтър тества за възли с това име, тоест избира всички възли с това име. Така например **child::elementName** връща всички наследници (деца) на контекстния възел с име **elementName**
  2. Във филтрите могат да се ползват специфични функции за тестване на типа на възела на избраната ос. Ако функцията върне стойност истина, то възелът ще бъде избран:
    1. Тестът **text()** връща истина за всеки текстов възел, затова **child::text()** ще избере само текстовите възли - деца на контекстния възел.
    2. **comment()** връща истина за всеки възел-коментар, така че **child::comment()** ще избере само тези деца на контекстния възел, които са от тип коментар.
    3. **processing-instruction()** връща истина за всеки текстов възел - инструкция за обработка.
    4. Тестът **node()** връща истина за всеки възел от всяка към тип, затова **self::node()** избира контекстния възел, тоест ''.

# Разширен синтаксис 2/2

**axis :: node-test [ predicate ] ... [predicate]**

- Предикатите, ако такива съществуват, филтрират допълнително набора от възли (върнат от филтъра) по отношение на дадена ос спрямо контекстния възел.
- Всеки предикат се изчислява за всеки възел от набора от възли и ако върне стойност истина, този възел се добавя към резултатния набор от възли. Така например:
  - **descendant::document[attribute::level = "confidential"]**  
- задава елементите-наследниците (те могат да бъдат само елементи, защото само елементи изграждат йерархията на дървото) на контекстния възел с име document и с атрибут с име level и стойност "confidential".
  - **preceding::\*** - адресира всички предшественици на контекстния възел чрез използване на заместващия символ "\*", който задава елемент с какво и да е име.

# Примери с разширен синтаксис 1/2

- Допълнително отношение в предиката може да има позицията на близостта (*proximity position*) спрямо дадена ос
- Позицията на близостта на член на набора възли по отношение на дадена ос се определя от номера на позицията на възела в подредените в документа възли
- Например **pElement[position()=3]** задава третият по ред възел с име **pElement**.
- Преминаването през елементите на йерархията от предния слайд:
  - **child::network/child::host/child::interface**

# Примери с разширен синтаксис 2/2

- Друг пример: **child::chapter/descendant-or-self::node()/child::section**
- - при изчислението му процесорът ще премине през цялата йерархията на елемента **chapter** и избере всички елементи с име **section**
- Изразът **descendant-or-self::node()** работи рекурсивно по йерархията на наследниците и е мощно средство за претърсване на дървета
- **child::para[attribute::type='warning'][position()=2]**
- - селектира втория пряк наследник **para** на контекстния възел, който има атрибут с име **type** и стойност **warning**

# Кратък синтаксис

- ❖ Очевидно разширеният синтаксис не е удобен за работа и затова XPath използва и кратък синтаксис
- ❖ Краткият запис на пътищата наподобява добре познатото адресиране по директориите на файловите системи и използва същия синтаксис.
- ❖ Краткият запис използва съкращения като ‘@’ (значение за атрибут) и ‘\*’ (избор на всички възли).

# Предшественици

- Избор на родителя на контекстния възел:
  - '...' или разширена форма
  - `parent()`
- Избор на всички 'title' елементи – деца на родителя на контекстния възел:
  - `../title` или разширена форма
  - `parent::node() / child::title`

# Абсолютни пътища

- Чрез '/' означаваме стартирането от корена (root element)
  - За да намерим всички 'para' елементи в даден документ, ползваме:
    - //**para** или разширена форма
    - /descendant-or-self::node() /para**

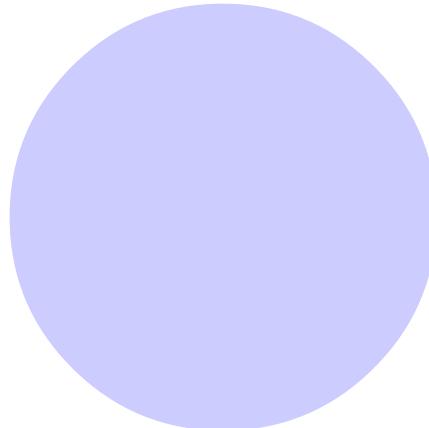
# Оператор на рекурсивния спуск

- Рекурсивно търсене сред наследените възли
  - **chapter//para** се спуска в йерархията на chapter и избира всички **para** елементи
    - <chapter> //Starting node<title>...</title><para>...</para> //Selected<note><para>...</para> //Selected<note></chapter>
  - Разширено: **child::chapter/descendant-or-self::node() / child::para**
- Избиране на всички **para** елементи от текущия възел: **..//para**
  - Разширено:  
**self::node() / descendant-or-self::node() / child::para**

# Примери

№	Разширен синтаксис	Кратък синтаксис
1	<b>self::node()/descendant-or-self::node()/child::p</b>	<b>//p</b>
2	<b>parent::node()/child::title</b>	<b>../title</b>
3	<b>child::host[service]</b>	<b>host[service]</b>
4	<b>/child::chapter/descendant-or-self::node()/child::section</b>	<b>/chapter//section</b>
5	<b>attribute::*</b>	<b>@*</b>
6	<b>child::*</b>	<b>*</b>
7	<b>self::node()/child::section[position()=last()]</b>	<b>./section[last()]</b>
8	<b>/child::doc/child::chapter[position()=5]/child::section[position()=2]</b>	<b>/doc/chapter[5]/section[2]</b>
9	<b>parent::node()/attribute::lang</b>	<b>../@lang</b>
10	<b>//descendant-or-self::node()/child::host[attribute::type='server']</b>	<b>//host[@type='server']</b>

Какво означава всеки от изразите от  
предния слайд?



# Отговори 1/2

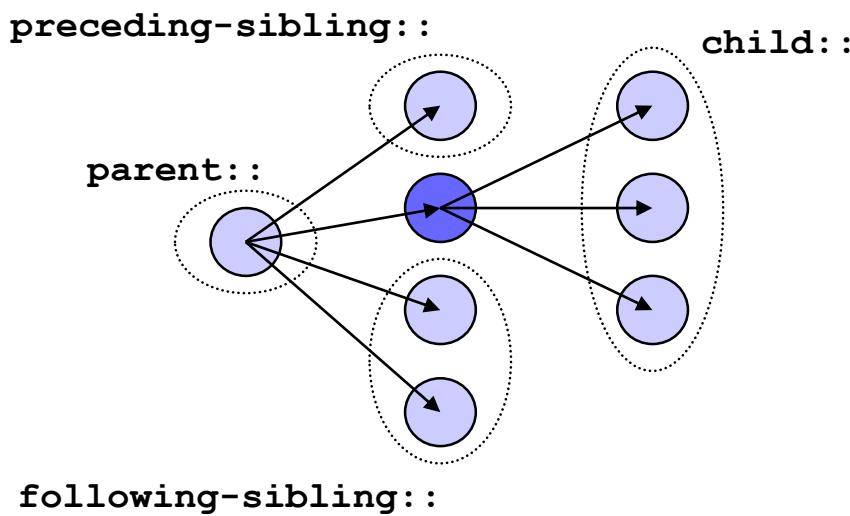
1. **self::node()/descendant-or-self::node()/child::p** или **./p** - задава всички възли с име **p** в йерархията на контекстния възел, т.е. всички негови преки или непреки наследници с това име;
2. **parent::node()/child::title** или **../title** - селектира всички възли с име **title**, които са преки наследници (деца) на родителя на контекстния възел;
3. **child::host[service]** или **host[service]** - избира децата с име **host** на контекстния възел, които имат поне едно дете с име **service**
4. **/child::chapter/descendant-or-self::node()/child::section** или **/chapter//section** - задава всички възли **section** в йерархията на контекстния възел, които имат за предшественик **chapter**
5. **attribute::\*** или **@\*** - избира всички атрибути на контекстния възел

# Отговори 2/2

6. **child::\*** или **\*** - избира всички деца на контекстния възел
7. **self::node()/child::section[position()=last()]** или **./section[last()]** - селектира последното дете на контекстния възел с име **section**
8. **/child::doc/child::chapter[position()=5]/child::section[position()=2]** или **/doc/chapter[5]/section[2]** - избира втората секция на петата глава на документа
9. **parent::node()/attribute::lang** или **../@lang** - избира атрибута **lang** на родителя на контекстния възел
10. **//descendant-or-self::node()/child::host[attribute::type="server"]** или **//host[@type="server"]** - избира всички потомци на корена на документа от фиг. от слайд 16, които имат име **host** и атрибут **type** със стойност **server**

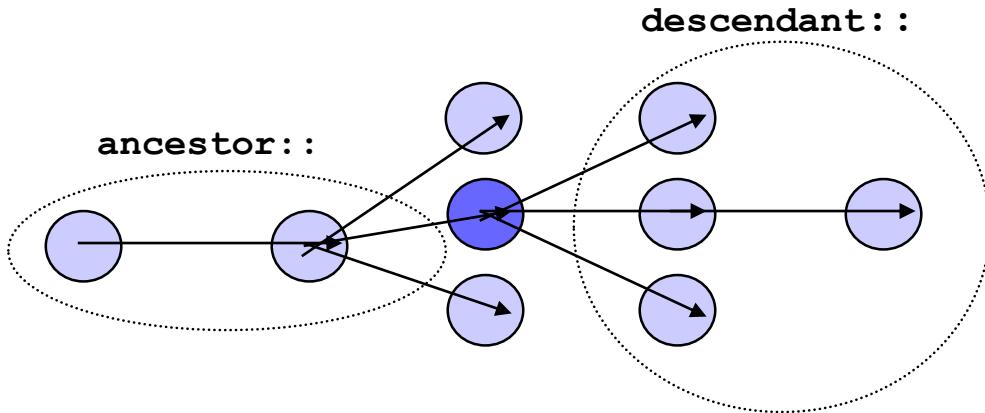
# Други релации 1/4

- Избор на братя и сестри (*siblings*) на текущия (контекстния) елемент:
  - preceding-sibling::**
  - following-sibling::**



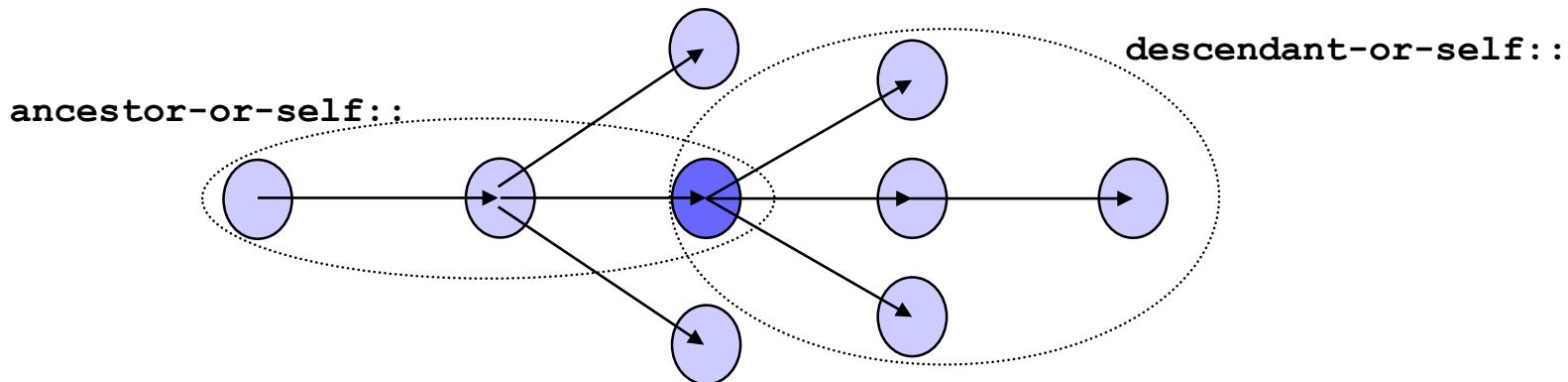
## Други релации 2/4

- Избор на всички предшественици (ancestors) и на потомци (descendants) на текущия (контекстния) елемент:
  - ancestor::**
  - descendant::**
- (без братя и сестри – siblings)



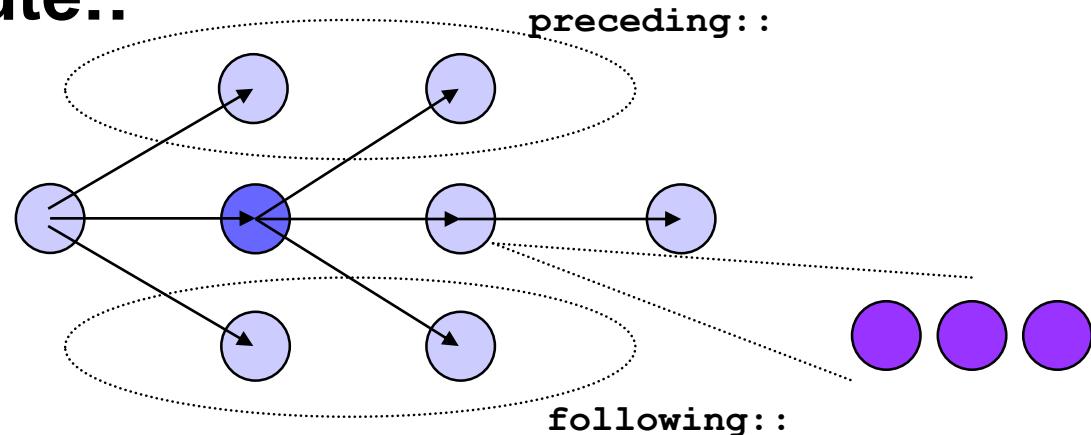
# Други релации 3/4

- Избор на всички предшественици (ancestors) и на потомци (descendants) на текущия (контекстния) елемент, вкл. и самия него:
  - ancestor-or-self::**
  - descendant-or-self::**



# Други релации 4/4

- Избор на всички предишни и следващи възли на текущия (контекстния) елемент:
  - preceding::**
  - following::**
- Избор на атрибутите му:
  - attribute::**



# Тестове

- Функция **position()**
  - `<xsl:template match="first/second[position() = 2]">` е както
  - `<xsl:template match="first/second[2]">`
- Функции **first() / last()**
  - Избор на първи/последен *sibling* в списък
- Функция **count()**
  - Изчислява броя на елементите в списък
  - **child::transcript[count(child::introduction) = 1]**
- Функция **id()**
  - Проверява идентификатора на елемента
  - **child::transcript[id("ENS0001")]**

# За повече информация – вижте спецификациите

<b>Part</b>	<b>Date</b>	<b>Status</b>	<b>URL</b>
<b>XSL-Formatting Objects ver. 2.0</b>	<b>04.02.2010</b>	<b>Version 2.0 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xslfo20/">http://www.w3.org/TR/xslfo20/</a></b>
<b>XSL Transformations (XSLT) ver. 2.1</b>	<b>11.05.2010</b>	<b>Version 2.1 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xslt-21/">http://www.w3.org/TR/xslt-21/</a></b>
<b>XML Path Language (XPath) ver. 2.0</b>	<b>23.01.2007</b>	<b>Version 2.0 W3C Working Draft</b>	<b><a href="http://www.w3.org/TR/xpath20/">http://www.w3.org/TR/xpath20/</a></b>

# XPath 2.0

- XML Path Language (XPath) 2.0 е препоръка (Recommendation) на W3C от 14.12.2010 и е налична на адрес <http://www.w3.org/TR/xpath20/>. Наборът от функции на тази спецификация е много по-богат, помощен и по-чувствителен към типа на данните.
- Също така новост в XPath 2.0 са поредиците или последователностите (*sequences*), които заменят познатите ни от XPath 1.0 множества (набори) от възли. Всички XPath 2.0 изрази се изчисляват върху такива поредици, като в тях може да използват променливи.
- XPath 2.0 също използва пътища за местоположение и използването на дефинираните от версия 1.0 оси, с изключение на оста за пространство от имената. В XPath 2.0 тази ос се счита за остатяла и неактуална, но е включена за обратна съвместимост с **XQuery**.

# XPath и XML Query (XQuery)

За адресиране (локализиране) на определени части и структури от XML документ:

- XPath се използва за адресиране и манипулиране на секции от XML документ:
  - много популярен стандарт от 1999 година насам
  - използва се от останалите XML спецификации XPointer, XQL, XSLT и XQuery
- XQuery (от 2007год.) е друг език за описание на заявки към XML документ, но подобно на SQL заявките към релационна база от данни.

*Следващите слайдове са базирани на презентация на Zaniolo, H. Yang, L.-J. Chen и F. Farfán*

# Цели на XQuery

- Увеличаване на количеството информация, съхранявана, обменена и представена като XML
- Възможност за интелигентно търсене на XML-базирани източници на данни
- Ефективно ползване на силата на XML: гъвкаво представяне на много видове информация от различни източници
- XML езикът за заявки трябва да извлича и интерпретира информация от тези различни източници
- Резултат: XQuery (2007г.)

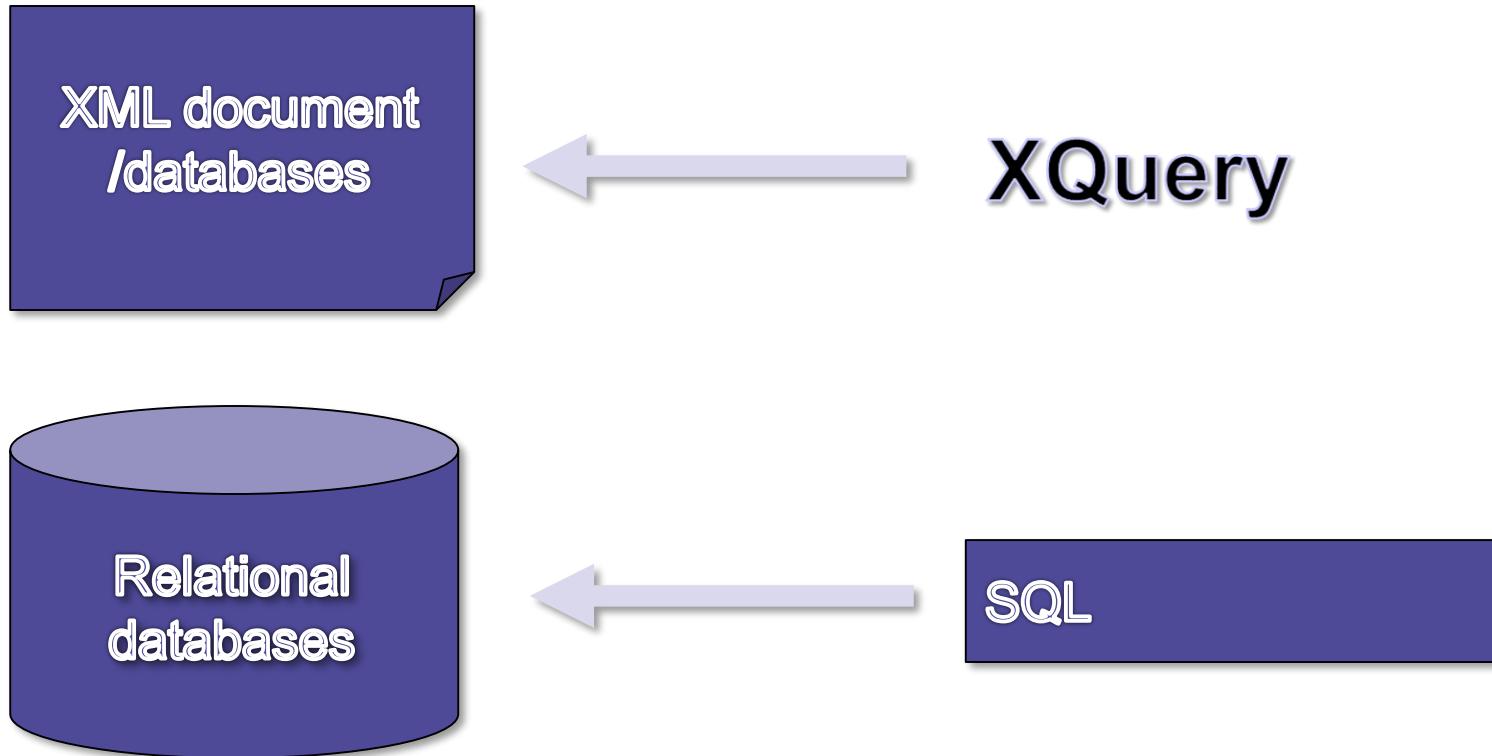
# Изисквания към XML Query

- Експресивна мощност
- Семантика
- Композируемост
- Използване на XML схема с цел валидация
- Манипулиране на програмата

# Езици за заявки към XML документи

- XPath: синтаксис на израз на път в XML документ, подходящ за йерархични документи
- XML-QL: свързващи променливи (binding variables ) и използване на променливи за създаване на нови структури
- SQL: SELECT-FROM-WHERE модел за преструктуриране на данните
- Quilt: приема много предимства от по-горните езици и е непосредственият предшественик на XQuery

# What is XQuery



# XQuery – какво?

- Създаден според изискванията на W3C XML Query Working Group
  - “XML Query 1.0 Requirements”
  - “XML Query Use Cases”.
- Проектиран да бъде малък и лесно-приложим език
- Достатъчно гъвкав, за да дефинира заявки към широк спектър от XML източници (както бази данни, така и документи).
- Използва синтаксис, който лесно може да се чете от човек

# XQuery – как?

- Ориентиран към изрази - основен градивен елемент
- Функционален език (поне според спецификациите)
- Силно-типизиран (Strongly-typed) език.

# XQuery спрямо XSLT

Преимущества или недостатъци?

- XSLT - document-driven; XQuery - program driven
- XSLT е написан на XML; XQuery – не

Твърдение (да се провери практически 😊):

- с XSLT 2.0 може да се направи всичко това, което може да стане с XQuery

# XQuery - концепции

- Заявката в XQuery е израз, който:
  - Чете различни XML документи или фрагменти
  - Връща последователност от добре оформени (well-formed) XML фрагменти

# Основни форми на XQuery изрази

1/5

- Примитивни (Primary)
  - Литерали, променливи, функционални извиквания и скоби (за задаване на приоритети)
- Път (Path)
  - Открива възли в дърво и връща последователност от отделни възли в реда, зададен в документа
- Последователност (Sequence)
  - Подредена колекция от нула или повече елементи, в която елементът може да бъде атомарна стойност или възел
  - Елементът е идентичен на последователност с дължина единица, съдържаща този елемент
  - Последователностите никога не са вложени

# Основни форми на XQuery изрази 2/5

- Аритметични
  - Аритметични оператори за добавяне, изваждане, умножение, разделяне и деление по модул
- Условни
  - Четири вида сравнения: на стойности, общи, на възли и сравнения на заявките
- Логически
  - Логическият израз е AND-израз или OR-израз.
  - Стойността на логически израз винаги е стойност от тип Boolean

# Основни форми на XQuery изрази 3/5

- Конструктор (Constructor)
  - Конструкторите могат да създават XML структури в рамките на заявка
  - Има конструктори за елементи, атрибути, раздели CDATA, инструкции за обработка и коментари
- FLWR (произнася се както "flower")
  - Изразяване за итерация и за свързване на променливи до междинни резултати
  - Полезни са за изчисляване на свързването между два или повече документа и за преструктуриране на данните.
  - FLWR означава ключовите думи **FOR, LET, WHERE** и **RETURN** – четири възможни клаузи

# Основни форми на XQuery изрази 4/5

- Сортиращи изрази
  - Предоставят начин да се контролира реда на елементите в последователности
- Условни изрази
  - Въз основа на ключовите думи IF, THEN и ELSE.
- Количествени изрази (Quantified expressions)
  - Поддържат количествено определяне
  - Стойността на един количествен израз винаги е верна или неверна

# Основни форми на XQuery изрази 5/5

- Типове данни
  - Проверка и манипулиране на типа по време на изпълнение
- Validate
  - Израз от тип validate валидира своя аргумент по отношение на дефинициите в обхвата от схемата, като използва процеса на валидация, описан в XML Schema

# Примерен XML документ: bib.xml

```
<bib>
 <book year="1994">
 <title>TCP/IP Illustrated</title>
 <author><last>Stevens</last><first>W.</first></author>
 <publisher>Addison-Wesley</publisher>
 <price> 65.95</price>
 </book>
 <book year="1992">
 <title>Advanced Programming in the Unix environment</title>
 <author><last>Stevens</last><first>W.</first></author>
 <publisher>Addison-Wesley</publisher>
 <price>65.95</price>
 </book>
</bib>
```

# XQuery пример 1

- Find all books with a price of \$39.95

## XQuery:

```
document("bib.xml")/bib/book[price = 39.95]
```

## Result:

```
<book year="2000">
 <title>Data on the Web</title>
 <author><last>Abiteboul</last><first>Serge</first></author>
 <author><last>Buneman</last><first>Peter</first></author>
 <author><last>Suciu</last><first>Dan</first></author>
 <publisher>Morgan Kaufmann Publishers</publisher>
 <price> 39.95</price>
</book>
```

Източник: Craig Knoblock. Xquery Tutorial

# XQuery пример 2

- Find the title of all books published before 1995

**XQuery:**

```
document("bib.xml")/bib/book[@year < 1995]/title
```

**Result:**

```
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix environment</title>
```

# XQuery пример 3 (For Loop)

- List books published by Addison-Wesley after 1991, including their year and title.

**XQuery:**

```
<bib>
{
 for $b in document("bib.xml")/bib/book
 where $b/publisher = "Addison-Wesley" and $b/@year >
1991
 return
 <book year="{ $b/@year }">
 { $b/title }
 </book>
}
</bib>
```

# XQuery пример 3 (For Loop)

- List books published by Addison-Wesley after 1991, including their year and title...

## Result:

```
<bib>
 <book year="1994">
 <title>TCP/IP Illustrated</title>
 </book>
 <book year="1992">
 <title>Advanced Programming in the Unix environment</title>
 </book>
</bib>
```

# XQuery пример 4 (Join)

- For each book found at both bn.com and amazon.com, list the title of the book and its price from each source.

## XQuery:

```
<books-with-prices>
{
 for $b in document("bib.xml")//book,
 $a in document("reviews.xml")//entry
 where $b/title = $a/title
 return
 <book-with-prices>
 { $b/title }
 <price-amazon>{ $a/price }</price-amazon>
 <price-bn>{ $b/price }</price-bn>
 </book-with-prices>
}
</books-with-prices>
```

# XQuery пример 4 (Join)

- For each book found at both bn.com and amazon.com, list the title of the book and its price from each source.

## Result:

```
<books-with-prices>
 <book-with-prices>
 <title>TCP/IP Illustrated</title>
 <price-amazon><price>65.95</price></price-amazon>
 <price-bn><price> 65.95</price></price-bn>
 </book-with-prices><book-with-prices>
 <title>Advanced Programming in the Unix
environment</title>
 <price-amazon><price>65.95</price></price-amazon>
 <price-bn><price>65.95</price></price-bn>
 </book-with-prices><book-with-prices>
 <title>Data on the Web</title>
 <price-amazon><price>34.95</price></price-amazon>
 <price-bn><price> 39.95</price></price-bn>
 </book-with-prices>
</books-with-prices>
```

# XQuery пример 5 (Grouping + quantifier)

- For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

**XQuery:**

```
<results>
{
 for $a in distinct-values(document("bib.com") //author)
 return <result>
 { $a }
 {
 for $b in document("http://bib.com")/bib/book
 where some $ba in $b/author satisfies deep-equal($ba,$a)
 return $b/title
 }
 </result>
}
</results>
```

XML

distinct-values function returns a sequence of unique atomic values from \$arg.

deep-equal returns true if the \$par1 and \$par2 sequences contain the same values, in the same order

# XQuery пример 5 (Grouping + quantifier)

- For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

**Result:**

```
<results>
 <result>
 <author>
 <last>Stevens</last>
 <first>W.</first>
 </author>
 <title>TCP/IP Illustrated</title>
 <title>Advanced Programming in the Unix environment</title>
 </result>
 <result>
 <author>
 <last>Abiteboul</last>
 <first>Serge</first>
 </author>
 <title>Data on the Web</title>
 </result>

</results>
```

# XQuery пример 6 (Sorting)

- List the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order.

**XQuery:**

```
<bib>
{
 for $b in document("www.bn.com/bib.xml") //book
 where $b/publisher = "Addison-Wesley" and $b/@year > 1991
 return
 <book>
 { $b/@year }
 { $b/title }
 </book>
 sortby (title)
}
</bib>
```

# XQuery пример 6 (Sorting)

- List the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order.

**Result:**

```
<bib>
 <book year="1992">
 <title>Advanced Programming in the Unix
environment</title>
 </book>
 <book year="1994">
 <title>TCP/IP Illustrated</title>
 </book>
</bib>
```

# XQuery пример 7 (Recursion)

- Convert a sample document from "partlist" format to "parttree" format. In the result document, part containment is represented by containment of one `<part>` element inside another. Each part that is not part of any other part should appear as a separate top-level element in the output document.
- partlist.xml:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<partlist>
 <part partid="0" name="car"/>
 <part partid="1" partof="0" name="engine"/>
 <part partid="2" partof="0" name="door"/>
 <part partid="3" partof="1" name="piston"/>
 <part partid="4" partof="2" name="window"/>
 <part partid="5" partof="2" name="lock"/>
 <part partid="10" name="skateboard"/>
 <part partid="11" partof="10" name="board"/>
 <part partid="12" partof="10" name="wheel"/>
 <part partid="20" name="canoe"/>
</partlist>
```

# XQuery пример 7 (Recursion)

- Convert the sample document from "partlist" format to "parttree" format.

**Result:**

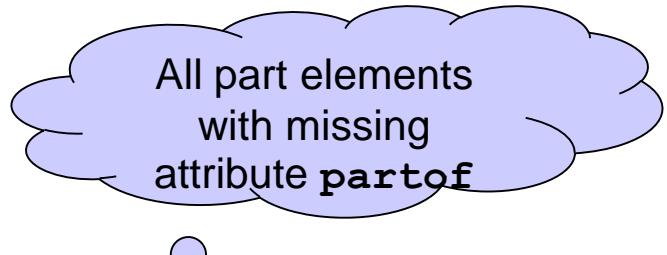
```
<parttree>
 <part partid="0" name="car">
 <part partid="1" name="engine">
 <part partid="3" name="piston"/>
 </part>
 <part partid="2" name="door">
 <part partid="4" name="window"/>
 <part partid="5" name="lock"/>
 </part>
 </part>
 <part partid="10" name="skateboard">
 <part partid="11" name="board"/>
 <part partid="12" name="wheel"/>
 </part>
 <part partid="20" name="canoe"/>
</parttree>
```

# XQuery пример 7 (Recursion)

- Convert a sample document from "partlist" format to "parttree" format.

## XQuery:

```
define function one_level (element $p) returns element
{
 <part partid="{ $p/@partid }" name="{ $p/@name }" >
 {
 for $s in document("partlist.xml")//part
 where $s/@partof = $p/@partid
 return one_level($s)
 }
 </part>
}
<parttree>
{
 for $p in document("partlist.xml")//part[empty(@partof)]
 return one_level($p)
}
</parttree>
```



All part elements  
with missing  
attribute **partof**

# XQuery поддръжка за RDBMS

- Oracle XQuery Engine
  - <http://www.oracle.com/technology/tech/xml/xquery/index.html>
- Introduction to XQuery in SQL Server
  - [http://msdn.microsoft.com/en-us/library/ms345122\(SQL.90\).aspx](http://msdn.microsoft.com/en-us/library/ms345122(SQL.90).aspx)
- Query DB2 XML data with XQuery
  - <http://www.ibm.com/developerworks/data/library/techarticle/dm-0604saracco/>
- DataDirect: Data Integration Suite – MySQL Database Support
  - <http://www.datadirect.com/products/data-integration/datasources/databases/mysql/index.ssp>

# Заключение 1/2

- Основната употреба на XPath е за определяне на части от документа, които да се обработва, но също така XPath изрази могат да се използват и за изчисления или обработка на низове, за проверка на логически условия и др.
- Механизмът, дефиниран от XPath спецификациите е ефективен и гъвкав, но до известна степен е и ограничен. Така например, XPath не позволява сложни търсения с обединения и друга комплексна обработка на XML документи, която може да се реализира процедурно например с използване на DOM.
- Трябва да се подчертва, че изчисляването на XPath израза връща множество от възли, а не XML документ. Ето защо XPath не се изпозва самостоятелно, а винаги като спомагателно средство за адресиране или сравнение на секции от XML документа.

# Заключение 2/2

- XQuery е проста алтернатива на XSLT, JSP, ASP, Servlet, CGI, PHP, ....
- Програмите на XQuery могат да изпълнят повечето задачи, традиционно решавани с посочените по-горе инструменти, но все пак са много по-лесни за учене и по-лесни за писане.
- Възможно е да се разшири XQuery за UPDATE и INSERT в XML база данни
- Все още липсва достатъчна подкрепа за XQuery от страна на индустрията

# Литература

- Jonathan Pinnock, et al. “**Professional XML, 2nd edition**”, ISBN: 1861005059, WROX Publishers, 2001
- Serge Abiteboul, Peter Buneman and Dan Suciu, “**Data on the Web: from Relations to Semistructured Data and XML**”, ISBN 1-55860-622-X, Morgan Kaufmann Publishers, 2000
- World Wide Web Consortium, “**XQuery 1.0. An XML Query Language**”, W3C Working Draft, Apr. 30, 2002
- World Wide Web Consortium, “**XML Path Language (XPath) Version 1.0**”, W3C Recommendation, Nov. 16, 1999
- **Qexo: The GNU Kawa implementation of XQuery**, <http://www.gnu.org/software/qexo/>
- Don Chamberlin, Jonathan Robie, and Daniela Florescu, “**Quilt: An XML Query Language for Heterogeneous Data Sources**”, WebDB 2000, Dallas, May 2000

# XSLT (eXtensible StyleSheet Language for Transformations)



XML

XSLT

Преглед на XSL  
Употреба  
Възможности  
XSLT елементи  
Шаблони  
Манипулиране  
Примери

# Стилови множества (Style Sheet)

- **CSS - Cascading Style Sheet Specification**

- Предоставя прост синтаксис за добавяне на стилове към елементи (в HTML браузъри)

- **DSSSL - Document Style and Semantics Specification Language**

- Международен SGML стандарт за стилове и конвертиране на документи

- **XSL - Extensible Style Language**

- Комбинира черти на DSSSL и CSS, използвайки XML синтаксис

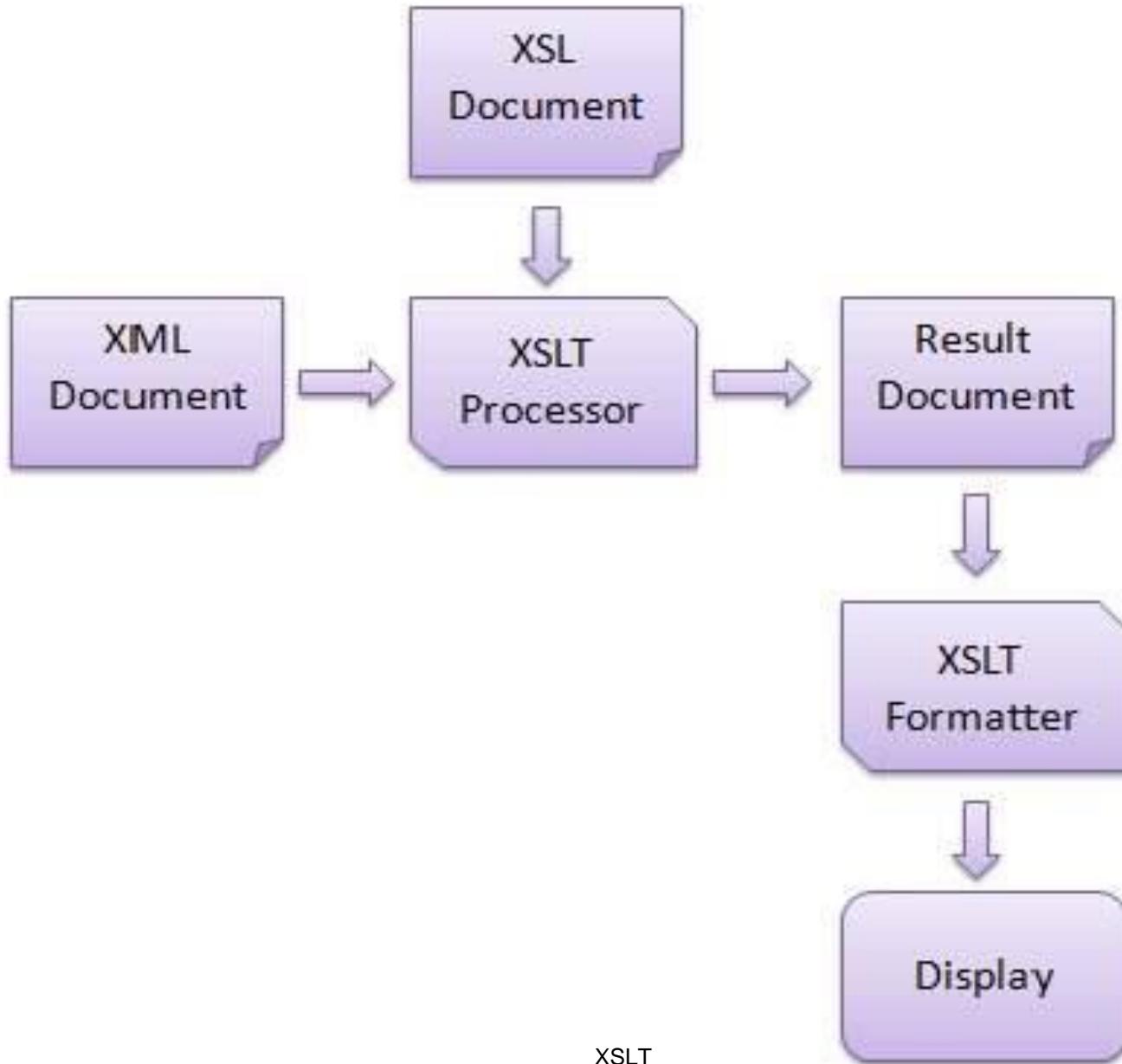
# Какво е XSL?

- **XSL = eXtensible Stylesheet Language**
- Трансформира XML до друг текстов формат, напр.
  - XML
  - HTML
  - Text
- Притежава т.нар. "scripting" engine – XSLT
- За разлика от императивни езици като JavaScript, XSL е *декларативен* => без странични ефекти

# XSL-FO и XSLT

XSL се състои от две части:

- **Extensible Stylesheet Language – Formatting Objects (XSL-FO)** – език за описание на форматирането на данните в XML документ с цел представянето им на различни медии (напр. экран, принтер или мултимедия);
- **Extensible Stylesheet Language for Transformation (XSLT)** – за трансформиране на XML документи с помощта на различни стилове и функции. Най-често се използва за конвертиране на документ от XML формат към документ в HTML формат, обикновен текстов файл или пък например друг XML документ. Полезен е, когато искаме да разделим презентационния слой XML на едно приложение от модела на данните му



# Предназначение на XSLT

Преобразуване на XML в специфичен формат за представяне, като например HTML

Преобразуване от формат, разбираем за едно приложение, във формат, разбираем за друго приложение

# XSLT като декларативен език 1-1

- Процедурен език
  - Софтуерът получава данни и ги обработва стъпка по стъпка
  - Всеки израз или блок изпълнява точно дефинирана задача, реализирайки промени в данните
- Пример: визуализиране на имена в колекция Author

**Java**

```
int index;
for (index = 0; index < allAuthors.Count; index++)
{
 Author thisAuthor = allAuthors[index];
 Console.WriteLine(thisAuthor.FirstName + " " + thisAuthor.LastName);
}
```

**C#**

```
foreach (Author thisAuthor in allAuthors)
{
 Console.WriteLine(thisAuthor.FirstName + " " +
 thisAuthor.LastName);
}
```

**SQL**

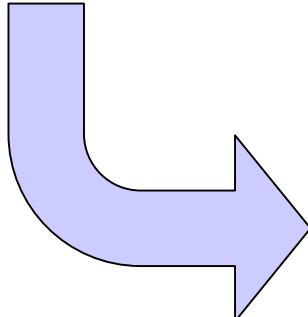
```
SELECT FirstName, LastName FROM Authors;
```



# XSLT като декларативен език 2-2

- Липса на необходимост от деклариране на променлива, с която да се управлява итерацията на елементите в колекцията
- XSLT шаблон

```
<xsl:template match="author" />
<xsl:value-of select="firstName" />
<xsl:value-of select="lastName" />
</xsl:template>
```



```
<authors>
 <author>
 <firstName>Danny</firstName>
 <lastName>Ayers</lastName>
 </author>
 <author>
 <firstName>Joe</firstName>
 <lastName>Fawcett</lastName>
 </author>
 ...
</authors>
```

# XSLT като функционален език

- Императивен език
  - Java, C++, C# и PHP диктуват какво точно трябва да се направи, опитвайки да променят състоянието на обекта, за да представят промените в обстоятелствата
- Пример: промяна на елемент в колекция Author

```
Author authorToEdit = getAuthor(12345);
//Get the required author using their ID
authorToEdit.LastName = "Marlowe"; //Change last name
```
- Функционален език
  - Изходът е резултат от прилагане на една или повече функции върху входа

# Конфигуриране на локална XSLT среда за разработка

- Saxon процесор
  - Поддържа актуална реализация на XSLT
  - Предлага се с отворен достъп (комерсиалните версии са с по-богата функционалност)
  - Предоставя се с версии за Java и .NET
- Уеб сайт: <http://saxon.sourceforge.net/>
- Инсталлиране за .NET
  - Конфигуриране на PATH променливата на средата (`c:\Program Files\Saxonica\SaxonHE9.3N\bin`)
    - Отпада необходимост от специфициране на пълния път до файловете в командния ред при трансформация
  - Команден ред: `Transform -?`
- Инсталлиране за Java
  - Изисква JVM
  - Конфигуриране на CLASSPATH променливата на средата (`<installation path>/saxon9he.jar`)
  - Команден ред: `java net.sf.saxon.Transform -?`
- Документация: <http://www.saxonica.com/documentation/>

# Възможности на XSL/XSLT

- Добавяне на префиксен или суфиксен текст към съществуващо съдържание
- Структурни промени на входното съдържание, като създаване, отстраняване, редактиране, пренареждане и сортиране на XML елементи
- Многократно използване на елементно или атрибутно съдържание на друго място в документа
- Трансформиране на данни между XML формати
- Определяне на XSL форматиращи обекти и на други средства за представяне на съдържанието в дадена медиа (напр. CSS), с цел да се прилагат към даден елемент

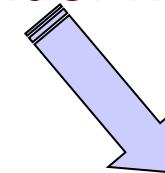
# Базови XSLT елементи

- <xsl:stylesheet>
  - Задава обхват на XSLT документа и осигурява конфигуриране на параметри
- <xsl:template>
  - Специфицира кои елементи трябва да бъдат обхванати от обработката
  - Дефинира какво ще бъде добавено в изходния документ
- <xsl:apply-templates>
  - Взима решение кои елементи ще бъдат обработени
- <xsl:value-of>
  - Изчислява израз и добавя резултата в изхода
- <xsl:for-each>
  - Осигурява групова обработка на елементи по еднотипен начин

# Прост пример

- Файл **data.xml**:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="render.xsl"?>
<message>Howdy!</message>
```



- Файл с дефиниция на трансформация **render.xsl**:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <!-- one rule, to transform the input root (/) -->
 <xsl:template match="/">
 <html><body>
 <h1><xsl:value-of select="message"/></h1>
 </body></html>
 </xsl:template>
</xsl:stylesheet>
```

# Файл с разширение .xsl

- Всеки XSLT документ има .xsl разширение
- XSLT документа започва с:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/
 XSL/Transform">
```
- Може да съдържа шаблони, като напр.:  

```
<xsl:template match="/" ... </xsl:template>
```
- Завършва с:  

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

XML

Важно:

Елементите `<xsl:stylesheet>` и `<xsl:transform>` са синоними и могат да се използват взаимозаменямо.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<person>
 ▼<name>
 <title>Bai</title>
 <first>Ganyo</first>
 <middle>Son of his Father</middle>
 <last>Balkanski</last>
 </name>
 <profession>Trader, Image Maker, Politician</profession>
 <position>President of Dirty Business UnLtd.</position>
 ▼<r sum >
 ▼<html>
 ▼<head>
 <title>Resume of G. Balkanski</title>
 </head>
 ▼<body>
 <h1>Ganyo Balkanski</h1>
 ▼<p>
 Who didn't know about Bai Ganyo, who has not heard of him?
 </p>
 </body>
 </html>
 </r sum >
</person>
```

# Без xsl трансформация...

# Намиране на текста message

- Шаблонът `<xsl:template match="/">` задава селектиране на целия входен документ, т.е. на *root* възела на XML дървото
- Вместо това,
  - `<xsl:value-of select="message"/>` селектира преките наследници на *message*
  - Това са Xpath изрази, както и аналогичните им:
    - `./message`
    - `./message/text()` (`text()` е **XPath функция**)
- Всеки *XSL stylesheet* трябва да бъде добре конструиран XML документ

# Как работи?

- XSL дефиницията е:

```
<xsl:template match="/">
 <html><body>
 <h1><xsl:value-of select="message"/></h1>
 </body></html>
</xsl:template>
```

- Шаблонът **<xsl:template match="/">** избира корена
- **<html><body>** **<h1>** се записва в изходния файл
- Съдържанието на **message** се записва в изходния файл
- **</h1> </body></html>** се записва в изходния файл
- Резултатният файл е:  
**<html><body>**  
  **<h1>Howdy!</h1>**  
**</body></html>**

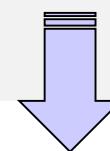
# Задаване на XSL трансформация в XML документ

- <?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>
- Тази инструкция задава на браузъра да извика XSLT процесор за изпълнение на XSL трансформацията, зададена в документа със стилове **stylesheet.xsl**, който в случая е от тип **text/xsl**, но би могъл да бъде и от друг тип, напр. **text/css**.

# Елемент <xsl:output>

- Позволява контрол над изходния документ
- Разполага се веднага след <xsl:stylesheet>
- Синтаксис:

```
<xsl:output method="xml or html or text"
version="version"
encoding="encoding"
omit-xml-declaration="yes or no"
standalone="yes or no"
cdata-section-elements="CDATA sections"
indent="yes or no" />
```



Тук задаваме всички елементи, които ще имат CDATA съдържания, като напр.

`<elem><![CDATA[<&>&]]></elem>`

# Елемент <xsl:template>

- Изпълнява кода в шаблона всеки път, когато срецне елемент, който съответства на специфицирания в атрибута match
- match="/"
  - Шаблонът се изпълнява, когато се срецне кореновият елемент на документа
  - Използва като контекст кореновия елемент
  - Аналог на синтаксиса XSL Pattern в XPath

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <!-- basic output here -->
 </xsl:template>
</xsl:stylesheet>
```

# Търсене на елемент с XSLT

- ```
<xsl:template match="exon">  
    We have found the EXON tag!  
</xsl:template>
```

Шаблон, дефиниран
чрез елемента
<xsl:template>

Атрибутът **<match>**
задава шаблон върху
входното дърво

Шаблонът ще се приложи над
всеки възел **exon** във входното
дърво.

Елемент <xsl:value-of>

- Извличане на информация от елементи от началното дърво става чрез елемента
- <xsl:value-of select="XPath-expression"/>**
- Атрибутът **select='.'** избира контекстния възел и всички негови наследници (за разлика от функцията text() !!!)
- Пример:
- <xsl:template match="exon">**
- <xsl:value-of select=". "/>**
We have displayed the contents of the EXON tag!
- </xsl:template>**

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
<persons >
  <person id="123456" age="53">
    <name title="Bai">
      <first>Ganyo</first>
      <last>Balkanski</last>
    </name>
    <profession>Trader, Image Maker, Politician</profession>
    <descr>
Who didn't know about Bai Ganyo,
who has not heard of him? ...
    </descr>
  </person>
  <person id="345" age="29">
    <name title="Mr.">
      <first>Balkan</first>
      <last>Ganyov</last>
    </name>
    <profession>Facilitator</profession>
    <descr>
Facilitator of people
who are in the middle of nowhere...
    </descr>
  </person>
</persons>
```

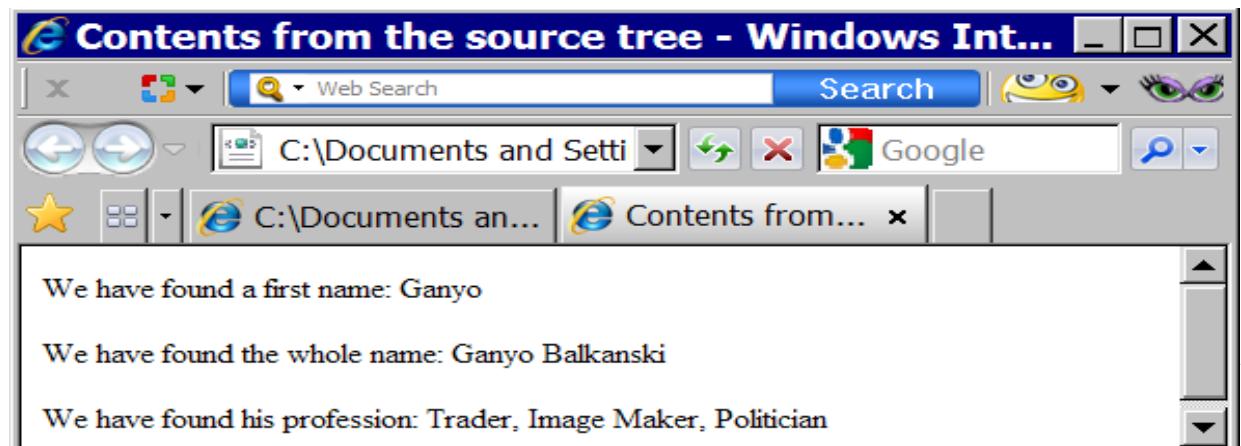
XML

XSLT

За този
документ...

...Трансформация без шаблони

```
• <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <html>
        <head>
            <title>Contents from the source tree</title>
        </head>
        <body>
            <p>We have found a first name:
            <xsl:value-of select="/persons/person/name/first"/></p>
            <p>We have found the whole name:
            <xsl:value-of select="/persons/person/name/. /></p>
            <p>We have found his profession:
            <xsl:value-of select="/persons/person/profession"/></p>
        </body>
    </html>
</xsl:stylesheet>
```



Търсене на атрибут

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/>
<html>
    <head>
        <title>Contents from the source tree</title>
    </head>
    <body>
        <p>We have found a first name:<br/>
<xsl:value-of select="/persons/person/name/first"/></p>
        <p>We have found his age:<br/>
<xsl:value-of select="/persons/person/@age"/></p>
        <p>We have found a second name:<br/>
<xsl:value-of select="/persons/person[@age='29']/name/first"/></p>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>
```



Друг XML пример

```
<?xml version="1.0"?>
<People>
    <Person diedDate="1965-01-24" bornDate="1874-11-30">
        <Name>Winston Churchill</Name>
        <Description> Winston Churchill was a mid 20th century British politician
who became famous as Prime Minister during the Second World War.
    </Description>
    </Person>
    <Person diedDate="1984-10-31" bornDate="1917-11-19">
        <Name>Indira Gandhi</Name>
        <Description> Indira Gandhi was India's first female prime minister and
was assassinated in 1984. </Description>
    </Person>
    <Person diedDate="1963-11-22" bornDate="1917-05-29">
        <Name>John F. Kennedy</Name>
        <Description> JFK, as he was affectionately known, was a United States
president who was assassinated in Dallas, Texas. </Description>
    </Person>
</People>
```

Роля на контекста в XSLT

- XSLT обработката винаги се извършва в контекста на определен елемент
 - match="People/Person"
 - Връща като резултат 3 <Person> элемента
 - Селектирането започва от кореновия елемент, преминава през People елемента и завършва в Person елемента
 - match="Person"
 - Не връща резултат
 - Селектирането започва от кореновия елемент и спира поради липсата на Person елемент
 - match="People"
 - Връща като резултат 1 <People> елемент
 - Селектирането започва от кореновия елемент и завършва в People елемента

Работа с шаблони 1/2

- шаблоните в XSLT се дефинират с елемента **<xsl:template>**, чийто атрибут **match** задава образец за съответствие на част от йерархията на началното дърво.
- Стойността на образца представлява XPath израз и се използва за адресиране на секции от началното дърво.

Работа с шаблони 2/2

- Освен параметъра **match**, един шаблон може да има и следните по-важни параметри:
 - ✓ **name** – задава QName, с цел извикването на такъв именован шаблон по име чрез **call-template**;
 - ✓ **priority** – задава числовой приоритет, който да се ползва за разрешаване на конфликтите при вземане на решение кой от няколко възможни шаблона да се ползва за даден възел от дървото. Стойността му по подразбиране е 0.5;
 - ✓ **mode** – дава възможност даден възел да бъде обработван многократно, с различен крайен резултат в зависимост от стойността на **mode**.

Извикване на шаблони 1/2

- За извикване на шаблони се използва XSLT инструкцията
- <xsl:apply-templates select="XPath expression" />**
- Активира рекурсивна обработка на всички наследници на контекстния възел
- Пример:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/persons">
    <html>
      <head>
        <title>Found
          <xsl:value-of select="count(person)" />
          people.
        </title>
      </head>
```

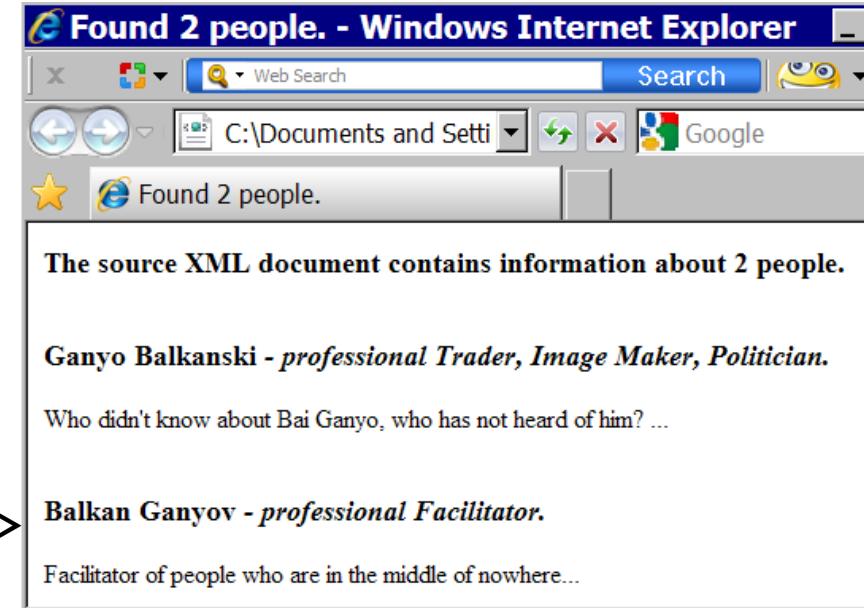
Извикване на шаблони 2/2

```
<body>
<h3>The source XML document contains information about
<xsl:value-of select="count(person)" /> people.</h3>
<br />
<xsl:apply-templates select="person" />
</body>
</html>
</xsl:template>
<xsl:template match="person">
<h3> <xsl:value-of select="name" />
<i> - professional
<xsl:value-of select="profession" />.</i></h3>
<p><xsl:value-of select="descr" /></p><br />
</xsl:template>
</xsl:stylesheet>
```

```

<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
<persons >
    <person id="123456" age="53">
        <name title="Bai">
            <first>Ganyo</first>
            <last>Balkanski</last>
        </name>
        <profession>Trader, Image Maker, Politician</profession>
        <descr>
Who didn't know about Bai Ganyo,
who has not heard of him? ...
        </descr>
    </person>
    <person id="345" age="29">
        <name title="Mr.">
            <first>Balkan</first>
            <last>Ganyov</last>
        </name>
        <profession>Facilitator</profession>
        <descr>
Facilitator of people
who are in the middle of nowhere...
        </descr>
    </person>
</persons>

```



Элемент <xsl:apply-templates>

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <head>
    <title>Famous People</title>
  </head>
  <body>
    <h1>Famous People</h1>
    <hr />
    ...
  </body>
</html>
</xsl:template>
...
</xsl:stylesheet>
```

```
<ul>
  <xsl:apply-templates
select="People/Person" />
</ul>
```

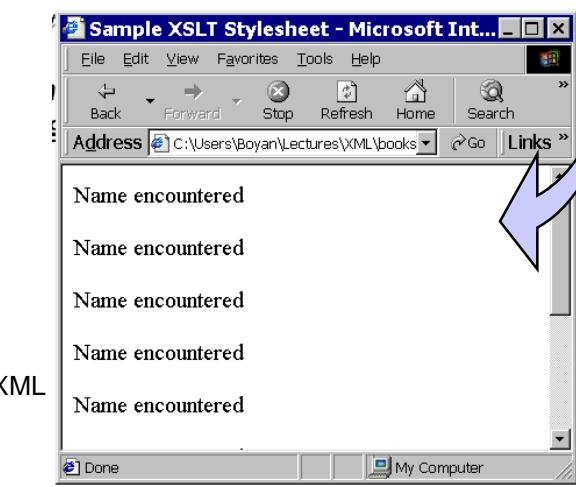
Apply-Templates елемент – пример 1

```
<?xml version="1.0"?>
<simple>
  <name>John</name>
  <name>David</name>
  <name>Andrea</name>
  <name>Ify</name>
  <name>Chaulene</name>
  <name>Cheryl</name>
  <name>Shurnette</name>
  <name>Mark</name>
  <name>Carolyn</name>
  <name>Agatha</name>
</simple>
```



```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head><title>Sample XSLT Stylesheet </title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="name">
    <p>Name encountered</p>
  </xsl:template>
</xsl:stylesheet>
```

XSLT in Details

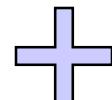


XML

34

Apply-Templates елемент – пример 2

```
<?xml version="1.0"?>
<simple>
  <name>John</name>
  <name>David</name>
  <name>Andrea</name>
  <name>Ify</name>
  <name>Chaulene</name>
  <name>Cheryl</name>
  <name>Shurnette</name>
  <name>Mark</name>
  <name>Carolyn</name>
  <name>Agatha</name>
</simple>
```



```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head><title>Sample XSLT Stylesheet </title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="name">
    <p><xsl:value-of select=". "> </p>
  </xsl:template>
</xsl:stylesheet>
```



Работа с шаблони

- По подобен начин шаблонът
- `<xsl:template match=".//name">`
- `<xsl:apply-templates select="name[@title='Mr.']/>`
- `</xsl:template>`
- адресира наследници на контекстния възел '.' с име **name** (изразите ".//name" и "name" са еквивалентни) задава прилагане на най-подходящия шаблон за всички възли с име **name** и с атрибут **title** имащ стойността '**Mr.**'.

Mode атрибут

- Част от елемента template; разрешава извикване на избран шаблон.
- Определя кое match правило да се използва

`<xsl:template match="chapter/title">
 <html:h1><xsl:apply-templates/></html:h1>
</xsl:template>`

`<xsl:template match="chapter/title" mode="h3">
 <html:h3><xsl:apply-templates/></html:h3>
</xsl:template>`

`<xsl:template match="intro">
 <xsl:apply-templates
 select="//chapter/title" mode="h3"/>
</xsl:template>`

This template is
NOT instantiated
by apply-
templates though
the MATCH is the
same!!

This template
is instantiated
by apply-
templates

Именувани шаблони 1/4

- Осигуряват елементарна обработка
- Атрибут **name**

```
<xsl:template name="iso8601DateToDisplayDate">  
  <xsl:param name="iso8601Date" />  
  <xsl:variable name="yearPart" select="substring($iso8601Date,1,4)" />  
  <xsl:variable name="monthPart" select="substring($iso8601Date,6,2)" />  
  <xsl:variable name="datePart" select="substring($iso8601Date,9,2)" />  
  <xsl:value-of select="concat($datePart, '/', $monthPart, '/',  
    $yearPart)" />  
</xsl:template>
```

YYYY-
MM-DD

DD/MM/
YYYY

Именувани шаблони 2/4

- Елемент `<xsl:param>`
 - Осигуряват подаване на стойности към шаблона
- Елемент `<xsl:variable>`
 - Променлива, която може да бъде инициализирана еднократно през жизнения си цикъл
- Функция XPath `substring`
 - `substring($iso8601Date, 6, 2)`
- Функция XPath `concat`
 - `concat($datePart, '/', $monthPart, '/', $yearPart)`

Именувани шаблони 3/4

- Инициализиране на променлива

- Атрибут select

```
<xsl:variable name="yearPart" select="..." />
```

- Инициализация със съдържание

```
<xsl:variable name="myVariable">  
  <myElement>Some content</myElement>  
</xsl:variable>
```

- Изиска създаване на ново дърво и добавяне на външен възел

- Достъп до променлива

- Използва се знакът \$

Именувани шаблони 4/4

- Обхват на променлива

- Определя се в зависимост от елемента, в който е дефинирана

```
<xsl:template name="usingVariables">
  <xsl:for-each select="someElements/someElement">
    <xsl:variable name="demo" select="'Some text'" />
    <!-- the line is okay as $demo is in scope --&gt;
    &lt;xsl:value-of select="concat(someElement, $demo)" /&gt;
  &lt;/xsl:for-each&gt;
  &lt;!--the line is an error as $demo is out of scope--&gt;
  &lt;xsl:value-of select="$demo" /&gt;
&lt;/xsl:template&gt;</pre>
```

Елемент <xsl:call-template>

- Елемент <xsl:call-template>
 - Предизвиква изпълнение на шаблон
 - Атрибут name дефинира името на извиквания шаблон
- Елемент <xsl:with-param>
 - Подаване стойност към <xsl:param> елемент
 - Стойността на параметъра се определя от атрибута select

```
<td>
  <xsl:call-template name="iso8601DateToDisplayDate">
    <xsl:with-param name="iso8601Date" select="@bornDate" />
  </xsl:call-template>
</td>
<td>
  <xsl:call-template name="iso8601DateToDisplayDate">
    <xsl:with-param name="iso8601Date" select="@diedDate" />
  </xsl:call-template>
</td>
```

Шаблони с параметър

- Елементът **param** е специална променлива:

- <xsl:param name="prefix">default</xsl:param>
- <xsl:with-param name="prefix">new</xsl:with-param>

- Елементът **call-template** предава новите **param** стойности към шаблона

Call to this template

```
<xsl:template match="name">
  <xsl:call-template name="salutation">
    <xsl:with-param name="greet">Hello </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

The template

```
<xsl:template name="salutation">
  <xsl:param name="greet">Dear </xsl:param>
  <xsl:value-of select="$greet"/>
  <xsl:apply-templates/>
</xsl:template>
```

XSLT

Инициализиране на стойности за атрибути от променлива

```
<tr style="{$rowCSS}" >
```

```
<xsl:value-of select="$rowCSS" />
```

Голяма скоба се поставя, ако атрибутът не очаква XPath израз като стойност

Attribute Value Templates (AVT)

Шаблони по подразбиране (Default Templates)

- Съществуват вградени (built-in) XSLT templates:

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="example6.3.2.3.xsl"?>
<persons>
  <person id="123456" age="53">
    <name title="Bai">
      <first>Ganyo</first>
      <last>Balkanski</last>
    </name>
    <profession>Trader, Image Maker, Politician</profession>
    <descr>
      Who didn't know about Bai Ganyo, who has not heard of him? ...
    </descr>
  </person>
  <person id="345" age="29">
    <name title="Mr.">
      <first>Balkan</first>
      <last>Ganyov</last>
    </name>
    <profession>Facilitator</profession>
    <descr>
      Facilitator of people who are in the middle of nowhere...
    </descr>
  </person>
</persons>
```

Стиловият документ обаче не съдържа нито една инструкция за трансформация:

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
```

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

Резултат



- Представяне на документ в браузер с използване на празен XSLT документ

Шаблони по подразбиране (Default Templates)

- Съществуват вградени (built-in) XSLT templates.
- XSLT процесорът ги ползва, ако не намери подходящ шаблон.
- Напр., ако няма шаблон за корена на документа, XSLT предоставя такъв по подразбиране, който пък прилага всички съществуващи шаблони:

```
<xsl:template match="*//>
  <xsl:apply-templates/>
</xsl:template>
```
- За всеки елемент в документа (вкл. и за корена) се извиква `<xsl:apply-templates>`

Други шаблони по подразбиране

- Вграден шаблон за текстови елементи и атрибути:

```
<xsl:template match="text()|@*>
  <xsl:value-of select="."/>
</xsl:template>
```
- Този шаблон просто добавя стойността на текстов възел или атрибут към резултата.
- Затова получаваме резултат като този:



Създаване на елементи – елемент **<xsl:element>**

- Елемент се създава чрез тага 'Element'
- Удобно е да се ползва с променливи

The
template

```
<xsl:template name="CreateHeader">  
  <xsl:param name="level">3</xsl:param>  
  <xsl:element namespace="html" name="h{$level}">  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```

Default
value

Call to
this
template

```
<xsl:template match="title">  
  <xsl:call-template name="CreateHeader">  
    <xsl:with-param name="level">1</xsl:with-param>  
  </xsl:call-template>  
</xsl:template>
```

The new
value

Динамично създаване на елементи

- Пример: имената на елементите в резултатното дърво ще са съдържанието на елементите в изходящото дърво:

```
<xsl:template match="name">
    <xsl:element name="{.}">
        Nice person!
    </xsl:element>
</xsl:template>
```

- Напр. за `<name>Milena</name>` -> резултатът е:

`<Milena>Nice person!</Milena>`

Пример: трансформиране на атрибути до елементи!

Начален XML документ:

```
<?xml version="1.0"?>
<people>
  <name first="John" middle="Fitzgerald Johansen" last="Doe"/>
  <name first="Franklin" middle="D." last="Roosevelt"/>
  <name first="Alfred" middle="E." last="Neuman"/>
  <name first="John" middle="Q." last="Public"/>
  <name first="Jane" middle="" last="Doe"/>
</people>
```

XSL документ

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" encoding="UTF-8"/>

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

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

<xsl:template match="@*"
  <xsl:element name="{local-name()}> <xsl:value-of select=". /> </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Връща името на
текущия възел без
namespace префикс

Резултатен XML документ

- <?xml version="1.0"?>
- <people>
- <name>
- <first>John</first>
- <middle>Fitzgerald Johansen</middle>
- <last>Doe</last>
- </name>
-
- <name>
- <first>Jane</first>
- <middle></middle>
- <last>Doe</last>
- </name>
- </people>

Копиране на елементи

- Елементът 'copy' копира контекстния възел

без атрибутите и наследниците му!!!

- <xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy>
 Header: <xsl:apply-templates/>
 </xsl:copy>
 </xsl:template>

- Оригиналните атрибути не се запазват

- Атрибути създаваме чрез <xsl:attribute>

- <xsl:template match="h1|h2|h3|h4|h5|h6|h7">
 <xsl:copy>
 <xsl:attribute name="style">purple</xsl:attribute>
 Header: <xsl:apply-templates/>
 </xsl:copy>
 </xsl:template>

attribute-set елемент

- Използва се за създаване на групи от атрибути за по-нататъшна употреба (чрез **use-attribute-sets**)

```
<xsl:attribute-set name="class-and-color">
  <xsl:attribute name="class">standard</xsl:attribute>
  <xsl:attribute name="color">red</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
  <xsl:copy use-attribute-sets="class-and-color">
    Header: <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

copy-of елемент

Копира фрагменти от входното XML дърво без загуба на атрибути и съдържани под-елементи (*deep copy*):

```
<xsl:template match="body">  
  <body>  
    <xsl:copy-of select="//h1 | //h2" />  
    <xsl:apply-templates/>  
  </body>  
</xsl:template>
```

xsl:for-each (loop in XSLT)

- Елементът **<xsl:for-each>** селектира всеки един XML елемент от определено множество възли (node-set):

```
<xsl:template match="/">
<html> <body>
    <h2>My CD Collection</h2>
    <table border="1">
        <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
            <tr>
                <td><xsl:value-of select="title"/></td>
                <td><xsl:value-of select="artist"/></td>
            </tr>
        </xsl:for-each>
    </table>
</body> </html>
</xsl:template>
```

Условия - if

- Елементът `<xsl:if>`

- прилага се само ако условието му е истина
- не предоставя else клауза
- атрибут test получава стойност XPath израз (True или False)

```
<xsl:template match="para">
  <html:p>
    <xsl:if test="position() = 1">
      <xsl:attribute name="style">color: red</xsl:attribute>
    </xsl:if>

    <xsl:if test="position() > 1">
      <xsl:attribute name="style">color: blue</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </html:p>
</xsl:template>
```

Source: http://www.w3schools.com/Xsl/tryxslt.asp?xmlfile=cdcatalog&xsltfile=tryxsl_if

Приимер: избор на title и artist само АКО
цена на CD е по-висока от 10

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:template match="/">
  <html> <body> <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32"> <th>Title</th> <th>Artist</th> </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:if test="price > 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table> </body> </html>
</xsl:template>
</xsl:stylesheet>
```

Други примери:

http://www.w3schools.com/xsl/el_if.asp



TRY IT FOR FREE

[Start my free trial](#)

fonts.com web
webfonts.fonts.

XML Code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<catalog>
    <cd>
        <title>Empire Burlesque</title>
        <artist>Bob Dylan</artist>
        <country>USA</country>
        <company>Columbia</company>
        <price>10.90</price>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <price>9.90</price>
        <year>1988</year>
    </cd>
    <cd>
        <title>Greatest Hits</title>
        <artist>Dolly Parton</artist>
        <country>USA</country>
        <company>RCA</company>
        <price>9.90</price>
    </cd>

```

[Edit and Click Me >>](#)
XSLT Code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <body>
            <h2>My CD Collection</h2>
            <table border="1">
                <tr bgcolor="#9acd32">
                    <th>Title</th>
                    <th>Artist</th>
                </tr>
                <xsl:for-each select="catalog/cd">
                    <xsl:if test="price>10">
                        <tr>
                            <td><xsl:value-of select="title"/></td>
                            <td><xsl:value-of select="artist"/></td>
                        </tr>
                    </xsl:if>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
```

Your Result:

My CD Collection

Title	Artist
XML Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore

XSLT

Условия – `xsl:choose` (1/2)

- `<xsl:choose>` се ползва заедно с `<xsl:when>` и `<xsl:otherwise>`
- задава многократни условни тестове

```
<xsl:template match="para">
  <html:p>
    <xsl:choose>
      <xsl:when test="position() = 1">
        <xsl:attribute name="style">color: red</xsl:attribute>
      </xsl:when>
      <xsl:otherwise test="position() > 1">
        <xsl:attribute name="style">color: blue</xsl:attribute>
      </xsl:otherwise>
    <xsl:apply-templates/>
  </xsl:choose>
</html:p>
</xsl:template>
```

опционално

Условия – `xsl:choose` (2/2)

- `<xsl:choose>`
- `<xsl:when test="salary[number(.) > 2000]">`
A big number for BG
- `</xsl:when>`
- `<xsl:when test="salary[number(.) > 1000]">`
A medium number for BG
- `</xsl:when>`
- `<xsl:otherwise>A small number`
- `</xsl:otherwise>`
- `</xsl:choose>`

Елементът `<xsl:sort>` 1/2

- Използва се като дъщерен елемент на `<xsl:apply-templates>` или `<xsl:for-each>` елементите
- Атрибутът `select`
 - XPath израз рефериращ възел(и) за сортиране
- Атрибутът `data-type`
 - Начин на тълкуване на стойностите за възлите (обикновено се използва текст или число)
- Атрибутът `order`
 - Ред на сортиране (възходящ или низходящ)

Елементът `<xsl:sort>` 2/2

- Пример: сортиране по дата
 - Необходимост от преобразуване на датата в число с функцията `translate()`

```
translate('The first of the few ', 'fiw', 'wot')
```

The worst of the wet.

```
translate(@bornDate, ' - ', '')
```

- Използване на елемента `<xsl:sort>`

```
<xsl:apply-templates select="People/Person">  
  <xsl:sort select="translate(@bornDate, ' - ', '')"  
            data-type="number"/>  
</xsl:apply-templates>
```

Два механизма за комбиниране на набори от стилове

- механизъм за включване, който позволява да бъдат комбинирани стилове, без да се променя семантиката на стиловете, които се съчетават - посредством елемента
 - **<xsl:include href=uri-reference />**
- механизъм за внасяне, който позволява на стилове да се отменят един друг - с използване на елемента
 - **<xsl:import href=uri-reference />**

XML

Важно:

Както включването, така и внасянето на стилове е разрешено само на най-високо ниво на элементната йерархия в XSLT документа.

Елементът <xsl:include> 1/2

- Извършва вмъкване на XSLT документ в друг XSLT документ
- Осигурява изграждане на модули
- Предимства
 - Преизползващост на кода
 - Спестяване на усилия при необходимост от промяна

Елементът `<xsl:include>` 2/2

```
<xsl:include href="DateTemplates.xslt" />
```

- Атрибут href
 - Задава местоположението и името на XSLT документа
- Начин на обработка от XSLT процесора
 - Премахва се външния `<xsl:stylesheet>` елемент
 - Изгражда в паметта документ, който включва основния документ и добавените документи с елемента `<xsl:include>`

Елементът `<xsl:import>`

- Осигурява функционалност подобна на `<xsl:include>`
- Ако импортираният шаблон е в конфликт с някой от шаблоните в основния XSLT документ, то той получава по-нисък приоритет
- Приложим е при многократна обработка на възли
 - Пример: изграждане на съдържание или резюме
 - Използват се няколко `<xsl:template>` элемента с атрибут mode

XSLT 2.0

Строго типизиране

Потребителски дефинирани функции

Множество изходни документи от една трансформация

Едновременна обработка на множество документи: функция collection()

Поддръжка на групиране

Обработка на файлове, които не са в XML формат, например CSV

По-добра обработка на текст: регулярни изрази

Поддръжка на XPath 2.0

Типове данни в XSLT 2.0

- XSLT 1.0
 - Текстови, числени и булеви стойности
- XSLT 2.0
 - Поддръжка множество типове, базирани на XML Schema: integer, double, decimal, day, month, year и др.
 - Възможност за импортиране на типове от други XML схеми
 - Поддръжка на 2 типа процесори
 - basic
 - schema-aware
 - Генериране на грешки от процесора при несъответствие на типовете
 - <http://www.w3.org/TR/xpath-functions/#datatypes>

Елемент <xsl:function> - дeфиниране на функция

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 xmlns:myData="http://wrox.com/namespaces/embeddedData"
                 xmlns:myFunc="http://wrox.com/namespaces/functions/datetime"
                 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 exclude-result-prefixes="myFunc">
<xsl:variable name="thisDocument" select="document('')"/>
...
<xsl:function name="myFunc:iso8601DateToDisplayDate" as="xs:string">
    <xsl:param name="iso8601Date" as="xs:date" />
    <xsl:variable name="yearPart" select="year-from-date($iso8601Date)"
as="xs:integer" />
    <xsl:variable name="monthPart" select="month-from-date($iso8601Date)"
as="xs:integer" />
    <xsl:variable name="monthName"
select="$thisDocument/xsl:stylesheet/myData:Months/Month[@index =
number($monthPart)]"
/>
<xsl:variable name="datePart" select="day-from-date($iso8601Date)"
as="xs:integer" />
    <xsl:value-of select="concat($datePart, ' ', $monthName, ' ', $yearPart)" />
</xsl:function>
</xsl:stylesheet>
```

Елемент <xsl:function> - извикване на функция

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
    xmlns:myFunc="http://wrox.com/namespaces/functions/datetime">  
...  
<tr style="{$rowCSS}">  
    <td style="{$nameCSS}"><a name="Person{position()}">  
<xsl:value-of select="Name" /></a></td>  
    <td><xsl:value-of select="myFunc:iso8601DateToDisplayDate(@bornDate)" />  
    </td>  
    <td><xsl:value-of select="myFunc:iso8601DateToDisplayDate(@diedDate)" />  
    </td>  
    <td> <xsl:value-of select="Description" /></td>  
</tr>
```

.NET

```
transform -s:people.xml -xsl:peopleToHtml-  
UsingFunctions.xslt  
-o:people-usingFunctions.html
```

Java

```
java net.sf.saxon.Transform -s:people.xml  
-xsl:peopleToHtml-UsingFunctions.xslt -  
o:people-usingFunctions.html
```

Елементът `<xsl:function>` 1/2

- **Дефиниция**

```
<xsl:function name="myFunc:iso8601DateToDisplayDate" as="xs:string">
  <xsl:param name="iso8601Date" as="xs:date" />
  <!-- rest of function -->
</xsl:function>
```

- **Атрибутът name**

- Включва префикс, рефериращ към пространство с имена

- **Атрибутът as**

- Специфицира типа на резултата

- **Елементът `<xsl:param>`**

- Дефинира параметър на функцията от определен тип, зададен с атрибут **as**

елементът `<xsl:function>` 2/2

- Елементът `<xsl:variable>`
 - Дефинира променлива във функцията от определен тип, зададен с атрибут **as**

- Функцията `year-from-date()`

```
<xsl:variable name="yearPart"  
    select="year-from-date($iso8601Date)" as="xs:integer" />
```

- Функцията `format-date()`

```
<xsl:value-of select="format-date(@bornDate, '[D1] [MNn] [Y]')"/>
```

Използване на XSLT при клиента

- Ако браузърът поддържа XSLT, то тази технология може да се ползва за трансформиране на документа до XHTML в самия браузър.
- По-гъвкаво решение е да се ползва за трансформацията и JavaScript, с цел:
 - Тестване на специфични за браузъра черти
 - Използване на различни style sheets според браузъра и нуждите на потребителя

Пример (<http://www.w3schools.com/xsl>)

```
...
function displayResult() {
    xml=loadXMLDoc("cdcatalog.xml");
    xsl=loadXMLDoc("cdcatalog.xsl");
    // code for IE
    if (window.ActiveXObject) {
        ex=xml.transformNode(xsl);
        document.getElementById("example").innerHTML=ex;
    } // code for Mozilla, Firefox, Opera, etc.
    else if (document.implementation &&
document.implementation.createDocument) {
        xsltProcessor=new XSLTProcessor();
        xsltProcessor.importStylesheet(xsl);
        resultDocument = xsltProcessor.transformToFragment(xml,document);
        document.getElementById("example").appendChild(resultDocument);
    }
}
```

Сървърна XSLT

- Не всички браузъри поддържат XSLT =>
- Разумно решение е да трансформиране XML документа до XHTML на сървъра
- Отново можем да отразим при трансформацията специфични черти на даден браузър
- Бързина и универсалност

DOM спрямо XSL

- За по-сложно преструктуриране и сортиране на документа, използваме DOM
- При DOM ние парсваме XML документа, после изпълняваме код за манипулиране на DOM дървото (напр. на Java) според дадени изисквания.
- Този код има пълен достъп до методите на DOM и на дървото, така че не сме ограничени от лимитираните възможности на XSL

Алтернатива на XSLT: W3C XQuery (a query language for XML)

- XQuery се ползва за XML публикуване на Web съобщения, динамични Уеб сайтове и др. приложения.
- Имплементация: DataDirect Xquery (<https://www.progress.com/xquery/xml>)
- Повечето от функционалността на XQuery, като напр. аритметични оператори, сравнения, работа с функции и др. са познати на програмистите
- Три отличителни особености на XQuery:
 - **Path изрази** - позволяват локиране на части от XML документ.
 - **XML конструктори** - за създаване на произволен XML документ.
 - **FLWOR** (произнасяно "flower") **изрази**: For, Let, Where, Order By, and Return – разрешават комбиниране на данни за създаване на нови XML структури. Подобни са на SQL Select statements и техните From и Where клаузи, но са адаптирани за XML обработка.

За повече информация – вижте спецификациите

Part	Date	Status	URL
XSL-Formatting Objects ver. 2.0	04.02.2010	Version 2.0 W3C Working Draft	http://www.w3.org/TR/xslfo20/
XSL Transformations (XSLT) ver. 2.1	11.05.2010	Version 2.1 W3C Working Draft	http://www.w3.org/TR/xslt-21/
XML Path Language (XPath) ver. 2.0	23.01.2007	Version 2.0 W3C Working Draft	http://www.w3.org/TR/xpath20/

Императивна обработка на XML съдържание чрез Document Object Model (DOM)

Предимства на DOM

DOM интерфейси

Използване

Пример

DOM, XSL и SAX

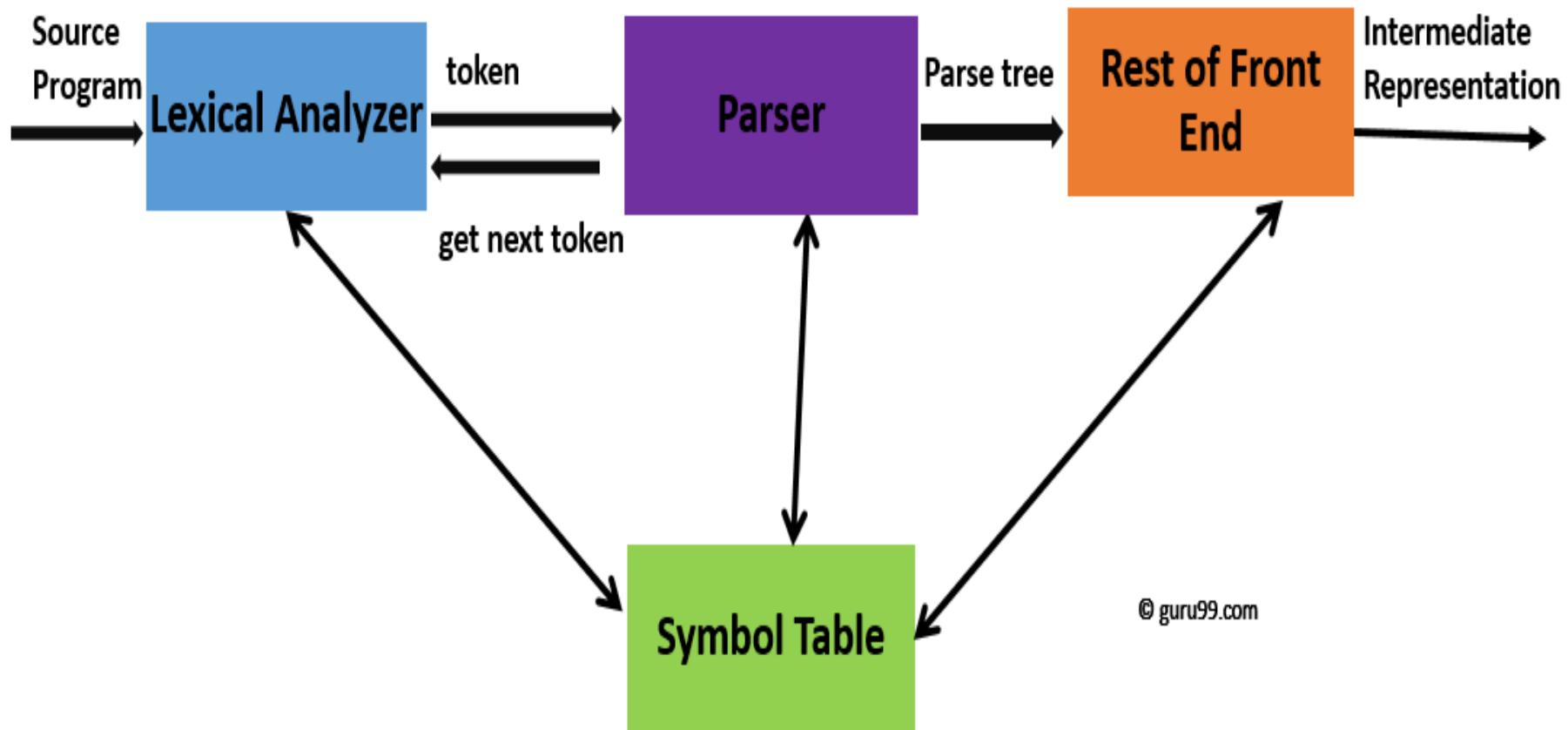


Анализ (разбор,
парсване) на текст

DOM



Процес на анализ (разбор, парсване) на текст



Използване на XML Parser

- Цел и приложение
- Три основни стъпки в използването на XML парсер:
 - Създаване на парсер-обект
 - Предаване на XML документа на парсера
 - Обработка на резултатите
- Принципно, генерацията на XML или друг формат е извън обхвата на парсерите

Типове парсъри

- Съществуват различни групи от парсери:
 - Валидиращи спрямо невалидиращи парсери
 - Парсери, написани на конкретен език (Java, C++, Perl, etc.) без използване на определен API – напр. с използване на регулярни изрази
 - ОО-парсери, използващи Document Object Model (DOM) API
 - Управлявани по събития парсери, използващи Simple API for XML (SAX)
 - Базирани на итератори парсери, използващи Streaming API for XML (StAX)

Невалидиращи парсери

- Скорост и ефикасност

- Валидиращ XML парсер - с обработка на DTD/ XML схема и последваща проверка дали всеки елемент в XML документа следва правилата на тази DTD/ XML схема, изисква значително време и ресурси
- Ако целта е единствено да се намерят маркери (етикети) на елементи и да се извлече информация - използвайте невалидиращи парсери

Разбор (парсване) на XML съдържание

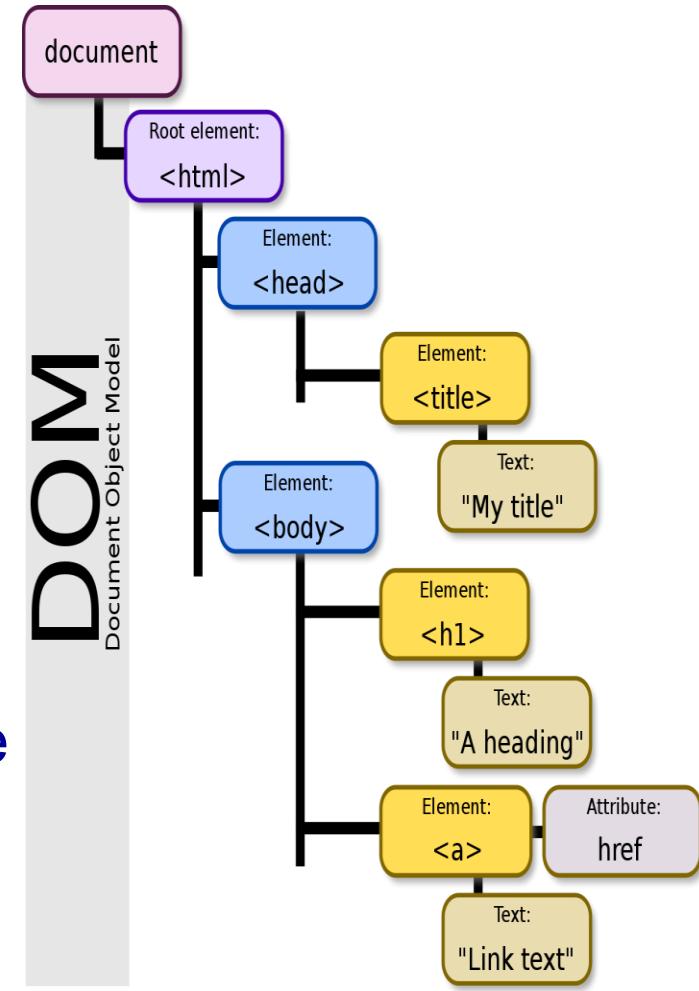
- Три широко-известни API's:
 - DOM (Document Object Model)
 - Дефинира логическо дърво, представяще анализирания XML документ
 - SAX (Simple API for XML)
 - Дефинира манипулатори (handlers), съдържащи методи за разбор на XML документа (*a la push*)
 - Streaming API for XML (StAX)
 - Парсване на XML, базирано на итератори (*a la pull*)
- Приложения без сложна манипулация на XML, но с ограничения по памет, могат да ползват SAX и StAX
- Структурната манипулация на XML елементи изиска използването на DOM

DOM - Document Object Model

- Комплект от приложни интерфейси за четене на XML файл в паметта и съхраняването му като дървовидна структура
- Абстрактните API дават възможност за изграждане, достъп и манипулиране на структурата и съдържанието на XML и HTML документи

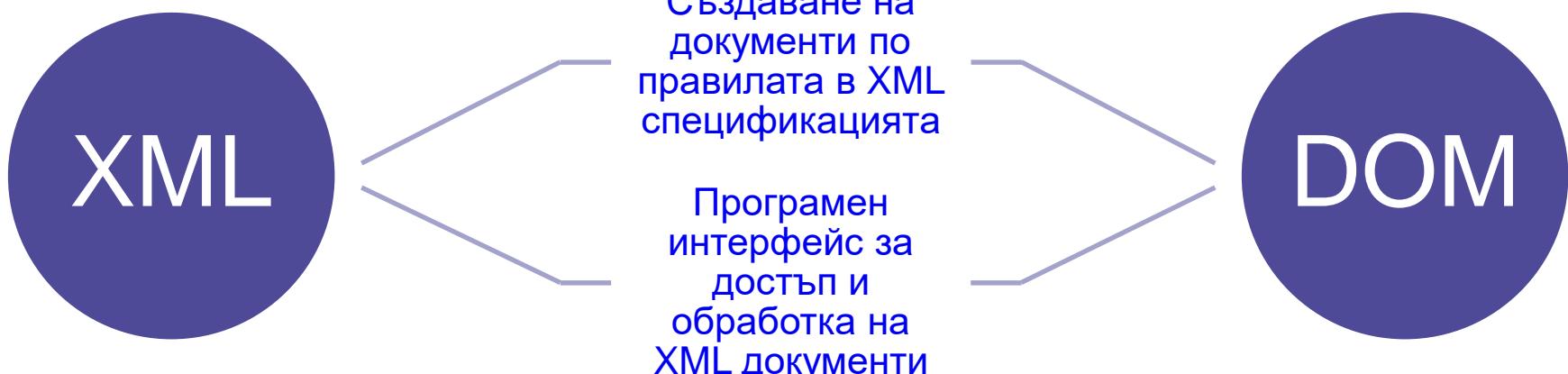
Предимства на DOM 1/2

- Когато правим разбор на XML документ с DOM парсер, получаваме дървовидна структура (наречена **DOM дърво**), която съдържа всички елементи на нашия документ
- DOM предлага разнообразие от функции, които можем да използваме, за да се прегледаме и променяме съдържанието и структурата на документа => удобен за приложни рамки и генератори



Предимства на DOM 2/2

- Стандартизиран начин за достъп до части на XML документ
 - Създаване на документ и части от документи
 - Обхождане на документ
 - Преместване, копиране и премахване на части от документ
 - Добавяне и промяна на атрибути
- Стандартизиран начин за обработка на данни посредством създаване на обектен модел, съответстващ на структурата на XML документа



Как работи DOM?

- DOM обикновено се добавя като междинен слой между XML парсера и приложението
- Изисквания за достъп на приложение до XML документ посредством DOM
 - XML парсер
 - DOM имплементация
 - Съществуват DOM имплементации с вградени XML парсери (например MS XML)

```
graph LR; A[XML документ] --> B[XML парсер]; B --> C[DOM]; C --> D[Приложение]
```

XML
документ

XML парсер

DOM

Приложение

Нива на DOM спецификацията

Level 4

- 2015-11-19

Level 3

- 2004-04-07 (Core)

Level 2

- 2000-11-13 (Core)

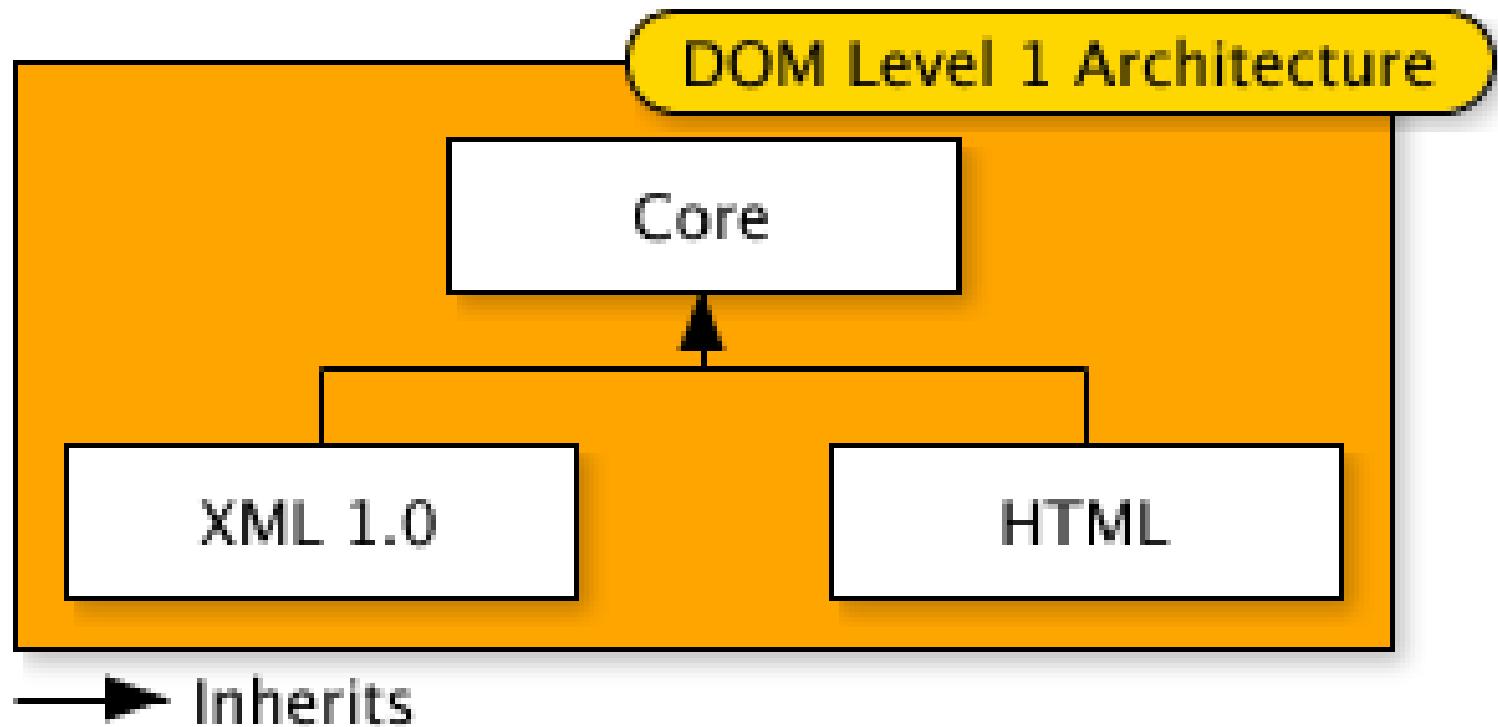
Level 1

- 1998-10-01

DOM нива

- **DOM Level 1, 2 и 3** са най-използваните спецификации на W3C.
- Спецификацията **DOM Level 2/3/4** е обширен набор от интерфейси, стандартизиращи начина на третиране на парсваните XML данни.
- Моделът на DOM е йерархичен, или дървовиден. Документът може да се обработва отгоре-надолу или обратно, с достъп до всеки междинен възел или листо.
- Спецификацията **DOM Level 2/3/4** е разделена на отделни (под)спецификации

DOM Level 1



DOM Level 2

XML DOM Level 2 под-спецификации са:

DOM Level 2 Core (<http://www.w3.org/TR/DOM-Level-2-Core/>)

DOM Level 2 Views

DOM Level 2 Style

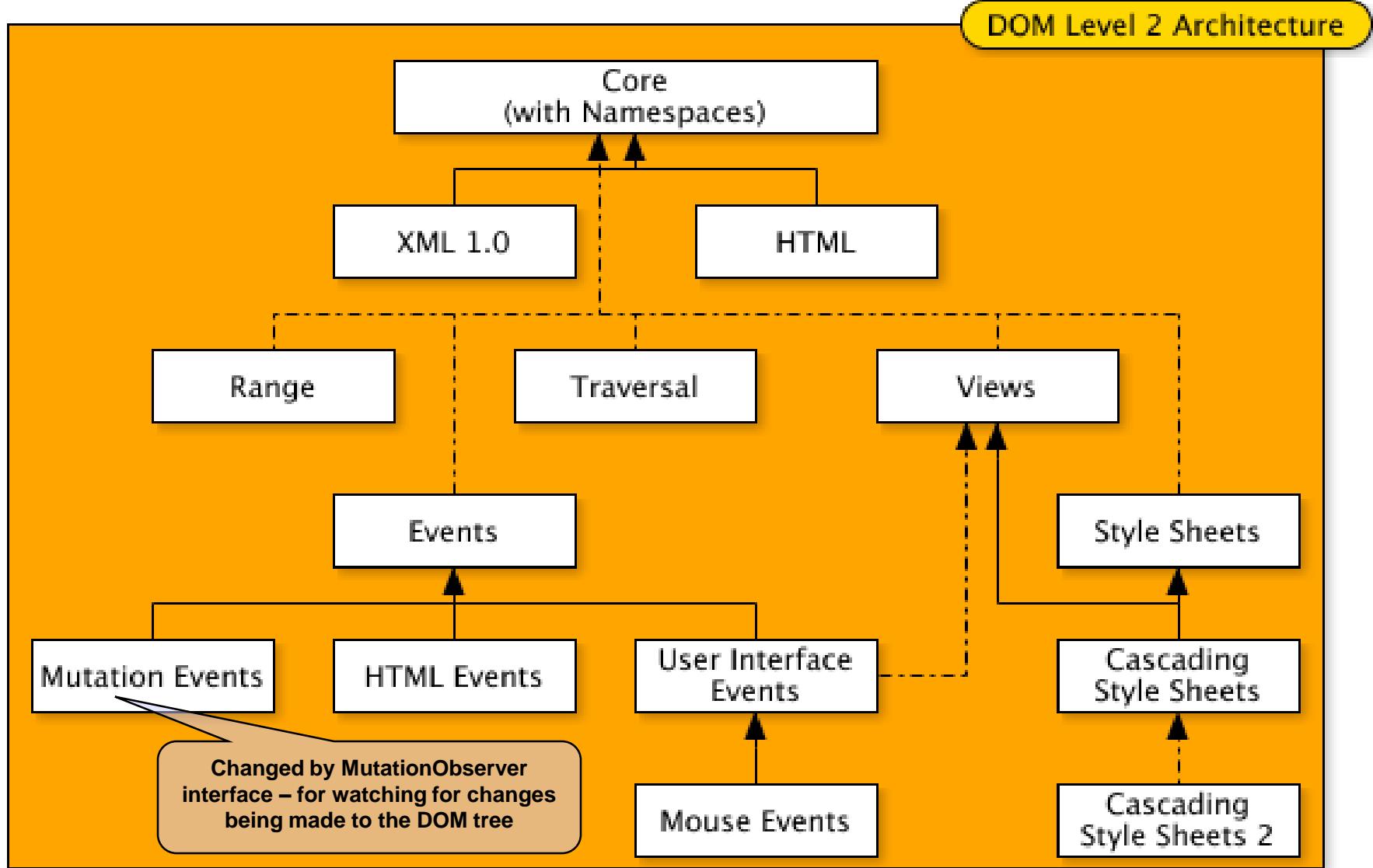
DOM Level 2 Events

DOM Level 2 Traversal-Range

DOM Level 2 HTML (<http://www.w3.org/TR/DOM-Level-2-HTML/>)

- Повечето приложения, които поддържат DOM Level 2, всъщност поддържат само DOM Level 2 Core спецификацията, т.е. достъп и манипулация на HTML или XML (т.нар. basic parsing)
- DOM Level 2 Core представя документа като йерархия от Node обекти и е платформено и езиково независим интерфейс за динамичен достъп и промяна на съдържанието на документа
- Официален W3C DOM Уеб сайт: <http://www.w3.org/DOM/>

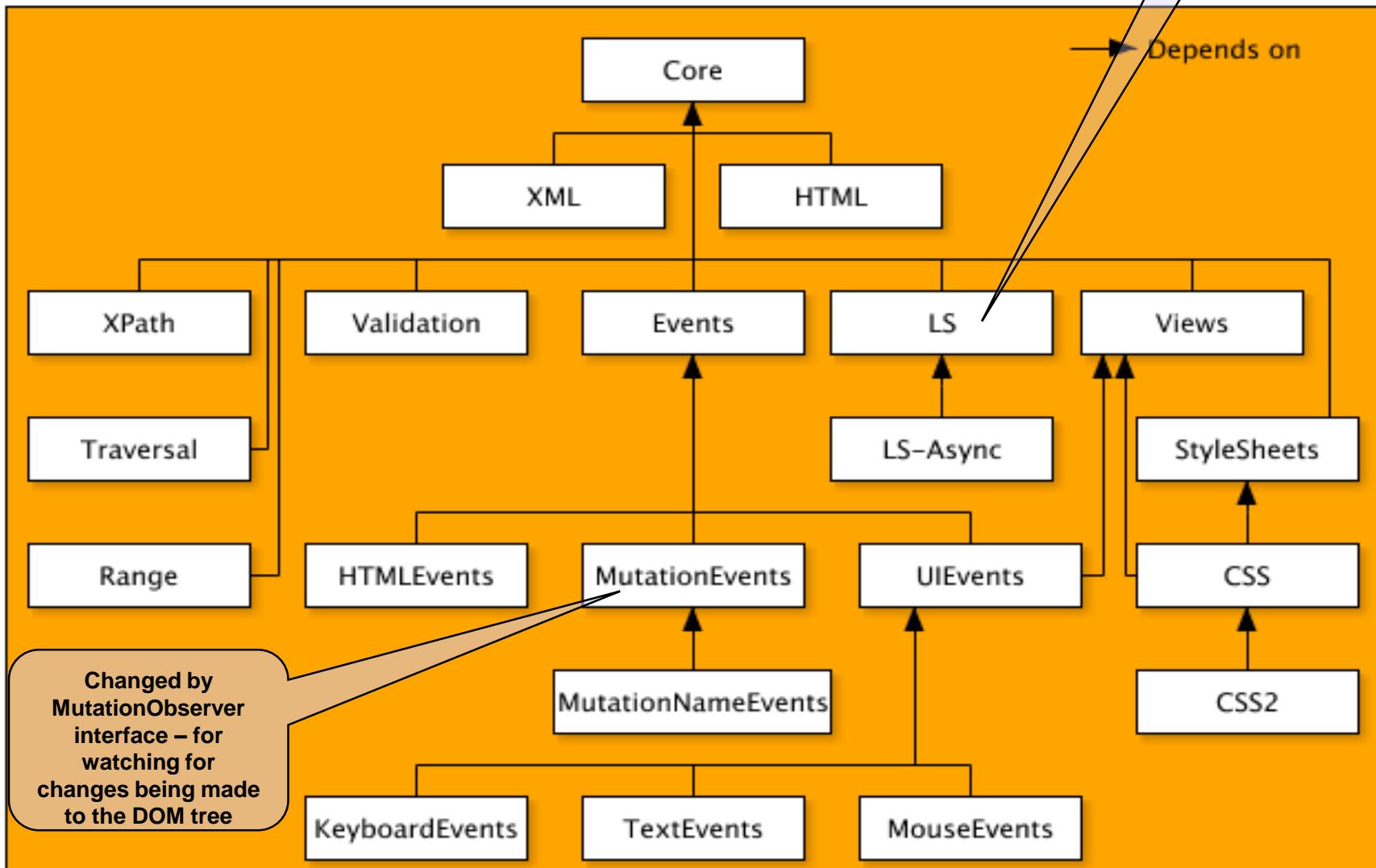
DOM Level 2



→ Inherits - - - → Depends on

DOM Level 3

Load
and
Save



DOM имплементации

● **Xerces**

- Част от Apache проекта, осигурява парсери за Java и C++, имплементира W3C XML и DOM Level 1, 2 (и 3 от Xerces-J 2.7.0 насам) стандартите (<http://xml.apache.org>)

● **4DOM**

- Осигурява на Python разработчиците инструмент за манипулиране на HTML и XML документи

● **ActiveDOM**

- Active-X контрол, който осигурява зареждане и създаване на XML файлове въз основа на W3C DOM 1.0 спецификацията

● **Docuverse DOM**

- Цялостна имплементация на W3C DOM (Document Object Model) API в Java

● **PullDOM и MiniDOM**

- Приложен програмен интерфейс за работа с DOM обекти в Python

● **TclDOM**

- Език, който свързва DOM със скриптовия език Tcl (Tool Command Language)

● **XDBM**

- XML Database Manager, осигурен като вградена база от данни за използване в рамките на други приложения посредством DOM базиран приложен програмен интерфейс

DOM стандарт 1/2

- DOM е официален стандарт на www.w3.org
- Използва ОО подход
- Композиран е от различни интерфейси
 - `org.w3c.dom.*`
- Централен клас е '`Document`' (DOM дърво)
- `DOMString`
 - Спецификация на тип данни, осигуряваща еднакво функциониране на всички DOM имплементации
 - 16-битов низ от символи
 - UTF-16 кодиране

DOM стандарт 2/2

- DOM ядро
 - Множество от интерфейси за работа с базови документи
- DOM optionalни модули
 - Допълнителни интерфейси за работа с други документи като HTML и CSS, които могат да се имплементират при необходимост
- DOM стандартът:
 - включва обхождане на дървото (от DOM Level 2 насам)
 - включва генериране на XML формат (от DOM Level 3 насам) - запазване на DOM дървото като XML документ

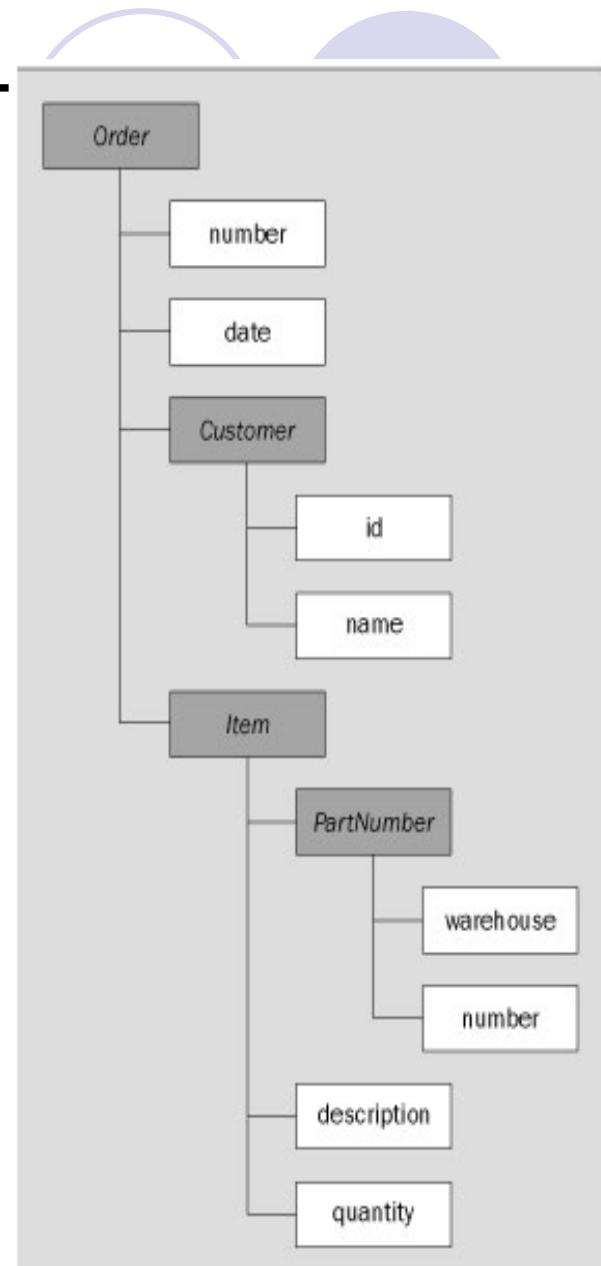
DOM опционални модули

- DOM Views
 - Осигурява на програмите и скриптовете динамичен достъп и обновяване на документния изглед
- DOM Events
 - Осигурява система за управление на събития
- DOM HTML
 - Осигурява на програмите и скриптовете динамичен достъп и обновяване на съдържанието и структурата на HTML документи
- DOM CSS
 - Осигурява на програмите и скриптовете динамичен достъп и обновяване на съдържанието и структурата на CSS документи
- DOM обхождане и обхват
 - Осигурява на програмите и скриптовете динамично обхождане и идентифициране на области в документ

XML документ като обект

```
<?xml version="1.0"?>
<order number="312597">
  <date>2018/11/2</date>
  <customer id="216A">
    Company A
  </customer>
  <item>
    <part-number warehouse="Warehouse 11">
      E16-25A
    </part-number>
    <description>
      Production-Class Widget
    </description>
    <quantity>
      16
    </quantity>
  </item>
</order>
```

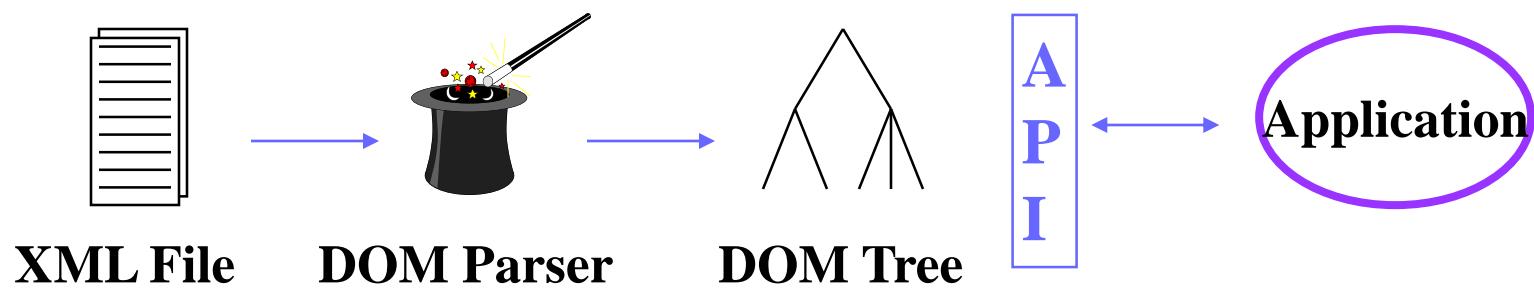
DOM



22

Създаване на DOM дърво

- Всяка DOM имплементация има метод за предаване на XML файл към метод-фабрика, който връща обект-екземпляр на Document – представя елемента корен на целия документ.
- След това използваме стандартния DOM интерфейс за интеракции с XML структурата



Създаване на DOM дърво - пример

```
import org.w3c.dom.*;           //DOM interfaces
import com.sun.xml.tree.*;       //Using Sun classes
import org.xml.sax.*;           //Need SAX classes
```

```
public class myClass {
...
Document myDoc; //Document object
try {
//if 'true' -> validate the XML!
myDoc =
 XmlDocument.createXmlDocument("file:/doc.xml", true);
} catch (IOException err) {...}
    catch (SAXException err) {...}
    catch (DOMException err) {...}

//If no exceptions, should have a 'Document' object
```

DOM

24

DOM интерфейси

- Възли
 - Вътрешно представяне на обектите в дървовидна структура, съхранена в паметта
- Интерфейси
 - Осигуряват механизъм за управление на обектите

```
<parent>
  <child id="123">
    text goes here
  </child>
</parent>
```

Document Node (Document Root)

NodeList

Element Node (<parent>)

NodeList

Element Node (<child>)

NamedNodeMap

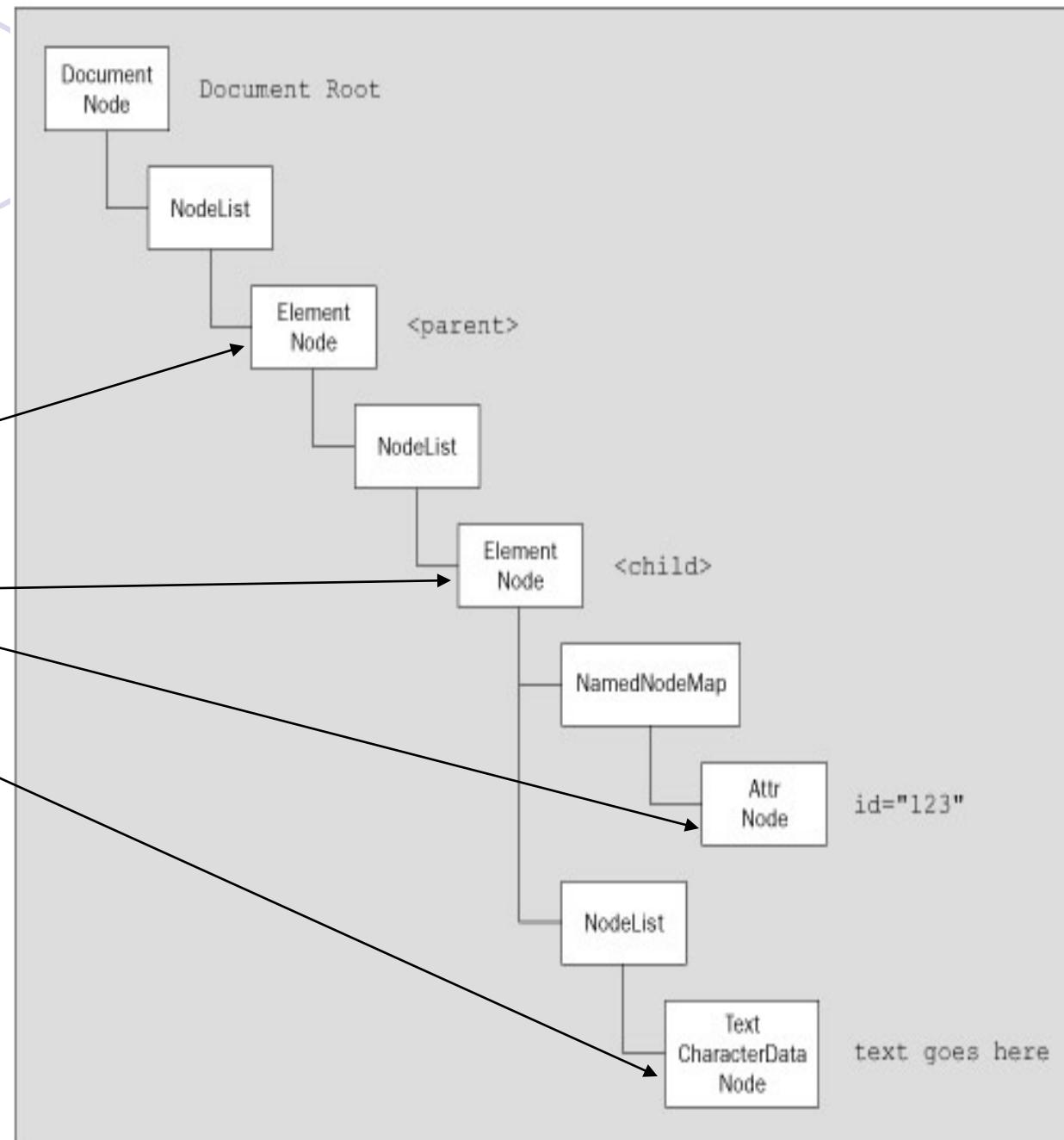
NodeList

Attr Node (id="123")

Text CharacterData
Node (text goes here)

DOM дърво на XML документ

```
<parent>
  <child id="123">
    text goes here
  </child>
</parent>
```



Структурен модел 1/2

- Елементи в XML документа
 - Елемент <parent>
 - Елемент <child>
 - Атрибут id на елемент <child>
 - Съдържание на елемент <child>
- Възел Document в DOM
 - Дефинира контекст на документа
 - Имплементира методи за създаване на обекти в документа
 - Поддържа интерфейс Document
 - Притежава само един дъщерен елемент
 - Може да притежава други XML елементи като коментари, инструкции за обработка, декларации на типове

Структурен модел 2/2

- Възел NodeList
 - Дефинира списък от възли Nodes на едно ниво в дървото
 - Поддържа интерфейс NodeList
 - Добавя се автоматично преди елемент
- Възел NamedNodeMap
 - Дефинира неподредено множество от възли, реферирали по име на атрибут
 - Добавя се автоматично преди атрибути

DOM имплементации

- Съществуват различни имплементации на DOM
- Поради това DOM предоставя **DOM Core** – множество от основни интерфейси
- Допълнително, DOM предоставя множества от интерфейси за други формати – **DOM HTML**, **DOM CSS**, и др. Напр. **DHTML** използва обекти от *DOM HTML* множеството.
- Всяка имплементация предоставя класове, имплементиращи интерфейсите

DOM + HTML = DHTML

- Комбинацията от DOM (с HTML API), скриптови езици и HTML изгражда т.нар. **Dynamic HTML** (или **DHTML**), където съдържанието на HTML документа се представя чрез обектен модел. *Когато Уеб страница се зареди в DHTML браузър, обекти се създават за всеки елемент от страницата. Това прави възможно изпълнението на скриптове в страницата, обръщащи се към методи и свойства (properties) на тези обекти.*
- Например, за HTML формата:
 - <form id="frmScratchForm" name="frmScratchForm">
 - <input name="rdoRadio" type="radio" checked>First

 - <input name="rdoRadio" type="radio">Second
 - </form>
- , JavaScript код може да избере втория радио бутон (индексът започва от **0**) така:
 - **document.frmScratchForm.rdoRadio[1].checked = true;**
- Така след зареждане на страницата стойностите на радио-бутоните ^{XML}_{DOM} могат да се променят програмно и *динамично*.

Примерен тест с ползване на MSXML

```
<html>
```

```
  <head><title>DOM Demo</title>
```

```
  <script language="JavaScript">
```

```
    var oDOM;
```

```
    oDOM = new
```

```
      ActiveXObject("MSXML.DOMDocument");
```

```
    oDOM.async = true;
```

```
    oDOM.load("domnode.xml");
```

```
    //our code will go here...
```

```
  </script>
```

```
  </head>
```

```
  <body>
```

```
    <p>This page demos some of the DOM capabilities.</p>
```

```
  XML</body>
```

DOM

```
</html>
```

Документът ще се зареди асинхронно, т.е. методът `load()` ще върне управлението веднага, а документът ще продължи да се зарежда във фонов режим.

Основни типове интерфейси

DOM ядро

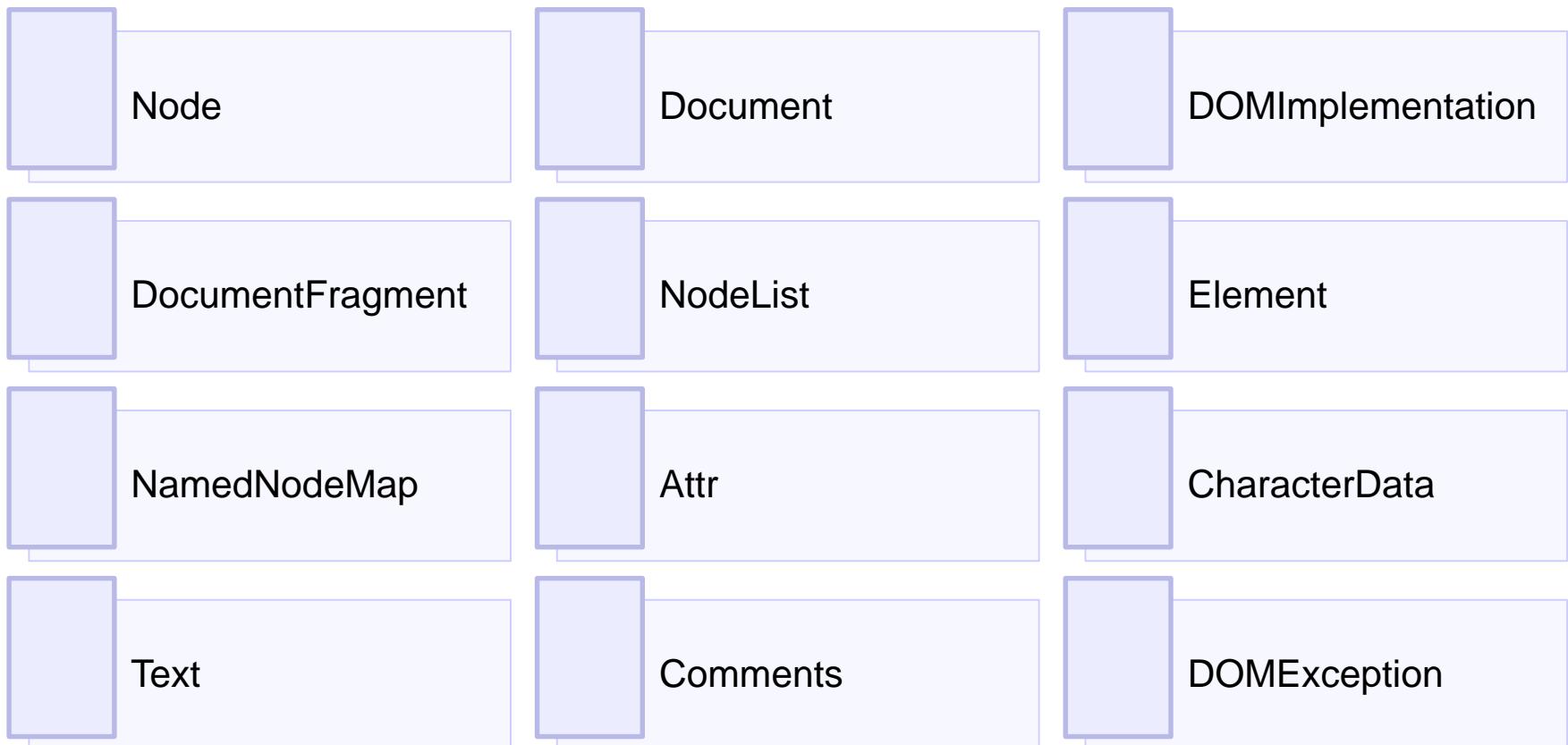
Базови интерфейси

Задължителни за всички
имплементации на DOM

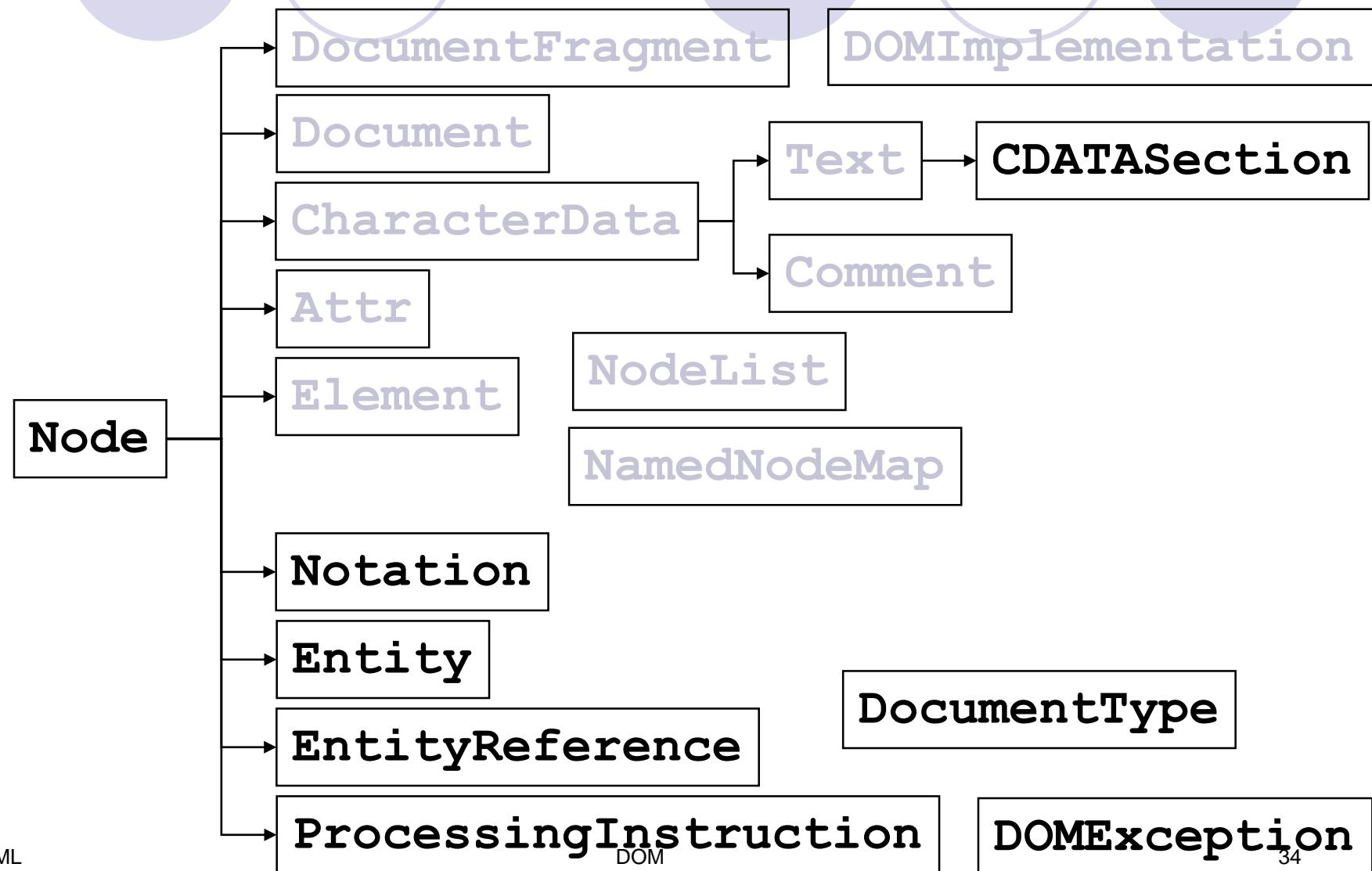
Разширени интерфейси

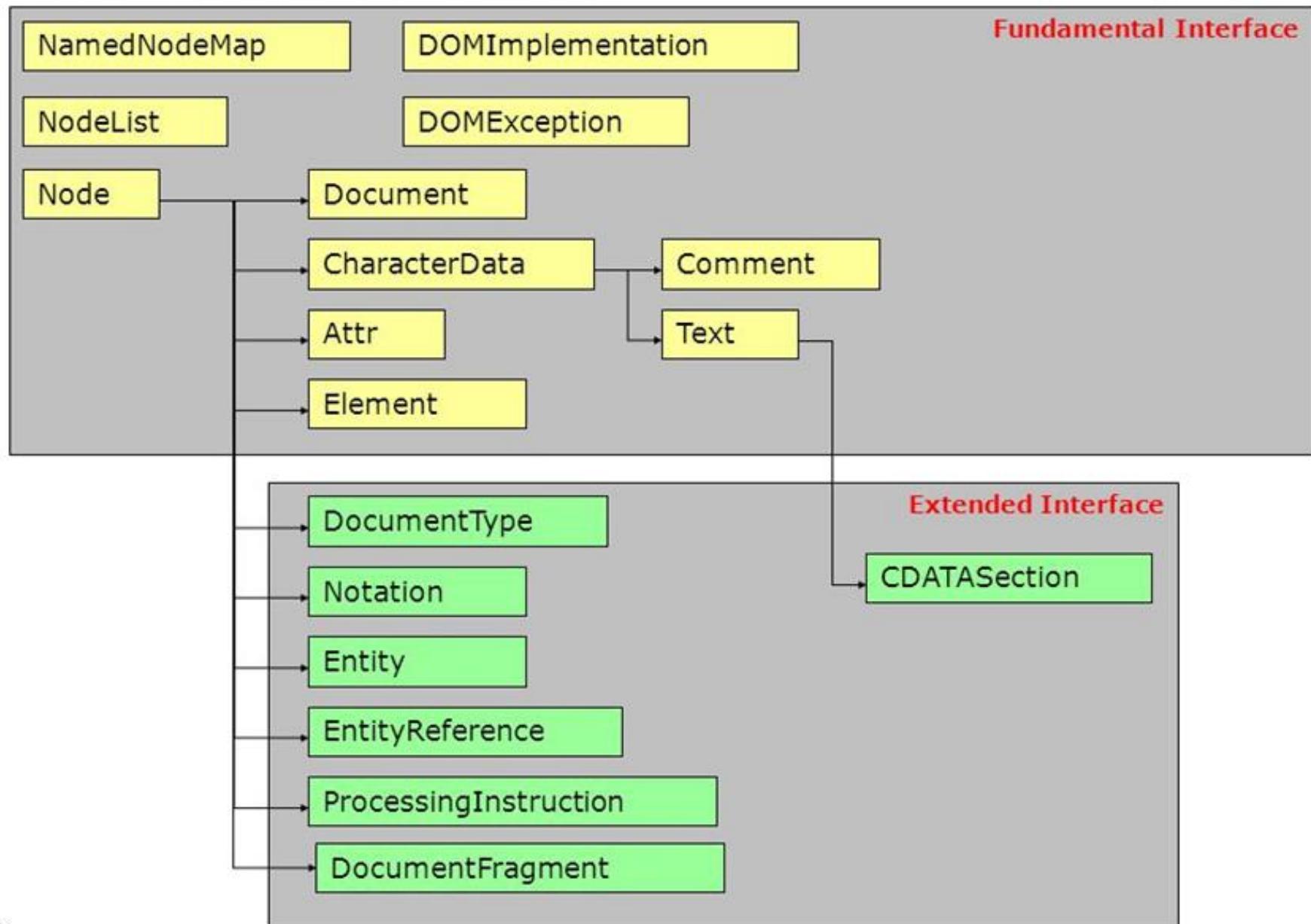
Необходими са само за DOM
имплементации, работещи с XML

Базови (фундаментални) DOM интерфейси



Разширени DOM интерфейси





DOM интерфейси

- DOM дефинира няколко главни интерфейса:

Node	Базов тип данни на DOM
Element	Представя елемент
Attr	Представя атрибут на елемент
Text	Представя съдържанието на атрибут или елемент
Document XML	Представя целия XML документ; Document обектът често се означава като DOM дърво.

Интерфейс NODE

- Всяка част от XML документа се представя с интерфейс NODE
- Функционалност
 - Обхождане на дървото: всяка обработка изиска позициониране на определено място в дървото
 - Получаване на информация за възел: тип, атрибути, име и стойност
 - Добавяне, премахване и обновяване на възли
 - Промяна на структурата на дървото

Свойства на интерфейс Node

nodeName

nodeValue

nodeType

parentNode

childNode

firstChild

lastChild

previousSibling

nextSibling

attributes

ownerDocument

namespace URI

prefix

localName

Методи на интерфейс Node

- insertBefore(newChild, refChild)
- replaceChild(newChild, oldChild)
- removeChild(oldChild)
- appendChild(newChild)
- hasChildNodes()
- cloneNode(deep)
- normalize()
- supports(feature,version)

Методи

Методи на Node 1/2

- Три категории от методи:
 - Характеристики на Node
 - name, type, value
 - Контекстна локация и достъп до съседни възли:
 - parents, siblings, children, ancestors, descendants
 - Модификации на Node:
 - Edit, delete, re-arrange child nodes

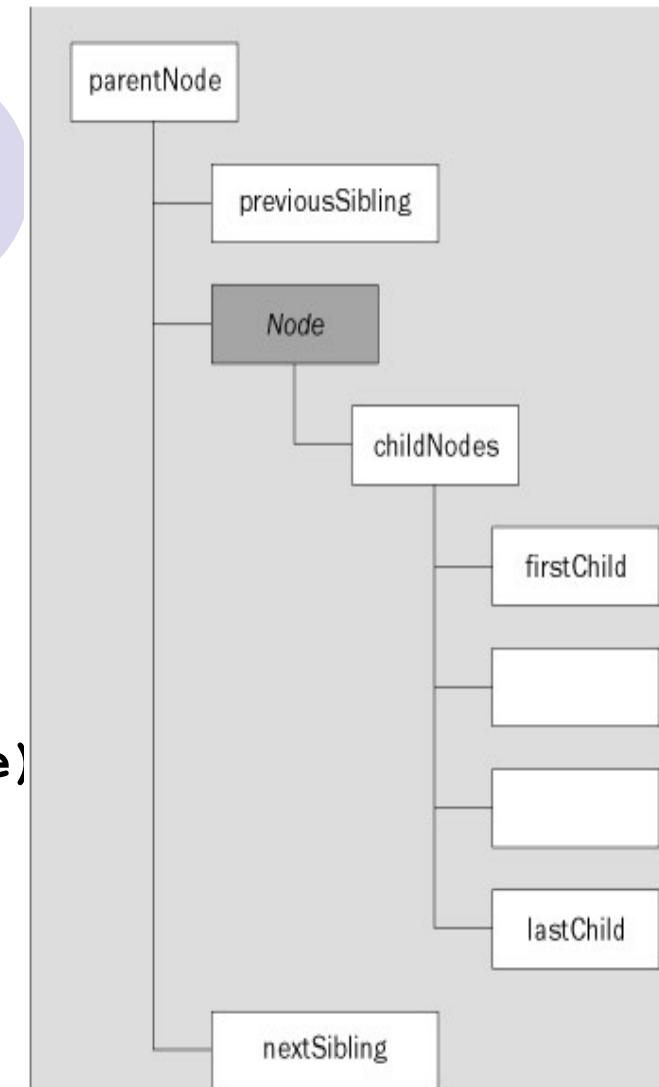
Методи на Node 2/2

```
short      getNodeType();  
  
String     getNodeName();  
  
String     getNodeValue()  
throws DOMException;  
  
void       setNodeValue(String value)  
throws DOMException;  
  
boolean    hasChildNodes();  
  
NamedNodeMap getAttributes();  
//returns unordered collection of nodes  
  
Document  getOwnerDocument();
```

DOM

Прилага се само, ако възелът е елемент с атрибути
Връща като резултат NamedNodeMap
колекция с атрибутите на възела или NULL

XML



Типове възли - getNodeTipe()

ELEMENT_NODE	= 1	PROCESSING_INSTRUCTION_NODE	= 7
ATTRIBUTE_NODE	= 2	COMMENT_NODE	= 8
TEXT_NODE	= 3	DOCUMENT_NODE	= 9
CDATA_SECTION_NODE	= 4	DOCUMENT_TYPE_NODE	= 10
ENTITY_REFERENCE_NODE	= 5	DOCUMENT_FRAGMENT_NODE	= 11
ENTITY_NODE	= 6	NOTATION_NODE	= 12

Пример:

```
if (myNode.getNodeType() == Node.ELEMENT_NODE) {  
    //myNode.getNodeType() returns short!!  
    //process node  
    ...
```

Node имена и стойности

- Всеки възел има име ... и евентуално - стойност
- Името не е уникален идентификатор (а само локация)

Тип	Име на интерфейс	Име	Стойност
ATTRIBUTE_NODE	Attr	<i>Attribute name</i>	<i>Attribute value</i>
DOCUMENT_NODE	Document	#document	NULL
DOCUMENT_FRAGMENT_NODE	DocumentFragment	#document-fragment	NULL
DOCUMENT_TYPE_NODE	DocumentType	<i>DOCTYPE name</i>	NULL
CDATA_SECTION_NODE	CDataSection	#cdata-section	<i>CDATA content</i>
COMMENT_NODE	Comment	<i>Entity name</i>	<i>Content string</i>
ELEMENT_NODE	Element	<i>Tag name</i>	NULL
ENTITY_NODE	Entity	<i>Entity name</i>	NULL
ENTITY_REFERENCE_NODE	EntityReference	<i>Entity name</i>	NULL
NOTATION_NODE	Notation	<i>Notation name</i>	NULL
PROCESSING_INSTRUCTION_NODE	ProcessingInstruction	<i>Target string</i>	<i>Content string</i>
TEXT_NODE	Text	#text	<i>Text string</i>

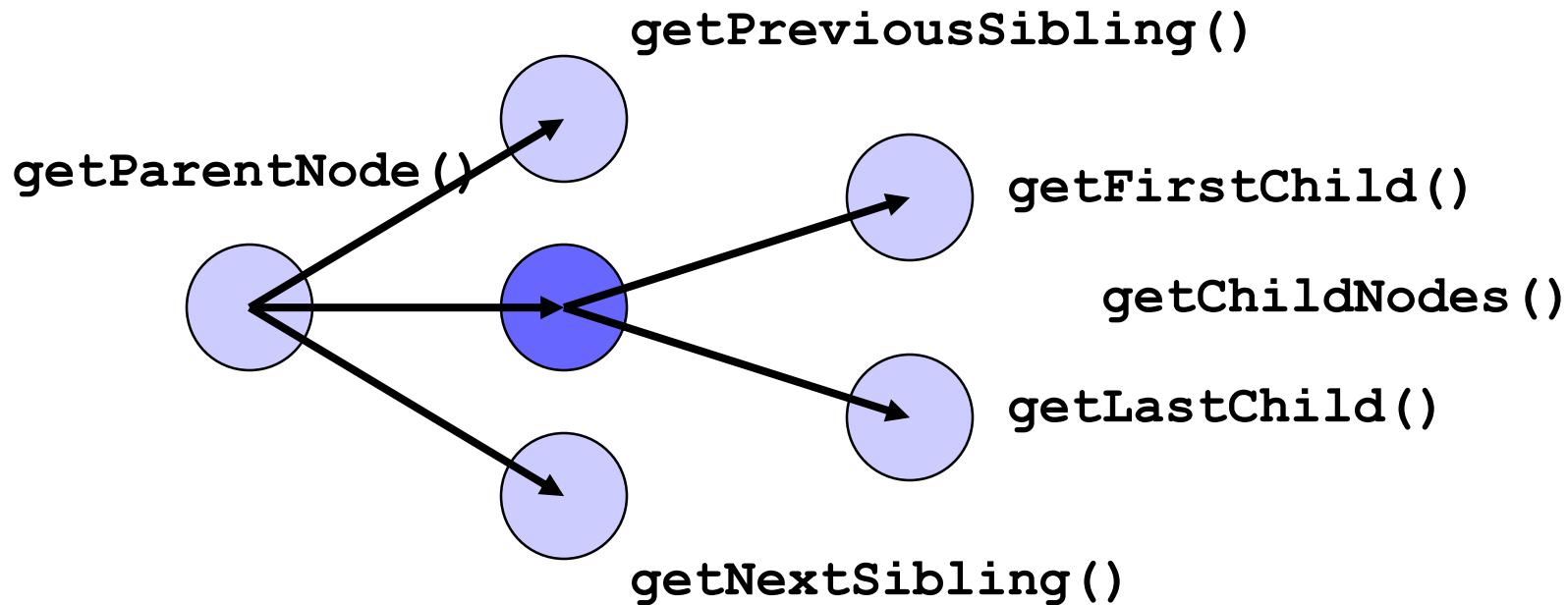
Възли-деца (Child Nodes)

- Повечето **Nodes** не могат да имат деца, с изключение на:
 - **Document**, **DocumentFragment**, **Element**
- Проверка за съществуване на деца:
 - `if (myNode.childNodes.length) {
 //process children of myNode
 ...
}`

Node навигация 1/2

- Всеки възел има специфично местоположение в дървото
- Node интерфейсът има методи за намиране на възлите от обкръжението:
 - `Node getChildNodes()`
 - `Node getFirstChild()`
 - `Node getLastChild()`
 - `Node getNextSibling()`
 - `Node getPreviousSibling()`
 - `Node getParentNode()`
 - `NodeList getChildNodes()`

Node навигация 2/2



```
Node parent = myNode.getParentNode();  
if (myNode.hasChildren()) {  
    NodeList children = myNode.getChildNodes();  
}
```

Пример

- For domnode.xml:

```
<root>
  <DemoElement
    DemoAttribute="stuff">
    This is the PCDATA
  </DemoElement>
</root>
```

```
<script language="JavaScript">
```

```
  var oDOM;
  oDOM = new
    ActiveXObject("MSXML.DOMDocument");
  oDOM.async = false;
  oDOM.load("domnode.xml");
  //our code will go here...
  alert(oDOM.firstChild.firstChild.firstChild.nodeValue);
```

XML

```
</script>
```

DOM



47

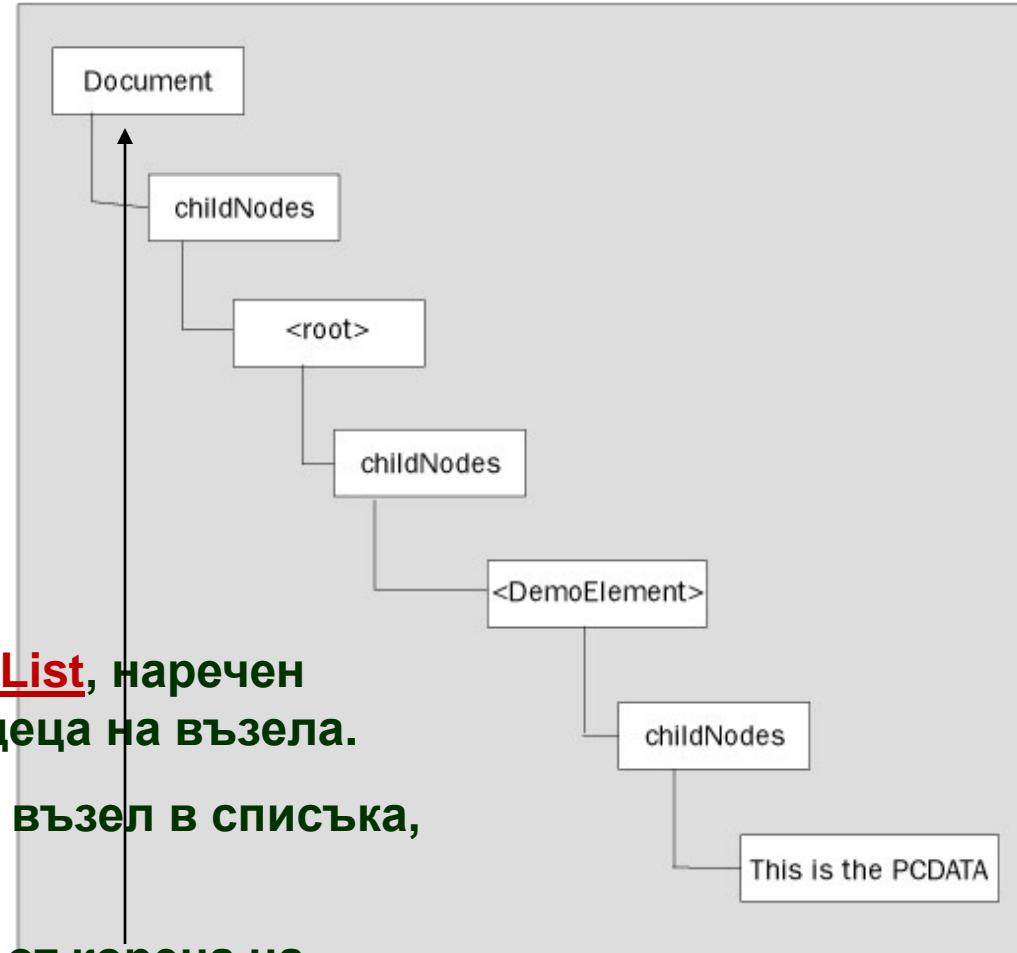
DOM дърво

`alert(oDOM.firstChild.firstChild.firstChild.firstChild.nodeValue)`

- За domnode.xml:

```
<root>
<DemoElement
  DemoAttribute="stuff">
  This is the PCDATA
</DemoElement>
</root>
```

- Всеки Node обект предоставя NodeList, наречен childNodes, който съдържа всички деца на възела.
- Имаме директен достъп до първия възел в списъка, посредством firstChild property.
- В предходния код се придвижваме от корена на документа към елемента `<DemoElement>` посредством firstChild property.



DOM

48

`oDOM.firstChild.firstChild.firstChild.firstChild.nodeValue`

Node манипулации 1/2

- Децата на възел в DOM дърво могат да бъдат манипулирани – добавяни, редактирани, изтривани, копирани, премествани и т.н.

Node **removeChild(Node old)**

throws DOMException;

Node **insertBefore(Node new, Node ref)** throws DOMException;

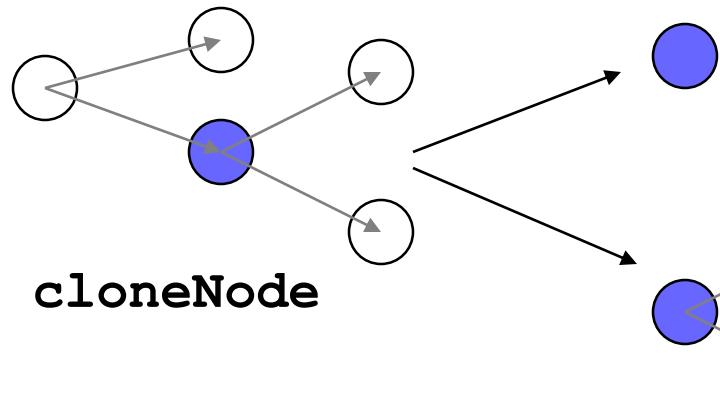
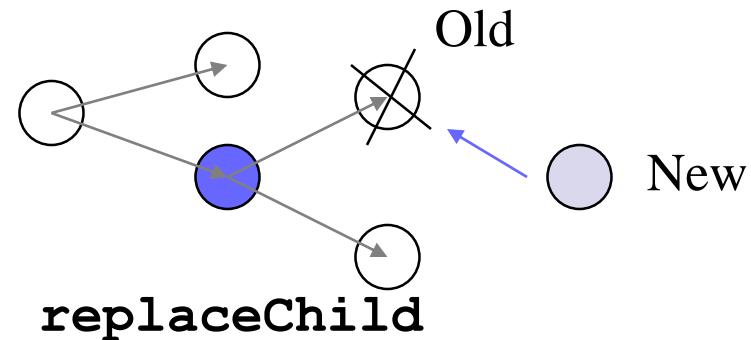
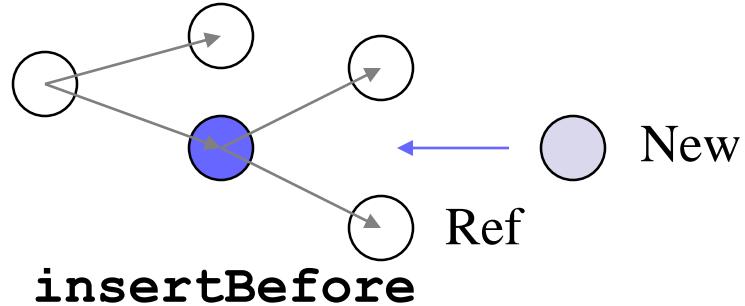
Node **appendChild(Node new)**

throws DOMException;

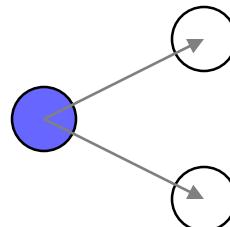
Node **replaceChild(Node new, Node old)** throws DOMException;

Node **cloneNode(boolean deep);**

Node манипулации 2/2



Плитко (Shallow) клониране –
при **deep='false'**



Дълбоко (Deep) клониране –
при **deep='true'**

Добавяне и премахване на възли

- Методът `appendChild`

```
objParentNode.appendChild(objChildNode);
```

- Методът `insertBefore`

- Ако не се укаже възел, обозначаващ място на вмъкване, то новият наследник се добавя в края на списъка
 - Ако възелът, който се добавя, вече съществува, то се извършва подмяна

```
objParentNode.insertBefore(objChildNode,  
                           objParentNode.lastChild);
```

- Методът `removeChild`

- Изтритият възел се връща като резултат в случай, че е необходима последваща обработка

```
objOldChild =  
    objParent.removeChild(objParent.lastChild);
```

Замяна и клониране на възли

Методът replaceChild

```
objOldChild = objParent.replaceChild( objNewChild,  
                                      objParent.firstChild);
```

Методът cloneNode

- Клонираните възли могат да се използват само в документа, в който са създадени

```
<name id="1">John</name>
```

```
objNewNode = objNode.cloneNode(false);  
//objNewNode is now <name id="1"/>
```

false - Default. Clone
only the node and its
attributes

```
objNewNode = objNode.cloneNode(true);
```

```
//objNewNode is now <name id="1">John</name>
```

true - Clone the node,
its attributes, *and* its
descendants

Document::Node интерфейс 1/2

- Представя целия XML документ (корен на дървото)
- Наследява интерфейса Node, като добавя функционалност за обхождане на DOM дървото
- Възелът Document обхваща всички възли в DOM дървото, включително и кореновия елемент
- Методи:

```
//Information from DOCTYPE - See 'DocumentType'  
DocumentType      getDocumentType();  
//Information about capabilities of DOM implementation  
DOMImplementation getImplementation();  
//Returns reference to root node element  
Element           getDocumentElement();  
//Searches for all occurrences of 'tagName' in nodes  
NodeList          getElementsByTagName(String tagName);  
//Търси елемент по атрибут ID и връща NodeList  
NodeList          getElementById(String ID);
```

Document::Node интерфейс 2/2

- Метод-фабрики за създаване на възли:

Element **createElement**(String tagName) throws DOMException;

DocumentFragment **createDocumentFragment**();

Text **createTextNode**(String data);

Comment **createComment**(String data);

CDATASection **createCDATASection**(String data) throws
DOMException;

ProcessingInstruction **createProcessingInstruction**(
String target, String data) throws DOMException;

Attr **createAttribute**(String name) throws DOMException;

EntityReference **createEntityReference**(String name)
throws DOMException; ⁵⁴

Създаване на XML документи

- Създаване на възел

```
var objNode, objText;  
objNode = objDOM.createElement("root");  
objText = objDOM.createTextNode("root PCDATA");  
objDOM.appendChild(objNode);  
objNode.appendChild(objText);
```

- Създаване на атрибут

```
var objAttr;  
objAttr = objDOM.createAttribute("id");  
//set the attribute's value  
objAttr.nodeValue = "123";  
//append the attribute to the element  
objNode.attributes.setNamedItem(objAttr);  
alert(objDOM.xml);
```

<root id="123">root PCDATA</root>

Element::Node интерфейс 1/2

- Две категории методи:

- Общи методи за елемента:

```
String    getTagName () ;  
NodeList  getElementsByTagName () ;  
Void      normalize () ;  
// removes empty Text nodes, and joins adjacent Text nodes.
```

- Методи за управление на атрибути:

```
String   getAttribute (String name) ;  
void     setAttribute (String name, String value)  
           throws DOMException ;  
void     removeAttribute (String name)  
           throws DOMException ;  
  
Attr     Attr  
void     void  
XML      void  
void     removeAttributeNode (Attr old)  
           throws DOMException ;  
dom      throws DOMException ;
```

разлика

XML

56

Element::Node интерфейс 2/2

- Само **Element** обектите могат да имат атрибути, като атрибутните методи на **Element** са прости:
 - Изискват име на атрибута
 - Не различават стойност по подразбиране (определена в DTD) от зададена в XML файла
 - Не могат да определят типа на атрибута [String]
- Вместо тях използвайте метода **getAttributes()** на **Node**
 - Връща **Attr** обекти като **NamedNodeMap**

Интерфейс element: Пример 1/2

```
<?xml version="1.0"?>
<root first='John' last='Doe' />
var objDOM;
objDOM = new ActiveXObject("MSXML2.DOMDocument");
objDOM.async = false;
objDOM.load("ch06_ex8.xml");
var objElement;
objElement = objDOM.documentElement;
...
alert(objElement.getAttribute("first"))
objElement.setAttribute("first", "Bill");
var objAttr;
objAttr = objElement.getAttributeNode("first");
objAttr.nodeValue = "Bill";
objElement.getAttributeNode("first").nodeValue = "Bill";
```

returns the Element
that is the root
element of the doc.

Интерфейс element: Пример 2/2

```
var objElement;  
objElement = objDOM.documentElement;  
...  
var objAttr;  
objAttr = objDOM.createAttribute("middle");  
objAttr.nodeValue = "Fitzgerald Johansen";  
objElement.setAttributeNode(objAttr);  
alert(objDOM.xml);  
objElement.setAttribute("middle", "Fitzgerald Johansen");  
  
<?xml version="1.0"?>  
<root first='John' last='Doe' />  
  
<?xml version="1.0"?>  
<root first='John'  
last='Doe'  
middle='Fitzgerald Johansen' />
```



Attr::Node интерфейс 1/2

- Интерфейс към обекти, съдържащи атрибутни данни
- Свойство ownerElement //deprecated in DOM 4
 - Връща като стойност елемент, на който атрибутът принадлежи
- Свойства name и value
 - Имат стойности аналогични на тези за nodeName and nodeValue
- Свойство specified
 - Определя дали атрибутът е физически специфициран или е скрит със стойност по подразбиране

```
//Get name of attribute
String getName();
//Get value of attribute
String getValue();
//Change value of attribute
void setValue(String value);
//if 'true' - attribute defined in element, else in DTD
boolean getSpecified();
```

Attr::Node интерфейс 2/2

- Атрибутите не са пряка част от дърводидната структура на документа
 - `parentNode`, `previousSibling` and `nextSibling` връщат `null` за `Attr` обект
- Създаваме атрибутни обекти чрез метод-фабрика на `Document`

```
//Create the empty Attribute node  
Attr newAttr = myDoc.createAttribute("status");
```

```
//Set the value of the attribute  
newAttr.setValue("secret");
```

```
//Attach the attribute to an element  
myElement.setAttributeNode(newAttr);
```

DOM

Интерфейсите CharacterData и Text

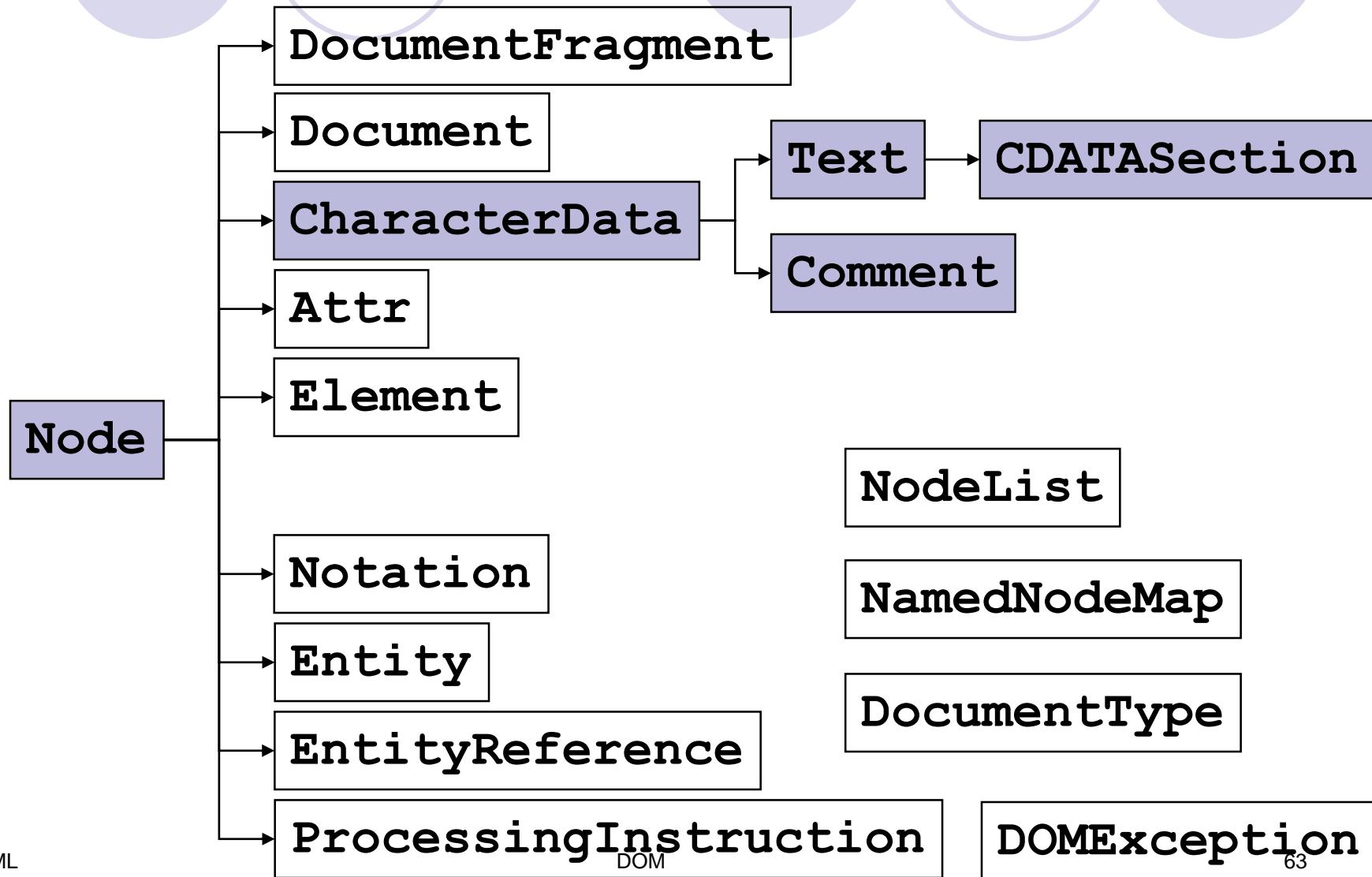
CharacterData

- Осигурява свойства и методи за работа с текст

Text

- Разширява CharacterData и се прилага специфично за PCDATA

Разширени DOM интерфейси



CharacterData::Node интерфейс

- Полезни общи методи за текстообработка
- Не се използват директно...
 - ... а в наследяващите го типове на възли Text и Comment

```
String getData() throws DOMException;
void setData(String data) throws DOMException;
int getLength();
void appendData(String data) throws DOMException;
String substringData(int offset, int length)
throws DOMException;
void insertData(int offset, String data)
throws DOMException;
void deleteData(int offser, int length)
throws DOMException;
void replaceData(int offset, int length, String data)
throws DOMException;
```

Text::Node интерфейс

- Представя текстовото съдържание на **Element** или **Attr**
 - Деца на тези възли
- Винаги са възли-листа (leaf nodes)
- Към CharacterData е добавен един метод:
 - **Text splitText(int offset) throws DOMException**
- Метод-фабрика в **Document** за създаване - **createTextNode()**
- Извикването на **normalize()** върху даден **Element** слива **Text** обектите (те може да са повече от един!)

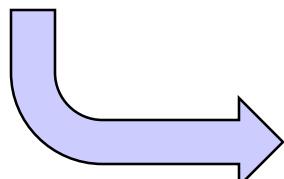
Comment::Text интерфейс

- Представя коментари
- Автоматично форматиране на коментарите с разделите '<!--' и '-->'
- Няма добавен методи към **CharacterData**
- Метод-фабрика в **Document** за създаване
 - `Comment newComment =
myDoc.createComment(" my comment ") ;
//Note spaces`

https://www.w3schools.com/xml/met_document_createcomment.asp

```
function myFunction(xml) {  
    var x, i, newComment, xmlDoc, txt;  
    xmlDoc = xml.responseXML;  
    txt = "";  
    x = xmlDoc.getElementsByTagName("book");  
    for (i = 0; i < x.length; i++) {  
        newComment = xmlDoc.createComment("Revised April 2018");  
        x[i].appendChild(newComment);  
    }  
    for (i = 0; i < x.length; i++) {  
        txt += x[i].getElementsByTagName("title")[0].childNodes[0].nodeValue +  
            " - " +  
            x[i].lastChild.nodeValue + "<br>";  
    }  
    document.getElementById("demo").innerHTML = txt;  
}
```

XML



Everyday Italian - Revised April 2018
Harry Potter - Revised April 2018
XQuery Kick Start - Revised April 2018
Learning XML - Revised April 2018

NodeList интерфейс

- Съдържа колекция от ordered Node обекти
- Два метода:

```
//Find number of Nodes in NodeList  
int getLength();
```

```
//Return the i-th Node  
Node item(int index);
```

```
Node child;  
NodeList children = element.getChildNodes();  
for (int i = 0; i < children.getLength(); i++) {  
    child = children.item(i);  
    if (child.getNodeType() == Node.ELEMENT_NODE) {  
        System.out.println(child.getNodeName());  
    }  
}
```

Интерфейс **NamedNodeMap**

- Дефинира неподредена колекция от възли
- Метод `getNamedItem`
 - Получава като параметър име на възел, например име на атрибут
 - Връща като стойност Node обект
- Метод `removeNamedItem`
 - Получава като параметър име на възел
 - Връща като стойност изтрития Node обект
- Метод `setNamedItem`
 - Получава като параметър възел за добавяне или замяна (ако такъв вече съществува)
- Свойство `length`
 - Връща като стойност броя на елементите в колекцията
- Метод `item`
 - Получава като параметър индекс на възел

NamedNodeMap интерфейс

```
NamedNodeMap myAttributes = myElement.getAttributes();  
NamedNodeMap myEntities = myDocument.getEntities();  
NamedNodeMap myNotations = myDocument.getNotations();  
-----  
int getLength();  
Node item(int index);  
Node getNamedItem(String name);  
Node setNamedItem(Node node) throws DOMException; //Node!  
Node removeNamedItem(String name) throws DOMException;
```

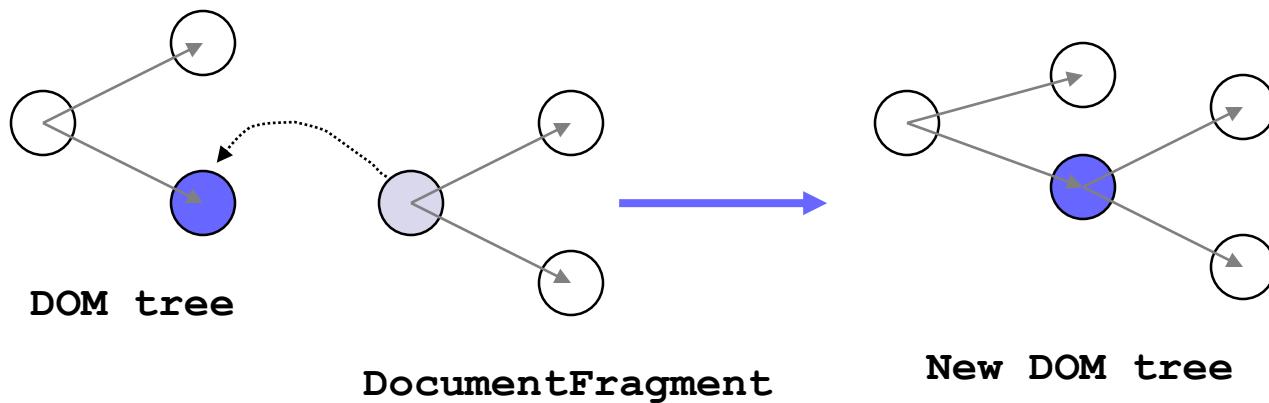
Node object

Required. The node you want to add/replace
in the NamedNodeMap collection

Tip: Instead of working with attribute nodes, you
could use the `element.setAttribute()` method to
add an attribute with a value to an element.

DocumentFragment::Node интерфейс

- Фрагмент от **Document** може да бъде временно съхранен в **DocumentFragment** възел
 - Напр. за 'cut-n-paste'
- Разрушава се, когато се прикачи към друг **Node**



DOMImplementation интерфейс

- За определяне на нивото на поддръжка в DOM парсър
 - `hasFeature(String feature, String version);`
 - `if (theParser.hasFeature("XML", "1.0") {
 //XML is supported
 ...
}`

Интерфейс **DOMException**

- **NOT_FOUND_ERR**
 - Опит да се получи референция към възел, който не съществува в контекста
- **DOMSTRING_SIZE_ERR**
 - Опит да се специфицира част от текст с некоректни граници
- **HIERARCHY_REQUEST_ERR**
 - Опит да се вмъкне възел на неподходящо място в йерархията на DOM дървото
- **INDEX_SIZE_ERR**
 - Опит да се достъпи отрицателен индекс или индекс, който е по-голям от допустимата стойност
- **NOT_SUPPORTED_ERR**
 - Опит да се достъпи обект, чийто тип не се поддържа от имплементация

Разширени интерфейси

CDATASection

- Добавя CDATA секция
- <![CDATA []]>

Notation

- Декларирана в DTD нотация

EntityReference

- Представя възел EntityReference (мнемоничен псевдоним на символ)

DocumentType

- Списък от предефинирани свойства на документа

Entity

- Представя възел Entity (специални символи)

ProcessingInstruction

- Добавя инструкции за обработка
- Свойства target и data

CDATASection::Text интерфейс

- Представя CDATA секция (немаркиран текст) – в него се разпознава като край на CDATA секция само разделителя "]]>"
- Атрибутът `DOMString` на възел `Text` съдържа текста на CDATA секция
- Няма добавени методи към `CharacterData`
- Метод-фабрика в `Document` за създаване
 - `CDATASection newCDATA = myDoc.createDATASEction ("press <<<ENTER>>>") ;`

ProcessingInstruction::Node интерфейс

- Представя декларация за инструкция за обработка
 - Име на възела е *target application name*
 - Стойност на възела е *target application command*

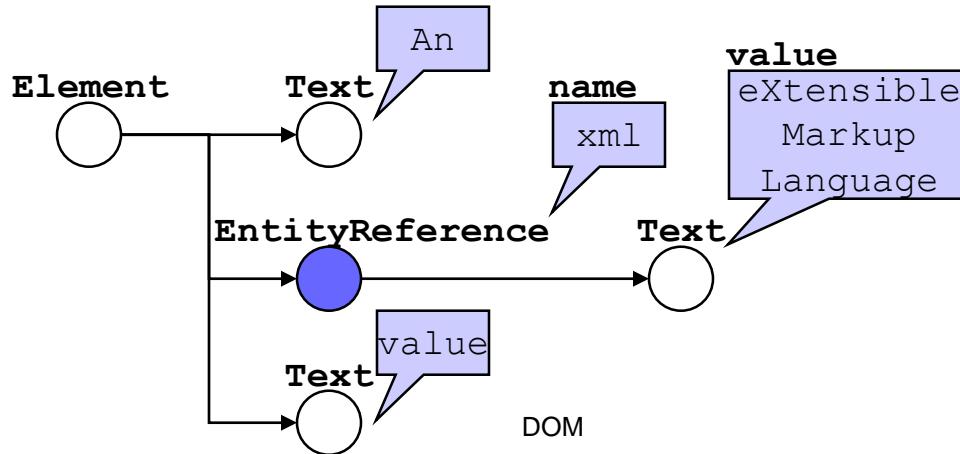
```
//Get the content of the processing instruction
String getData()
//Set the content of the processing instruction
void setData(String data)
//The target of this processing instruction
String getTarget();
```

- Метод-фабрика в **Document** за създаване
 - `ProcessingInstruction newPI =
myDoc.createProcessingInstruction ("ACME" ,
"page-break") ;`

EntityReference::Node интерфейс

- DOM включва интерфейси за работа с нотации, единици (entities) и референции към тях (entity references)
 - Ако единиците не са били заменени от парсъра със съдържанието им:

```
<!ENTITY xml "eXtensible Markup Language">
<para>An &xml; value</para>
```



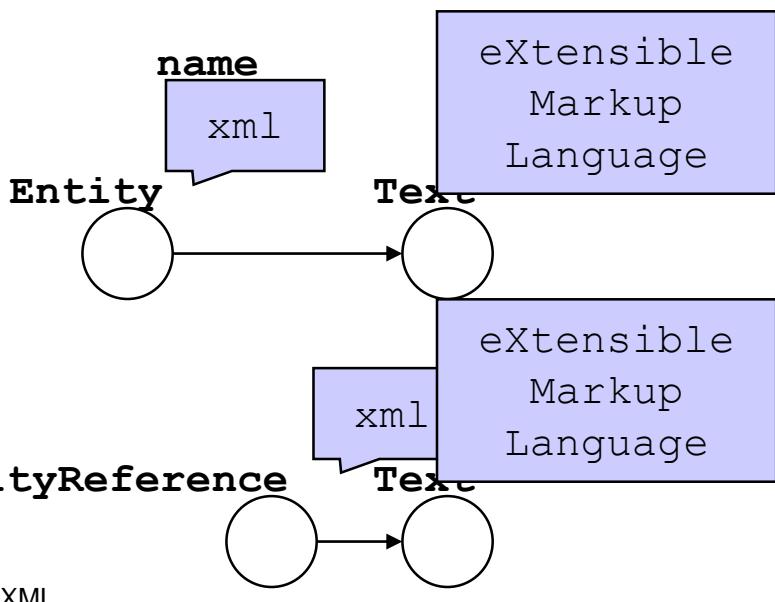
Entity::Node интерфейс

- Представя единица (parsed или unparsed в XML документ
 - Парсърът може да замени entity references или да създаде **EntityReference** възли
- Разширява интерфейса **Node** и добавя нови методи
- За non-parsable entities – достъп до името на нотацията:

```
String getPublicId();  
String getSystemId();  
String getNotationName();
```

Entity::Node интерфейс (2)

- Едно parsable Entity може да има възел дете, който представя стойността за замяна на единицата
- Всички единици на Document са достъпни чрез метода `getEntities()` на `DocumentType`



```

<!ENTITY MyBoat PUBLIC "BOAT" SYSTEM
"boat.gif" NDATA GIF>

String publicId = ent.getPublicId();
//BOAT

String systemId = ent.getSystemId();
//boat.gif

String notation =
ent.getNotationName(); //GIF
DOM

```

Notation::Node интерфейс

- Всяка декларация на нотация в DTD е представена чрез възел **Notation**
- Към интерфейса **Node** са добавени методите:
 - //Returns content of PUBLIC identifier
String getPublicId();
 - //Returns content of SYSTEM identifier
String getSystemId();
- Всичките нотации в даден **Document** са достъпни чрез метода **getNotations()** на обекта **DocumentType**

DocumentType::Node интерфейс

- Информация за съдържанието на DTD
- DOM 1.0 не позволява редактирането на този възел

```
//Returns name of document  
String getName();
```

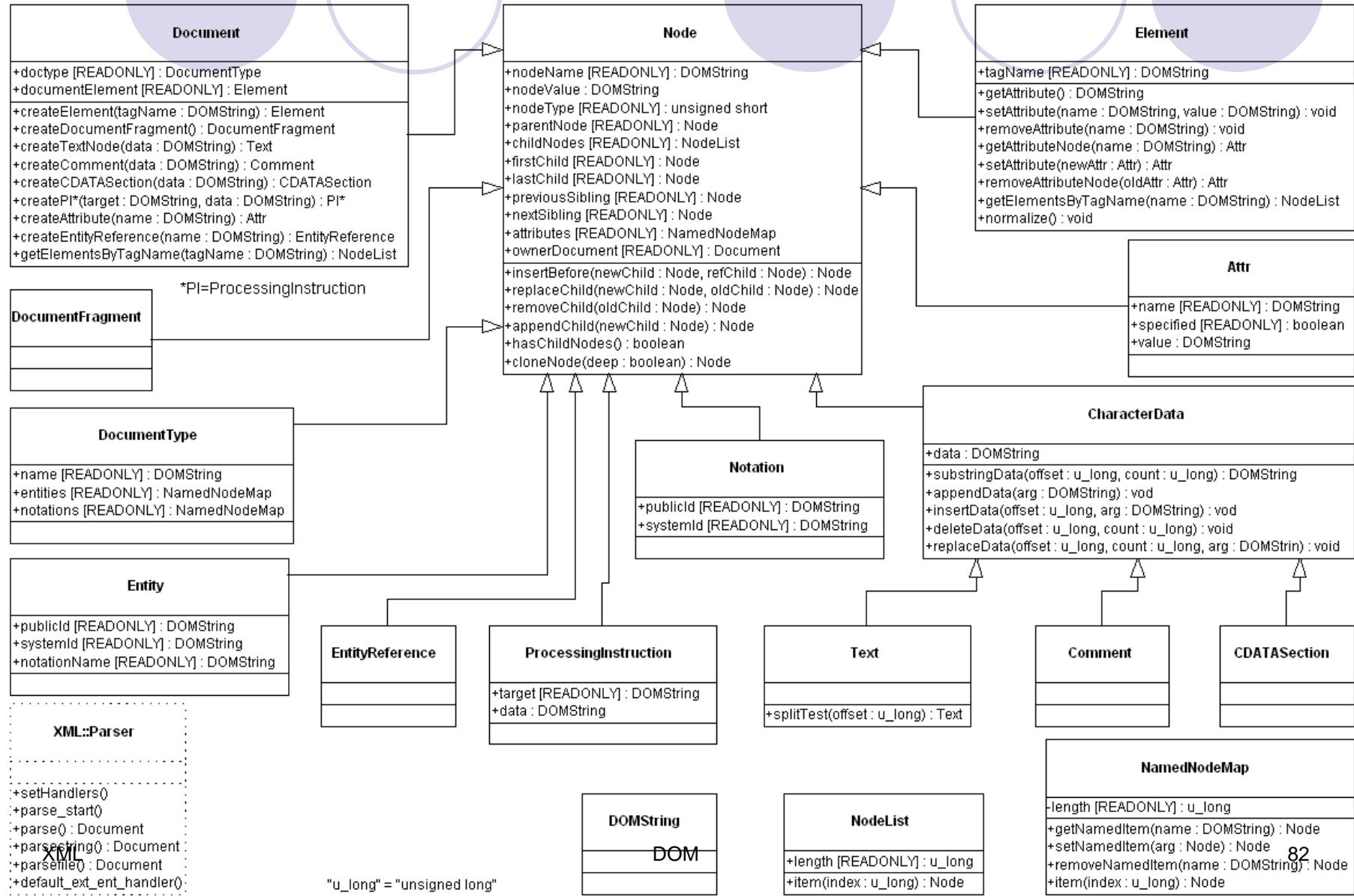
```
//Returns general entities declared in DTD  
NamedNodeList getEntities();
```

```
//Returns notations declared in DTD  
NamedNodeList getNotations();
```

Logical View

DOM (Core) Level One

Invoke "getName" to read instance variable `name` when using XML::DOM or XML4J

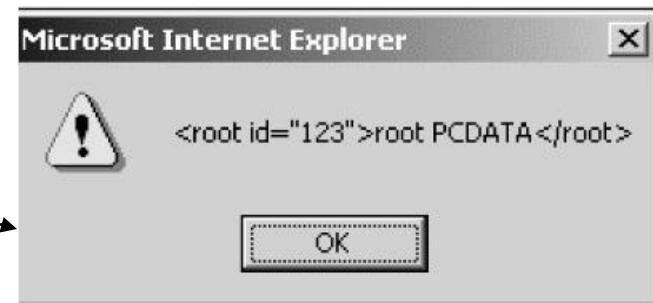


Създаване на XML документ from Scratch (DHTML)

```
<script language="JavaScript">
    var oDOM;
    oDOM = new ActiveXObject("MSXML.DOMDocument");
    var oNode, oText;
    oNode = oDOM.createElement("root");
    oText = oDOM.createTextNode("root PCDATA");
    oDOM.appendChild(oNode); oNode.appendChild(oText);

    var oAttr;
    oAttr = oDOM.createAttribute("id"); //set the attribute's value
    oAttr.nodeValue = "123"; //append the attribute to the element
    oNode.attributes.setNamedItem(oAttr);

    alert(oDOM.xml);
</script>
```



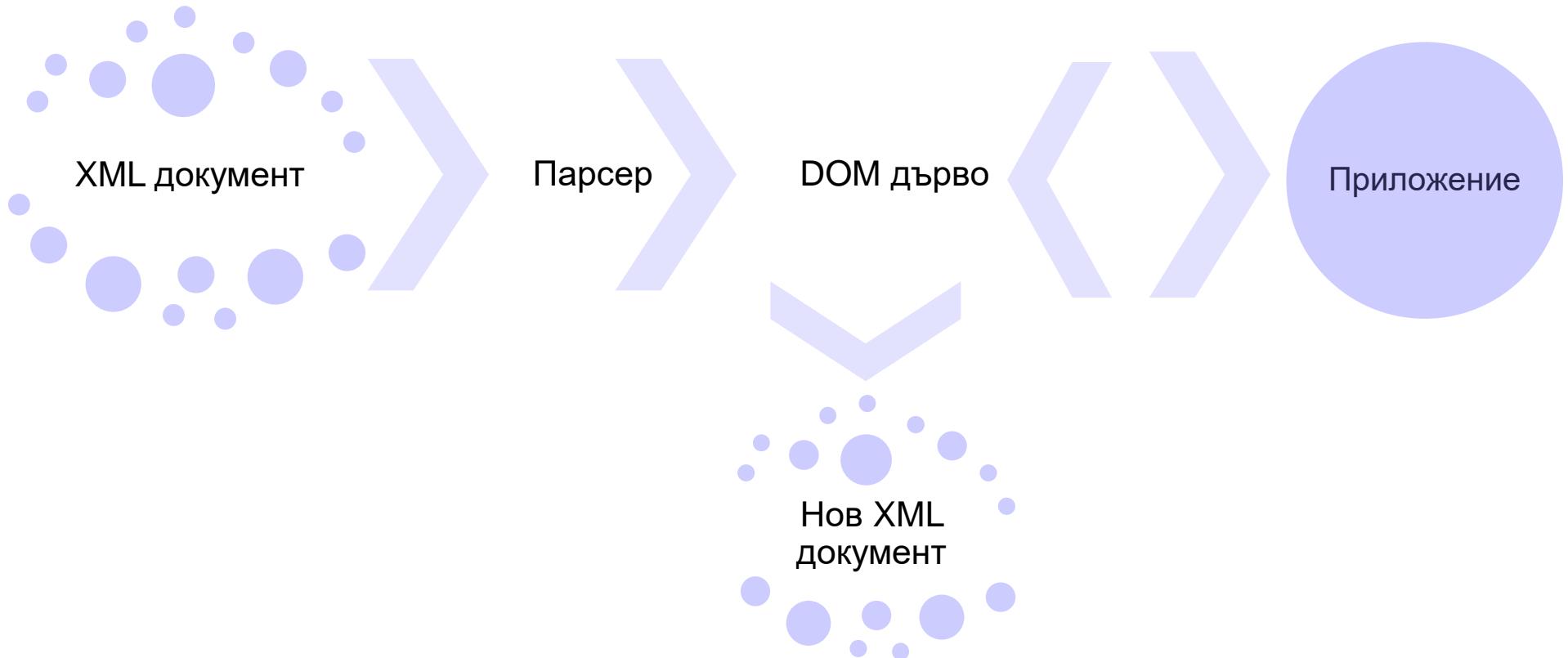
DOM обекти

- DOM обект \Leftrightarrow компилиран XML
- Спестяваме време и усилия ако изпращаме и получаваме DOM обекти вместо XML сорс – *използвайте сериализация!*
 - Спестяваме парсването на XML до DOM при изпращача и приемника
 - DOM обектът може да бъде по-голям от XML сорса

Размер на документа

- Натоварване на паметта
 - DOM изисква представяне на XML документа в паметта
- Осигуряване на интегритет
 - Заключване на документа при промяна на данните в него
 - Полза: предпазване на потребителите от получаване на некоректни данни или промяна на данни от друг потребител в процес на обработка
- Намаляване на системния товар
 - Използване на фрагменти
 - Забележка: За да бъде получен фрагмент, е необходимо да се зареди пълният документ

Обработка на документ с DOM

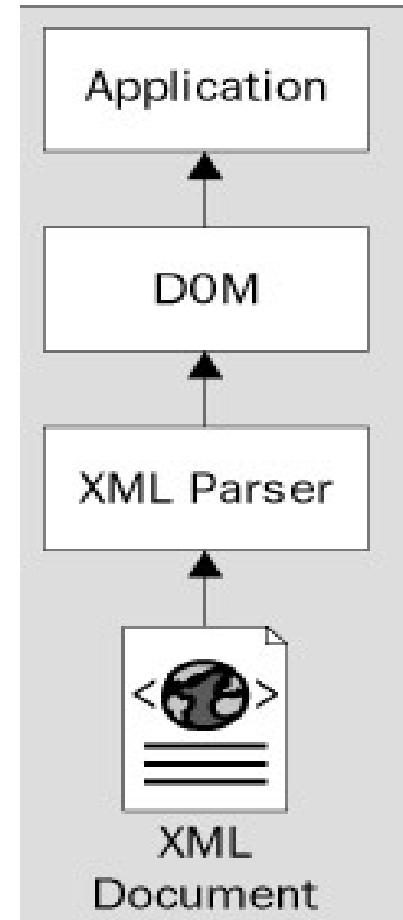


DOM спрямо XSL

- За сложно сортиране и преструктуриране на документа – използвайте DOM
- При DOM парсваме XML документа и после ползваме програмен код за манипулиране на DOM дървото. Този код има пълен достъп до възлите на дървото, без ограниченията на XSL
- XSL процесорът трансформира входния XML документ опосредствено – на база на правила, зададени в друг XML документ

DOM спрямо SAX

- При големи документи и ако извличаме само някои елементи – по-добре SAX
- При обработка на много елементи и структурни промени в XML документа – по-добре DOM
- При многократен достъп до XML документа – по-добре DOM.



За домашна работа

XML документ

```
<?xml version="1.0"?>
<SalesData Status="NewVersion">
    <Invoice InvoiceNumber="1"
        TrackingNumber="1"
        OrderDate="01012000"
        ShipDate="07012000"
        ShipMethod="FedEx"
        CustomerIDREF="Customer2">
        <LineItem Quantity="2" Price="5"
            PartIDREF="Part2" />
    </Invoice>
</SalesData>
```

```
</Invoice>
<Customer ID="Customer2"
    firstName="Bob"
    lastName="Smith"
    Address="2AnyStreet"
    City="Anytown" State="AS"
    PostalCode="ANYCODE" />
<Part PartID="Part2" PartNumber="13"
    Name="Winkle" Color="Red"
    Size="10" />
</SalesData>
```

Създаване на HTML документ и инстанция на Microsoft MSXML парсер

```
<HTML>
  <HEAD>
    <TITLE>DOM Demo</TITLE>
    <SCRIPT language="JavaScript">
      ...
    </SCRIPT>
  </HEAD>
  <BODY>
    <P>We have retrieved a lot<P>
  </BODY>
</HTML>
```



```
var objDOM;
objDOM = new ActiveXObject ("MSXML2.DOMDocument");
objDOM.async = false;
objDOM.load("salesData.xml");
```

Извеждане на кореновия елемент

```
//Get to the root element
document.write("<B>The root element:
</B>");
varSalesData = objDOM.documentElement;
alert(varSalesData.tagName);
document.write(varSalesData.tagName);
```

```
<?xml version="1.0"?>
<SalesData Status="NewVersion">
    <Invoice InvoiceNumber="1"
        TrackingNumber="1"
        OrderDate="01012000"
        ShipDate="07012000"
        ShipMethod="FedEx"
        CustomerIDREF="Customer2">
        <LineItem Quantity="2" Price="5"
            PartIDREF="Part2" />
```

Достъп до втория наследник на кореновия елемент

```
//Find the Customer elements and select the first one
document.write
(""<B><P>The name of the first Customer Element: </B>"");
varElemCust1 =
varSalesData.getElementsByName ("Customer").item(0);
alert(varElemCust1.xml);
document.write(varElemCust1.tagName);

<Customer
  ID="Customer2"
  firstName="Bob"
  lastName="Smith"
  Address="2AnyStreet"
  City="Anytown" State="AS"
  PostalCode="ANYCODE" />
```

Достъп до ID атрибута на елемента Customer

```
//Find the Customer ID Attribute  
document.write  
(""<B><P>The Customer ID attribute is: </B>");  
varAttrCustID = varElemCust1.getAttribute("ID");  
alert(varAttrCustID);  
document.write(varAttrCustID);
```

```
<Customer  
ID="Customer2"  
firstName="Bob"  
lastName="Smith"  
Address="2AnyStreet"  
City="Anytown" State="AS"  
PostalCode="ANYCODE" />
```

Достъп до атрибутите firstName и lastName на елемента customer

```
//Find the next attribute of Name  
document.write("<B><P>The customer's name is: </B>");  
varAttrFirstName =  
varElemCust1.getAttribute("firstName");  
alert(varAttrFirstName);  
document.write(varAttrFirstName);  
varAttrLastName = varElemCust1.getAttribute("lastName");  
alert(varAttrLastName);  
document.write(varAttrLastName);
```

```
<Customer  
ID="Customer2"  
firstName="Bob"  
lastName="Smith"  
Address="2AnyStreet"  
City="Anytown" State="AS"  
PostalCode="ANYCODE" />
```

Достъп до атрибутите ADDRESS и City на елемента customer

```
//Now let's write out the address  
document.write("<B><P>Their address is: </B>");  
varAttrAddr = varElemCust1.getAttribute("Address");  
alert(varAttrAddr);  
document.write(varAttrAddr);  
//Find the next attribute of City  
varAttrCity = varElemCust1.getAttribute("City");  
alert(varAttrCity);  
document.write(varAttrCity);
```

```
<Customer  
ID="Customer2"  
firstName="Bob"  
lastName="Smith"  
Address="2AnyStreet"  
City="Anytown" State="AS"  
PostalCode="ANYCODE" />
```

Добавяне на елемент в документа

```
//Find the Customer elements and select the first one
varElemCust1 = varSalesData.getElementsByTagName("Customer").item(0);
<!-- adding an element -->
document.write("<HR><H1>Updates appear in alert boxes:</H1>");
//create a new element
varNewElem = objDOM.createElement("MonthlySalesData");
//append the element
varNewElem = varSalesData.insertBefore(varNewElem, varElemCust1);
```

Добавяне на съдържание в елемент и създаване на атрибут

```
//create a new text-type node and append it
newText = objDOM.createTextNode("Can you see that we have created a new element?");
varNewElem.appendChild(newText);
alert(objDOM.xml);
<!-- adding an attribute -->
//create a new attribute and give it a value
varElemCust1.setAttribute("telephoneNo", "3591765524");
```

Добавяне на информация от друго DOM дърво

```
<?xml version="1.0"?>
<SalesData Status="NewVersion">
<Invoice InvoiceNumber="1"
    TrackingNumber="1"
    OrderDate="01012000"
    ShipDate="07012000"
    ShipMethod="FedEx"
    CustomerIDREF="Customer2">
    <LineItem Quantity="2" Price="5"
        PartIDREF="Part2" />
</Invoice>
```

```
<Customer ID="Customer1"
    firstName="Tom"
    lastName="Boswell"
    Address="39BrownhillCrescent"
    City="Anothertown" State="IN"
    PostalCode="OTHERCODE" />
<Part PartID="Part2" PartNumber="13"
    Name="Winkle" Color="Red"
    Size="10" />
</SalesData>
```

Получаване на достъп до кореновия елемент на първото DOM дърво

```
<SCRIPT language="JavaScript">
  var objDOM;
  objDOM = new ActiveXObject("MSXML2.DOMDocument");
  objDOM.async = true;
  objDOM.load("salesData.xml");
  //Get to the root element
  varSalesData = objDOM.documentElement;
  ...
</SCRIPT>
```

```
<?xml version="1.0"?>
<SalesData Status="NewVersion">
  <Invoice InvoiceNumber="1"
    TrackingNumber="1"
    OrderDate="01012000"
    ShipDate="07012000"
    ShipMethod="FedEx"
    CustomerIDREF="Customer2">
    <LineItem Quantity="2" Price="5"
      PartIDREF="Part2" />
```

Получаване на достъп до елемент Customer във второто DOM дърво

```
//second instance of the DOM
var objSecondDOM;
objSecondDOM =
    new
ActiveXObject ("MSXML2.DOMDocument");
objSecondDOM.async = true;
objSecondDOM.load ("salesData2.xml");
//Get to the root element
varSalesDataB =
objSecondDOM.documentElement;
varImportCust1 =
varSalesDataB.getElementsByName ("Customer").item(0);

<Customer ID="Customer1"
           firstName="Tom"
           lastName="Boswell"
           Address="39BrownhillCrescent"
           City="Anothertown"
           State="IN"
           PostalCode="OTHERCODE" />
```

Клониране на елемента Customer и добавяне на новия елемент в първото DOM дърво

```
<!-- adding an element -->  
document.write("<HR><H1>Updates appear in alert  
boxes:</H1>");  
  
//clone the node from the second DOM  
varClone = varImportCust1.cloneNode(true);  
  
//append node to the first DOM  
varSalesData.appendChild(varClone);
```

Императивна обработка на XML съдържание чрез Simple API for XML – SAX 2.0 и Streaming API for XML - StAX

Предимства на SAX
SAX интерфейси

Използване

StAX

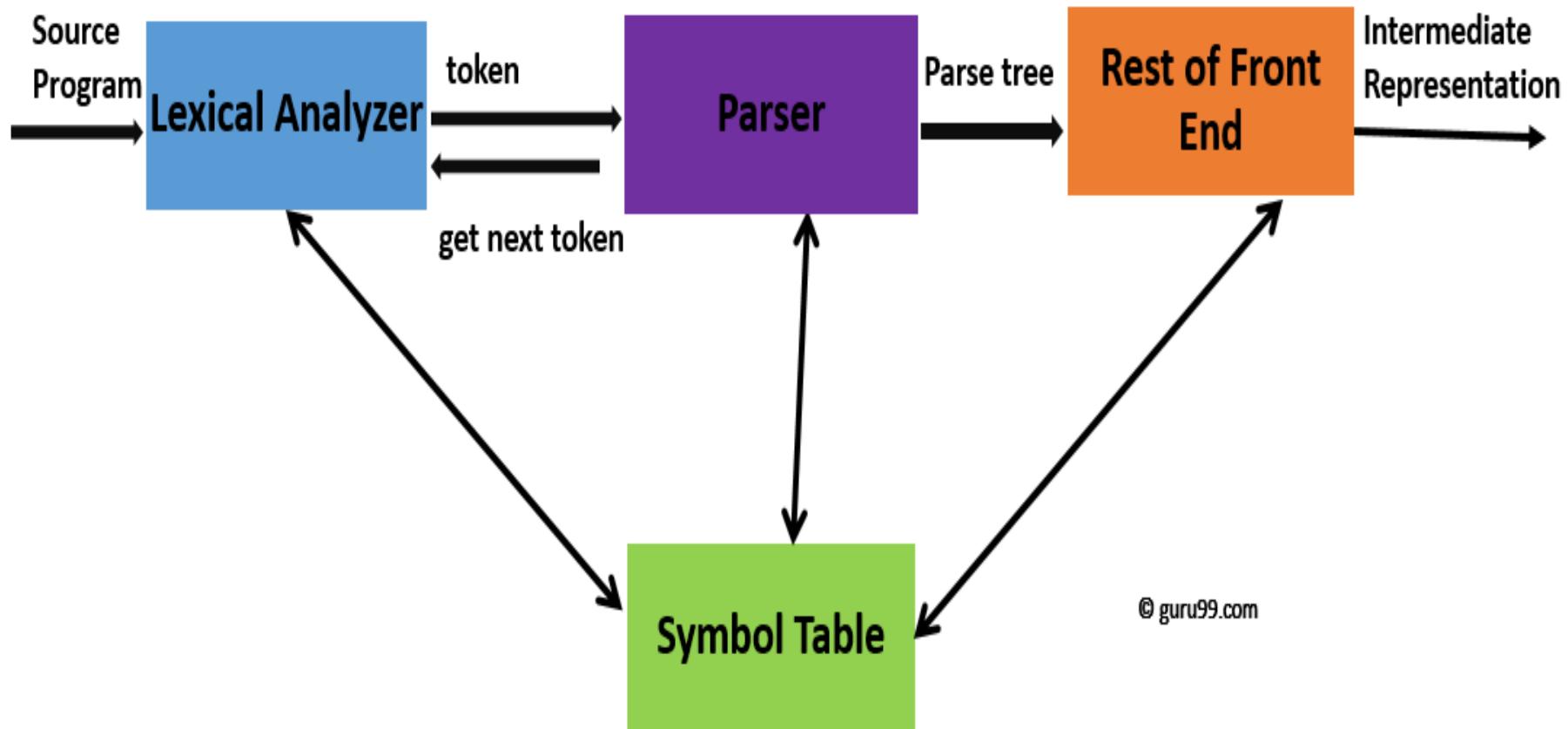
DOM, StAX и SAX



Анализ (разбор,
парсване) на текст



Процес на анализ (разбор, парсване) на текст



Използване на XML Parser

- Цел и приложение
- Три основни стъпки в използването на XML парсер:
 - Създаване на парсер-обект
 - Предаване на XML документа на парсера
 - Обработка на резултатите
- Принципно, генерацията на XML или друг формат е извън обхвата на парсерите

Типове XML парсъри

- Съществуват различни групи от типове:
 - Валидиращи спрямо невалидиращи парсъри
 - Парсери, използващи Document Object Model (DOM)
 - Парсери, използващи Simple API for XML (SAX)
 - Парсери, използващи Streaming API for XML (StAX)
 - Парсери, написани на конкретен език (Java, C++, Perl, etc.) без използване на определен API

Начини на обработка на XML

съдържание



Зареждане на целия
документ в паметта

Последователна
обработка на документа

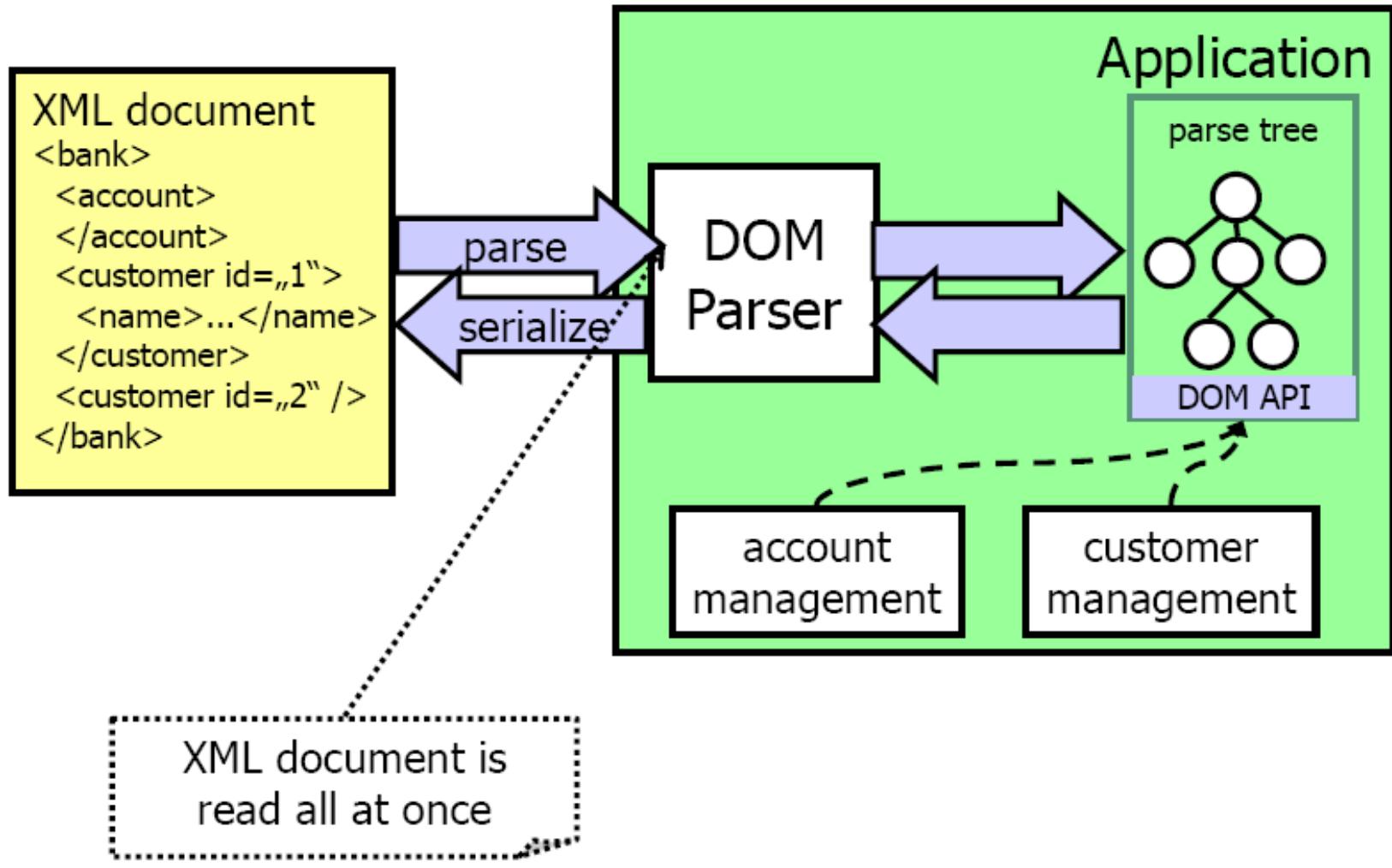
Последователна обработка

- Начини за последователна обработка
 - Обработка, базирана на събития
 - Обработка, базирана на съдържание
- Недостатъци на последователната обработка
 - Невъзможност за връщане назад при четене
 - Евентуална необходимост от повторно четене на документа при необходимост от валидация – напр. за намирането на съответствие между намерен ID и последващ IDREF – *как да я избегнем?*

Разбор (парсване) на XML съдържание

- Три широко-известни API's
 - DOM (Document Object Model)
 - Дефинира логическо дърво, представяще парсвания XML документ
 - SAX (Simple API for XML)
 - Дефинира манипулатори (handlers), съдържащи методи за разбор на XML документа (push)
 - Streaming API for XML (StAX)
 - Парсване на XML, базирано на итератори (a la push)
- Приложения без сложна манипулация на XML, но с ограничения по памет, могат да ползват SAX и StAX
- Структурната манипулация на XML елементи изиска използването на DOM

Работен процес на DOM

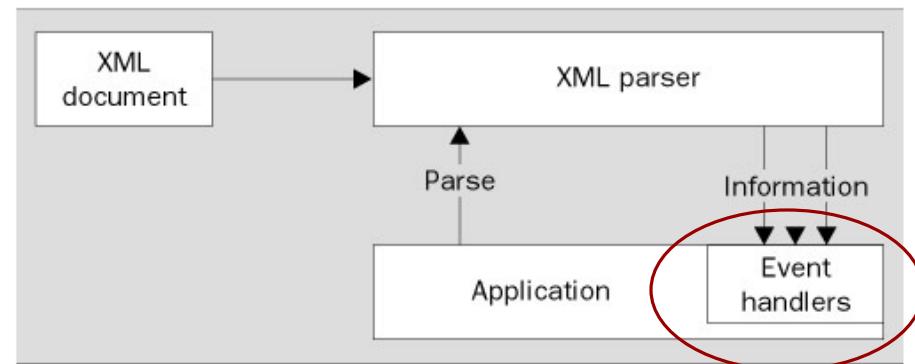
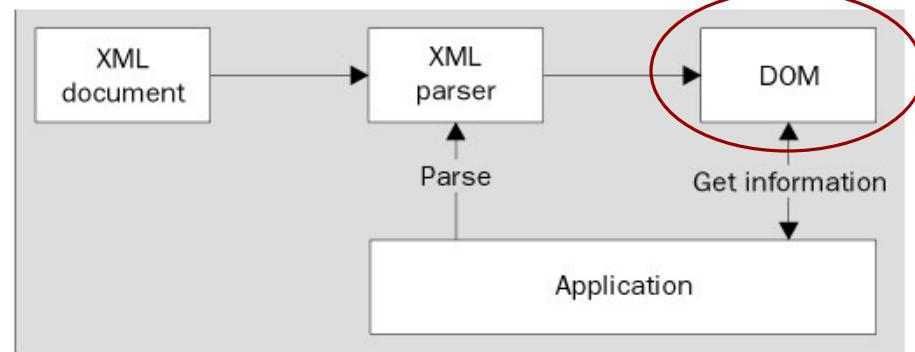


SAX спрямо DOM

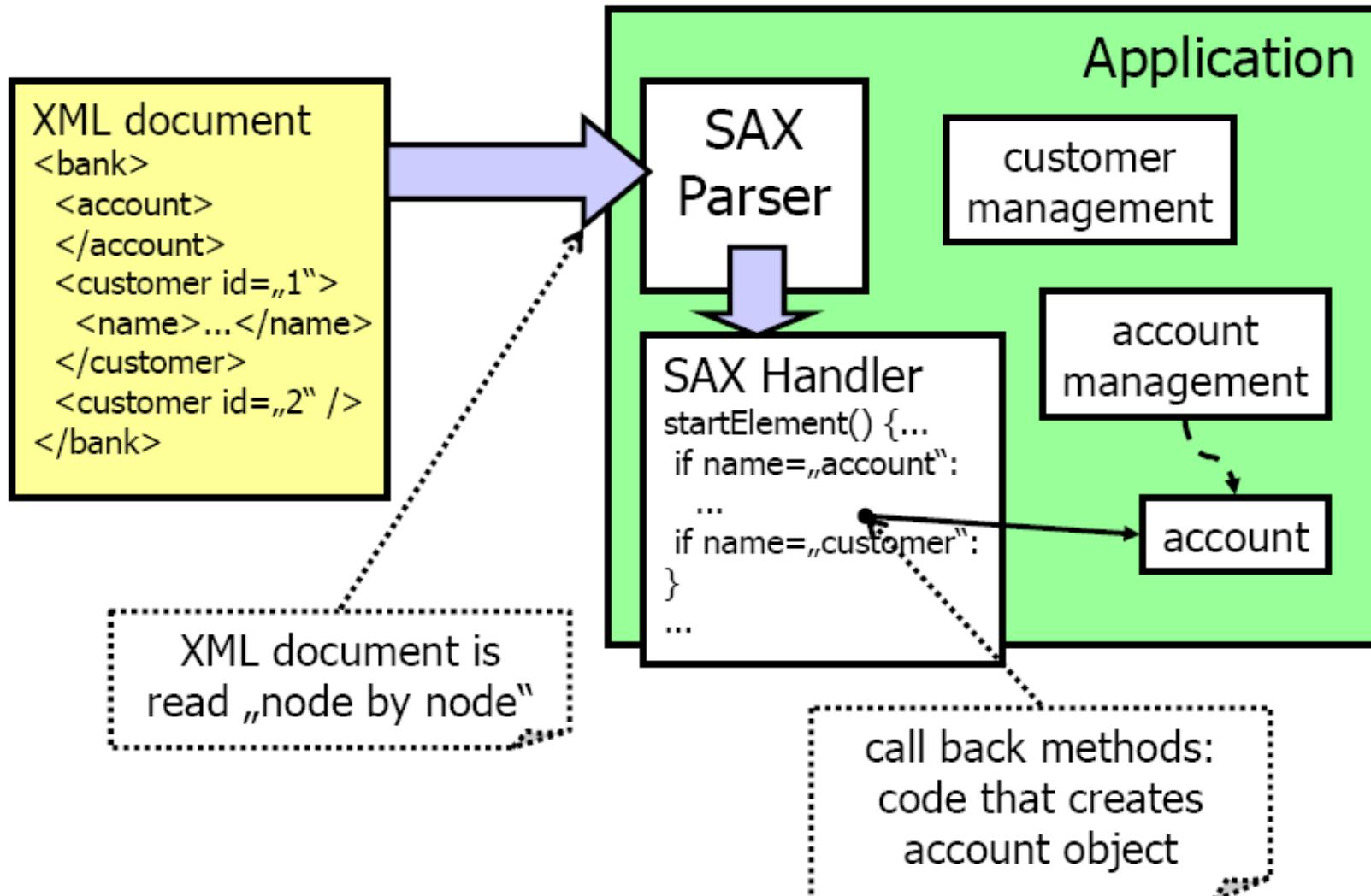
(фигури от Beginning XML, 2nd Edition)

- За **по-ефикасен** анализ на големи XML документи
- Цел: да реши проблема на DOM – създаването на масивно дърво на документа в паметта, преди да започнем работа с него, т.е. така да спести:
 - памет
 - време
- Решение:

SAX



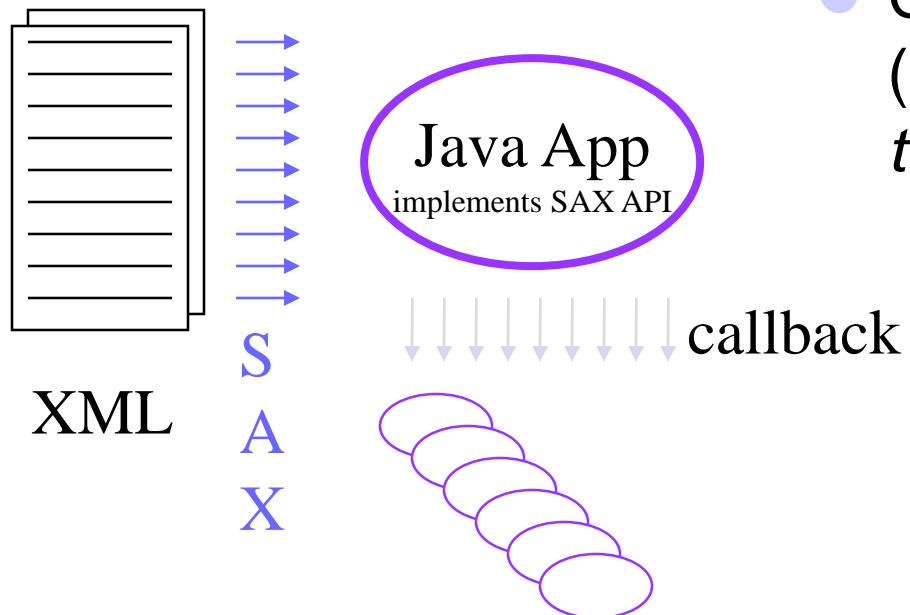
Работен процес на SAX



История на SAX

- Разработен от членовете на XML-DEV mailing list (сега поддържан от OASIS на <http://www.oasis-open.org/>), с цел *поефикасен анализ на големи XML документи*
- Спестява: **time & space**
- SAX 1.0 спецификация - 1998
- SAX 2.0, 2000 - подобрява се поддръжката на пространства с имена + строго придръжане към XML спецификацията
- SAX 2.0.2, 2004 - специфициран е като множество от Java интерфейси (www.saxproject.org, <http://sourceforge.net/projects/sax>)
- Голям брой SAX compliant парсъри – повечето отворени
 - <http://xml.apache.org/xerces-j>
 - Crimson
 - nanoXML
 - ...

Simple API for XML - SAX



- С управление по събития (*event-driven API*) вместо *tree-based API*
 - XML документът се изпраща до SAX парсера
 - XML файлът се прочита ред по ред
 - Парсерът известява за събития, вкл. и грешки
 - Имплементациите на API методите обработват събитията

Парсър, управляем по събития

- За документа:
 - <?xml version="1.0"?> <doc> <para>Hello, world!</para> </doc>
- Парсер, управляем по събития, ще създаде събитията:
 - start document
 - start element: doc
 - start element: para
 - characters: Hello, world!
 - end element: para
 - end element: doc
 - end document
- Приложението ги обработва, без да кешира целия документ

Simple API for XML - SAX

- SAX парсърът генерира събития
 - При начало и край на документа
 - При начало и край на елемент
 - При четене на символи в елемент
 - При грешки
 - При намиране на *negligible whitespace*
 - и много други...
- Използва *callback* механизъм за известяване на приложението
- Можем да напишем код за обработката на всяко събитие

A callback function that is passed (by reference) to another function, which calls the callback function under defined conditions (for example, upon completion).

SAX събития

- **startDocument**
 - Начало на обработката и възникване на първото събитие
- **endDocument**
 - Край на бработката и възникване на последното събитие
- **startElement**
 - Достигане до отварящ таг на елемент (<document>)
- **endElement**
 - Достигане до затварящ таг на елемент (</document>)
- **characters**
 - Достигане до низ от символи ("Sample text")
- **processingInstruction**
 - Достигане до инструкция за обработка (xmlstylesheet href="web.xsl" type="text/xml")
- **ignorableWhitespace**
 - Достигане до секция с празни символи, които не са част от документа
- **skippedEntry**
 - Възниква, когато външен обект е пропуснат
- **setDocumentLocator**
 - Позволява на парсера да подате Locator обект на приложението

DefaultHandler Class

Повече за SAX?

► <http://www.saxproject.org/>

Address <http://www.saxproject.org/> Go

SAX

General

[About SAX](#)
[Copyright Status](#)
[Events vs. Trees](#)
[FAQ](#)
[Links](#)

Java API

[Quickstart](#)
[Features and Properties](#)
[Filters](#)
[Namespaces](#)
[JavaDoc](#)

SAX Evolution

[Genesis](#)
[SAX 1.0 Overview](#)
[SAX 2.0 Changes](#)
[SAX 2.0 Extensions](#)
[Other Languages](#)

SourceForge Services

[Bugs/RFEs](#)
[Project Page](#)

About SAX

This is the official website for SAX. It replaces David Megginson's original [SAX page](#).

SAX is the *Simple API for XML*, originally a Java-only API. SAX was the first widely adopted API for XML in Java, and is a "de facto" standard. The current version is SAX 2.0.1, and there are versions for several programming language environments other than Java.

SAX has recently switched over to the SourceForge project infrastructure. The intent is to continue the open development and maintenance process for SAX (no NDAs required) while making it easier to track open SAX issues outside of the high-volume [xml-dev](#) list. Project resources include [archived mailing lists](#) and a [download area](#). See the Project Page link for full information about project facilities which are being used, as well as news announcements. Use the sax-users@lists.sourceforge.net mailing list to discuss problems that come up when you're trying to use SAX.

[David Megginson](#) has resumed maintaining SAX, after a period of excellent work by David Brownell (if you use SAX, you should think about buying his [SAX2 book](#)).

27-April 2004: SAX 2.0.2 (sax2 r3) is out. Download it by going to the Sourceforge [download area](#). That download includes full source, pre-generated javadoc, and a JAR file you can install.

29-January 2002: SAX 2.0.1 (sax2 r2) is out! Download it by going to the Sourceforge [download area](#). That download includes full source, pre-generated javadoc, and a JAR file you can install. Or, consult the javadoc link at left. That's current, and includes the preliminary "SAX2 Extensions 1.1" APIs.

12-November 2001: There are some SAX2 conformance tests available, using the JUNIT testing framework. Download the "sax2unit" tests at [the xmlconf project](#). These are in addition to the SAX2-hosted XML conformance tests mentioned on the "links" page, and address different issues.

SAX наръчник

← → C



docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html



The Java™ Tutorials

Search the online Java Tutorials

Submit

Hide TOC

Simple API for XML

When to Use SAX

Parsing an XML File

Using SAX

Implementing SAX

Validation

Handling Lexical Events

Using the DTDHandler and
EntityResolver

Further Information

« Previous • Trail • Next »

Home Page > Java API for XML Processing (JAXP) > Simple API for XML

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Parsing an XML File Using SAX

In real-life applications, you will want to use the SAX parser to process XML data and do something useful with it. This section examines an example JAXP program, `SAXLocalNameCount`, that counts the number of elements using only the `localName` component of the element, in an XML document. Namespace names are ignored for simplicity. This example also shows how to use a SAX `ErrorHandler`.

Note - After you have downloaded and installed the sources of the JAXP API from the [JAXP download area](#), the sample program for this example is found in the directory `install-dir/jaxp-1_4_2-release-date/samples/sax`. The XML files it interacts with are found in `install-dir/jaxp-1_4_2-release-date/samples/data`.

Creating the Skeleton

The `SAXLocalNameCount` program is created in a file named `SAXLocalNameCount.java`.

```
public class SAXLocalNameCount {  
    static public void main(String[] args) {  
        // ...  
    }  
}
```

Пакетът org.xml.sax.*

- Основен SAX1 интерфейс
 - **DocumentHandler** – управление на събитията за съдържанието на документа в SAX1:
 - StartDocument
 - EndDocument
 - StartElement
 - EndElement
 - Characters
- **Deprecated.** Този интерфеис е заменен в SAX2 от **ContentHandler**, който поддържа и Namespace.
- Имплементация на **DocumentHandler** - **HandlerBase**

org.xml.sax.ContentHandler

- Основен нов интерфейс в SAX2
 - **ContentHandler** – управление на събитията за съдържанието на документа. Съдържа:
 - StartDocument
 - EndDocument
 - StartElement
 - EndElement
 - Characters
- Подобен на SAX 1.0 DocumentHandler (deprecated), но с поддръжка на пространства от имена
- Имплементация на **ContentHandler**: **DefaultHandler**
- Забележка:
 - съществува Java клас с име **ContentHandler** в пакета java.net;
 - внимание с:
 - **import java.net.*; import org.xml.sax.*;**

Как да получаваме SAX събития

- `public class MyClass implements ContentHandler`
- `MyClass` ще имплементира callback методите на интерфейса за събития, свързани с парсваното съдържание (напр. събития за намирането на дадени елементи, атрибути и тяхното съдържание).
- Тези методи ще се извикват от SAX парсерът при настъпването на събитията.
- Интерфейсът `ContentHandler` съдържа голям брой методи, повечето от които са излишни за 80% от случаите. Затова SAX предоставя празна (default) имплементация на интерфейса, наречена DefaultHandler:
- `public class MyClass extends DefaultHandler`
- Можем да пренапишем (method overriding) онези методи, които обработват важни за нас събития.

Пример от “Beginning XML” – *the Band XML*

```
<?xml version="1.0"?>
<bands>
  <band type="progressive">
    <name>King Crimson</name>
    <guitar>Robert Fripp</guitar>
    <saxophone>Mel
    Collins</saxophone>
    <bass>Boz</bass>
    <drums>Ian Wallace</drums>
  </band>
  <band type="punk">
    <name>X-Ray Spex</name>
    <vocals>Poly
    Styrene</vocals>
    <saxophone>Laura
    Logic</saxophone>
    <guitar>Someone
    else</guitar>
  </band>
```

```
<band type="classical">
  <name>Hilliard Ensemble</name>
  <saxophone>Jan
  Garbarek</saxophone>
</band>
<band type="progressive">
  <name>Soft Machine</name>
  <organ>Mike Ratledge</organ>
  <bass>Hugh Hopper</bass>
  <drums>Robert Wyatt</drums>
  <saxophone>Elton
  Dean</saxophone>
</band>
</bands>
```

Примерен Java код 1/2

```
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class BandReader extends DefaultHandler
{
    public static void main(String[] args) throws Exception
    {
        System.out.println("Here we go ...");
        BandReader readerObj = new BandReader();
        readerObj.read(args[0]);
    }
}
```

Примерен Java код 2/2

```
public void read (String fileName) throws Exception
{
    XMLReader readerObj =
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAX
Parser");
    readerObj.setContentHandler (this);
    //we registered for interface callback!!
    readerObj.parse (fileName);
}

public void startDocument() throws SAXException
{ System.out.println("Starting ...");
}

public void endDocument() throws SAXException
{ System.out.println("... Finished");
}

public void startElement(String uri, String localName, String qName, Attributes
atts) throws SAXException
{ System.out.println("Element is " + qName);
}
```

Результат

- Now let's run it:
 - `java BandReader bands.xml`
- Here's what we see:
 - Here we go ...
 - Starting ...
 - Element is bands
 - Element is band
 - Element is name
 - Element is guitar
 - Element is saxophone
 -
.....
 - Finished

XMLReader (SAX 2.0)

- public interface **XMLReader** – интерфейс за читане на XML документ с използване на callbacks
- Въпреки името си, този интерфейс не разширява стандартния Java **Reader** интерфейс, защото четенето на XML е много по-различно от простото четене на символни данни
- Разрешава приложението:
 - да зададе и провери свойства (features, properties) за парсъра,
 - да регистрира обработчици на събития (event handlers) за обработка на документа и
 - да инициира самото парсване

XMLReader (SAX 2.0) спрямо Parser (SAX 1.0)

- Всички SAX интерфейси са **синхронни**:
 - Методите `parse` не връщат управлението преди приключване на парсването, и
 - Четците (`readers`) трябва да изчакат връщане от `event-handler callback` преди да рапортuvат следващото събитие
- Интерфейсът `XMLReader` заменя SAX 1.0 `Parser` (deprecated). `XMLReader` добавя две важни неща към стария `Parser` интерфейс:
 - Стандартен начин за задаване и проверка на свойства (`features`, `properties`) за парсера;
 - Поддръжка на пространства от имена

Интерфейс XMLReader

- Регистрира други обекти за callbacks
 - `void setContentHandler(ContentHandler handler)`
 - `void setDTDHandler(DTDHandler handler)` – разрешава приложението да регистрира DTD event handler:
 - Ако приложението не регистрира DTD handler, всички DTD събития, рапортувани от SAX парсера, се игнорират.
 - Важно: приложението може да регистрира нов (различен) handler в процеса на парсване и от този момент SAX парсерът започна да го използва.
 - `void setErrorHandler(ErrorHandler handler)` – разрешава приложението да регистрира error handler.
 - `void setEntityResolver(EntityResolver resolver)` – ...
- Стартираме парсването чрез метода `parse()`
- Когато парсерът прочете значим за него участък от документа, той извиква метод от регистрирания обект
- Парсерът продължава четенето на XML файла след изпълнението на метода

Отново за ContentHandler

- Интерфейс за получаване на основни събития по маркирано съдържание
- Или използваме базовата имплементация – **HandlerBase** (SAX1) или **DefaultHandler** (SAX2) класа и преписваме някои методи
- Или имплементираме интерфейса **ContentHandler**

```
class myClass implements ContentHandler {  
    ...  
    myParser.setContentHandler(this);  
    ...  
}  
XML
```

ContentHandler имплементации

- **DefaultHandler** (SAX2) – заменя класа **HandlerBase** от SAX1 ;
- Предоставя базови имплементации за всички callbacks на 4 основни обработващи интерфейси в SAX2:
 - [EntityResolver](#)
 - [DTDHandler](#)
 - [ContentHandler](#)
 - [ErrorHandler](#)
- **XMLFilterImpl** – стои между [XMLReader](#) и event handlers на приложението и предава събитията на обработчиците без промяна; негови подкласове могат да препишат специфични методи
- **XMLReaderAdapter** - обвива SAX2 [XMLReader](#) и го представя като SAX1 [Parser](#)
- *Кодът и документацията са Public Domain -> NO WARRANTY*

ContentHandler методи 1/2

- Интерфейсни методи (повечето хвърлят SAXException):
 - `void startDocument()` – извикван в началото на документа
 - `void endDocument()`
 - `void startElement(String namespaceURI, String localname, String qName, Attributes attr)`
 - извикван в началото на всеки елемент, с параметри:
 - **Namespace URI** и **local name** – изискват се при стойност на свойството namespaces равна на `true` (default), и са опционални при `namespaces == false`; за повече виж <http://www.saxproject.org/namespaces.html>
 - **localName** – локално име (без префикс), ако не се извършва обработка на Namespace
 - **qName** - квалифицирано име (с префикс) или празен стринг ако не се използват такива имена
 - **attr** – атрибутите на елемента

ContentHandler методи 2/2

- `void endElement(String name) throws SAXException`
– при край на всеки елемент в XML документа; кореспондира си със събитието `startElement`
- `void characters(char[] ch, int start, int length) throws SAXException` - при четене на символни данни; приложението не трябва да опитва да чете извън обхвата
- `void ignorableWhitespace(char[] ch, int start, int length)` – извикван при ignorable whitespace в съдържанието на елемент. Валидиращите парсъри използват този метод за рапортuvане на всяка порция от празни пространства
- `void processingInstruction(String target, String data)` – при инструкция за обработка

SAXFinder пример от Beginning XML 2nd Ed. 1/2

```
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class SaxFinder extends DefaultHandler
{
    private StringBuffer saxophonist = new StringBuffer(); private boolean isSaxophone = false;

    public static void main(String[] args) throws Exception
    { System.out.println("Here we go ...");
        SaxFinder readerObj = new SaxFinder();
        readerObj.read(args[0]);
    }

    public void read (String fileName) throws Exception
    { XMLReader readerObj =
            XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
        readerObj.setContentHandler (this);
        readerObj.parse (fileName);
    }

    public void startDocument() throws SAXException
    { System.out.println("Starting ...");
    }
```

SAXFinder пример от Beginning XML 2nd Ed. 2/2

```
public void endDocument() throws SAXException
{
    System.out.println("... Finished");
}

public void startElement(String uri, String localName, String qName, Attributes atts) throws
SAXException
{
    if (qName.equals("saxophone"))
    {
        isSaxophone = true;
        saxophonist.setLength(0);
    }
    else
        isSaxophone = false;
}

public void endElement(String uri, String localName, String qName)    throws SAXException
{
    if (isSaxophone)
    {
        System.out.println("Saxophonist is " + saxophonist.toString());
        isSaxophone = false;
    }
}

public void characters(char[] chars, int start, int len)      throws SAXException
{
    if (isSaxophone)
        saxophonist.append(chars, start, len);
}
```

Извличане на атрибути

- **public void startElement(String uri, String localName, String qName, Attributes atts) throws SAXException**
- Четири основни метода за **atts** обекта:
 - **getLength** – връща броя на атрибутите в списъка
 - **getQName** – връща квалифицираното име на атрибутит на дадена позиция в списъка (броене от 0).
 - **getValue** – връща стойност на атрибут (определен по име или по позиция от 0 до N-1)
 - **getType** – връща тип на атрибут (определен по име или по позиция от 0 до N-1).

чтение на XML файл 1-5

```
<People>
  <Person bornDate="1874-11-30"
           diedDate="1965-01-24">
    <Name>Winston
    Churchill</Name>
    <Description>
      Winston Churchill was a
      mid-20th ...
    </Description>
  </Person>
  <Person bornDate="1917-11-19"
           diedDate="1984-10-31">
    <Name>Indira Gandhi</Name>
    <Description>
```

```
      Indira Gandhi was India's
      first female...
    </Description>
  </Person>
  <Person bornDate="1917-05-29"
           diedDate="1963-11-22">
    <Name>John F. Kennedy</Name>
    <Description>
      JFK, as he was
      affectionately known, ...
    </Description>
  </Person>
</People>
```

Четене на XML файл 2-5

```
public class SaxParser1 extends DefaultHandler {  
    public void startDocument( ) throws SAXException {  
        System.out.println( "SAX Event: START DOCUMENT" );  
    }  
    public void endDocument( ) throws SAXException {  
        System.out.println( "SAX Event: END DOCUMENT" );  
    }  
    public void startElement(String namespaceURI, String  
localName, String qName, Attributes attr) throws SAXException  
{  
        System.out.println( "SAX Event: START ELEMENT[ " +  
localName + " ]" );  
    }  
    public void endElement(String namespaceURI, String  
localName, String qName) throws SAXException {  
        System.out.println( "SAX Event: END ELEMENT[ " + localName  
*ML" ]" );  
    }  
}
```

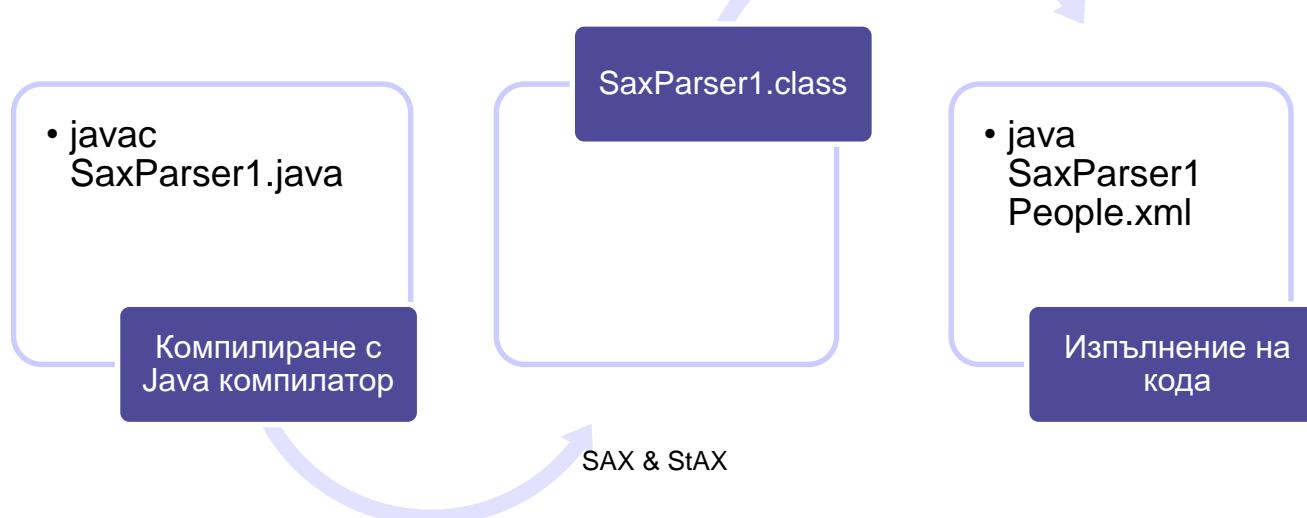
чтение на XML файл 3-5

```
public void characters(char[] ch, int start,  
int length ) throws SAXException {  
    System.out.print( "SAX Event: CHARACTERS [ "  
);  
    try {  
        OutputStreamWriter output = new  
OutputStreamWriter(System.out);  
        output.write( ch,start,length );  
        output.flush();  
    } catch ( Exception e ) {  
        e.printStackTrace();  
    }  
    System.out.println( " ]" );  
}
```

чтение на XML файл 4-5

```
public static void main( String[] argv ) {  
    String inputFile = argv[0];  
    System.out.println( "Processing " + inputFile  
+ " ." );  
    System.out.println( "SAX Events:" );  
    try {  
        XMLReader reader =  
XMLReaderFactory.createXMLReader();  
        reader.setContentHandler( new SaxParser1() );  
        reader.parse( new InputSource( new  
FileReader( inputFile ) ) );  
    } catch ( Exception e ) {  
        e.printStackTrace();  
    }  
}
```

Четене на XML файл 5-5



SAX Events:

```
SAX Event: START DOCUMENT
SAX Event: START ELEMENT [ People ]
SAX Event: CHARACTERS [
]
SAX Event: START ELEMENT [ Person ]
SAX Event: CHARACTERS [
]
SAX Event: START ELEMENT [ Name ]
SAX Event: CHARACTERS [ Winston Churchill ]
...
SAX Event: END ELEMENT [ People ]
SAX Event: END DOCUMENT
```

Четене на Атрибути 1-2

```
public void startElement (String namespaceURI, String
localName, String qName,
                        Attributes attr ) throws
SAXException {
    System.out.println( "SAX Event: START ELEMENT[ " + localName
+ " ]" );
    for ( int i = 0; i < attr.getLength(); i++ ) {
        System.out.println( " ATTRIBUTE: " + attr.getLocalName(i)
+ " VALUE: "
                            + attr.getValue(i) );
    }
}
```

- Достъп до конкретен атрибут в колекцията Attributes

getValue(String qName)

getValue(String uri, String localName)

Четене на Атрибути 2-2

SAX Event: START ELEMENT[Person]

ATTRIBUTE: bornDate **VALUE:** 1917-05-29

ATTRIBUTE: diedDate **VALUE:** 1963-11-22

SAX Event: CHARACTERS [

]

SAX Event: START ELEMENT[Name]

SAX Event: CHARACTERS[John F. Kennedy]

SAX Event: END ELEMENT[Name]

SAX Event: CHARACTERS [

]

SAX Event: START ELEMENT[Description]

SAX Event: CHARACTERS [

JFK, as he was affectionately known, was a
United States president
who was assassinated in Dallas, Texas.]

SAX Event: CHARACTERS [

]

SAX Event: END ELEMENT[Description]

XML SAX Event: CHARACTERS [SAX & StAX

] ...

Прихващане на събитие characters

1-2

- При вложени текстови елементи възниква повече от едно събитие characters
- Деклариране на обект StringBuffer

```
public class SaxParser3 extends DefaultHandler {
```

```
    private StringBuffer buffer = new StringBuffer();
```

- Добавяне на обработчик за събитието startElement

```
public void startElement(String namespaceURI, String  
localName, String qName, Attributes attr) throws SAXException  
{
```

```
    System.out.println("SAX Event: START ELEMENT[ " + localName  
+ " ]");
```

```
    for (int i = 0; i < attr.getLength(); i++) {  
        System.out.println(" ATTRIBUTE: " + attr.getLocalName(i)  
+ " VALUE: " + attr.getValue(i));  
    }
```

```
XML buffer.setLength(0);
```

SAX & StAX

Прихващане на събитие characters

2-2

- Добавяне на текст в буфера в обработчика на събитието characters

```
public void characters(char[] ch, int start, int length )
throws SAXException {
    try {
        buffer.append(ch, start, length);
    } catch (Exception e) {
        e.printStackTrace();    }
}
```

- Преобразуване на буфера в символен низ в обработчика на събитието endElement

```
public void endElement(String namespaceURI, String localName,
String qName ) throws SAXException {
    System.out.print("SAX Event: CHARACTERS[ " );
    System.out.println(buffer.toString());
    System.out.println( " ]" );
    System.out.println( "SAX Event: END ELEMENT[ " + localName +
" ]" );
}
```

събитието ignorableWhitespace

- Парсерът използва DTD дефиницията (ако е реферирана в документа), за да определи дали секциите с празни символи са част от документа
- Ако в DTD дефиницията е указано, че елемент може да съдържа единствено символни данни от тип PCDATA, то наличието на празните редове ще доведе до възникване на събитието ignorableWhitespace

```
public void ignorableWhitespace(char[ ] ch,  
                                int start, int len) throws SAXException
```

събитието skippedEntity

- Възниква, когато SAX парсерът достигне до съдържание, което приложението може или трябва да пропусне
 - Референция към външен източник, който не може да бъде парснат или открит
 - Външен глобален ресурс със стойност false на характеристиката <http://xml.org/sax/features/external-general-entities>
 - Външен параметричен ресурс със стойност false на характеристиката <http://xml.org/sax/features/external-parameter-entities>
 - Името на параметъра, който трябва да се пропусне, се подава от събитието с ‘%’

```
public void skippedEntity(String name) throws SAXException
```

Събитието processingInstruction

- Възниква, когато SAX парсерът достигне до инструкция за обработка

```
public void processingInstruction(String target, String data)  
throws SAXException
```

- Пример

```
<?xmlstylesheet type="text/xsl" href="myTransform.xsl"?>
```

- Параметърът target се инициализира с xmlstylesheet
 - Параметърът data съдържа type="text/xsl"
href="myTransform.xsl"
- **XML декларацията в началото на XML документа не се третира като инструкция за обработка!**

Прихващане на невалидно съдържание

- DTD и XML Schema се прилагат при валидация, но осигуряват ограничени възможности
- Пример за случай, който не може да се дефинира с DTD и XML Schema
 - Ако атрибут *x* е равен на *y*, то следващ елемент трябва да бъде *<a>*, в противен случай **
- Изключението SAXException
 - Създава изключение с дефинирано потребителско съобщение **SAXException(String message)**
 - Прихваща предефинирано изключение от тип Exception **SAXException(Exception e)**
 - Прихваща предефинирано изключение от тип Exception и дефинира потребителски дефинирана информация за него **SAXException(String message, Exception e)**

Събитието setDocumentLocator

- Използва се предимно за подобряване на разбираемостта на съобщенията за грешка
- Получава като аргумент обект от клас **Locator**
- getLineNumber()**
 - Връща номера на реда за текущото събитие
- getColumnNumber()**
 - Връща номера на колоната за текущото събитие (SAX спецификацията номерира колоните отляво наясно)
- getSystemId()**
 - Връща системния идентификатор на документа за текущото събитие
- getPublicId()**
 - Връща публичния идентификатор на документа за текущото събитие

Събитието setDocumentLocator:

Пример 1-2

- Създаване на екземпляр на класа Locator

```
public class SaxParser4 extends DefaultHandler {  
    private Locator docLocator = null;  
    private StringBuffer buffer = new StringBuffer();
```

- Добавяне на обработчик за събитието setDocumentLocator

```
public void setDocumentLocator(Locator locator)  
{  
    docLocator = locator;  
}
```

Събитието setDocumentLocator:

Пример 2-2

- Получаване на текущия номер на ред в събитието startElement

```
public void startElement(String namespaceURI, String localName,
String qName, Attributes attr) throws SAXException {
    int lineNumber = 0;
    if (docLocator != null)
    {
        lineNumber = docLocator.getLineNumber();
    }
    System.out.println("SAX Event: START ELEMENT[ " + localName + " ]");
    if (lineNumber != 0)
    {
        System.out.println("\t"(Found at line number: " + lineNumber +
"."));
    }
    for (int i = 0; i < attr.getLength(); i++) {
        System.out.println(" ATTRIBUTE: " + attr.getLocalName(i) + " "
VALUE: " + attr.getValue(i));
    }
}
```

Трябва да проверим дали обектът Locator не е Null - ако парсерът не поддържа Locator обект.

Събитието setDocumentLocator: Резултат

Processing 'people.xml'.

SAX Events:

SAX Event: START DOCUMENT

SAX Event: START ELEMENT[People
]
(Found at line number: 1.)

SAX Event: START ELEMENT[Person
]
(Found at line number: 2.)

ATTRIBUTE: bornDate VALUE:
1874-11-30

ATTRIBUTE: diedDate VALUE:
1965-01-24

SAX Event: START ELEMENT[Name]
(Found at line number: 3.)

SAX Event: CHARACTERS[Winston
Churchill
XML
]
SAX & StAX

SAX Event: END ELEMENT[Name]

SAX Event: START ELEMENT[
Description]

(Found at line number: 4.)

SAX Event: CHARACTERS[
Winston Churchill was a mid
20th century

British politician who
became famous as

Prime Minister during the
Second World
War.
]

SAX Event: END ELEMENT[
Description]

SAX Event: CHARACTERS[
...
SAX & StAX

Интерфейсът ErrorHandler

- Осигурява информация за възникнали грешки
- Събития
 - **warning**
 - Осигурява възможност на парсера да извести (нотифицира) приложението с предупреждение
 - **error**
 - Осигурява възможност на парсера да нотифицира приложението за грешка (не блокира парсването)
 - **fatalError**
 - Осигурява възможност на парсера да нотифицира приложението за фатална грешка (блокира парсването)
- Стъбове за събитията се осигуряват от класа **DefaultHandler**
 - Генерира **SAXException** при настъпването на събитията

Error Handler интерфейс

- Ако SAX приложение трябва да имплементира специфична обработка на грешка, то трябва да:
 - имплементира **Error Handler** интерфейса и да регистрира екземпляр в XML reader чрез метода [setErrorHandler](#); тогава парсерът ще рапортова всички грешки през този интерфейс
 - не указва къде е станала грешката
- Три нива на изключения
 - **void error(SAXParseException ex);** – викан при възстановяма грешка
 - **void fatalError(SAXParserException ex);** – викан при невъзстановяма грешка
 - **void warning(SAXParserException ex);**

Интерфейсът ErrorHandler: пример

1-3

```
public static void main( String[] argv ) {
    String inputFile = argv[0];
    System.out.println("Processing '" + inputFile + "'.");
    System.out.println( "SAX Events:" );
    try {
        XMLReader reader = XMLReaderFactory.createXMLReader();
        SaxParser parser = new SaxParser();
        reader.setContentHandler(parser);
        reader.setErrorHandler(parser);
        reader.parse( new InputSource(new FileReader(
inputFile )) );
    }catch ( Exception e ) {
        e.printStackTrace();
    }
}
```

Интерфейсът ErrorHandler: пример 2-3

- Активиране на характеристиката за валидация

```
try
{
    reader.setFeature(
        "http://xml.org/sax/features/validation", true);
} catch (SAXException e) {
    System.err.println("Cannot
activate validation");
}
reader.parse(new InputSource(
    new
FileReader(inputFile )));
```

- Създаване на DTD дефиниция

```
<!DOCTYPE People [
<!ELEMENT People (Person*)
<!ELEMENT Person (Name,
Description)>
<!ATTLIST Person bornDate CDATA
#REQUIRED>
<!ATTLIST Person diedDate CDATA
#REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
]>
<!-- rest of people.xml -->
```

Интерфейсът ErrorHandler: пример

3-3

- Създаване на обработчик за събитието warning

```
public void warning (SAXParseException exception) throws SAXException {  
    System.err.println("[Warning] " + exception.getMessage() + " at  
line " + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() ); }
```

- Създаване на обработчик за събитието error

```
public void error (SAXParseException exception) throws SAXException {  
    System.err.println("[Error] " + exception.getMessage() + " at line  
" + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() ); }
```

- Създаване на обработчик за събитието fatalError

```
public void fatalError (SAXParseException exception) throws  
SAXException {  
    System.err.println("[Fatal Error] " + exception.getMessage() + " at  
line " + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() );  
    XMLthrow exception; }
```

Интерфейсът ErrorHandler: результат от примера

- При премахване на атрибута `diedDate` от втория `<Person>` елемент (`Indira Gandhi`)
- Съобщение за грешка при обработка на элемента

```
[Error] Attribute "diedDate" is required and must be  
specified for element type  
"Person" at line 17, column 33  
SAX Event: START ELEMENT[ Person ] (Found at line number:  
17.)  
ATTRIBUTE: bornDate VALUE: 1917-11-19
```

DTDHandler интерфейс

- Предоставя методи за известяване за DTD събития
 - notationDecl
 - Осигурява възможност на парсера да нотифицира приложението за декларации на нотации
 - unparsedEntityDecl
 - Осигурява възможност на парсера да нотифицира приложението за декларации на ресурси, които не подлежат на парсване
- Ако SAX приложението трябва да информира за нотации и единици (entities), то трябва да имплементира този интерфейс и да регистрира негов екземпляр в парсера чрез метода **setDTDHandler**.
 - **void notationDecl(String name, String publicId, String systemId)**
Известяван за декларация на нотация
 - **void unparsedEntityDecl(String name, String publicId, String systemId, String notationName)**
Известяван за декларация на unparsed entity

Интерфейс EntityResolver 1-2

- Определя поведението на SAX парсера при преобразуване на външни референции в рамките на DTD дефиниция
- Осигурява функция `resolveEntity`, позволяваща на приложението да прихваща събития, възникващи при опит на SAX парсера да преобразува външна референция
- Реализира се от класа `DefaultHandler`
- Начин на използване
 - Аналогичен на начина на използване на интерфейстите `ContentHandler` и `ErrorHandler`

```
reader.setEntityResolver(SaxParser);
```

Интерфейс EntityResolver 2-2

- Метод **resolveEntity**

```
InputSource resolveEntity(String publicId, String systemId)  
<!ENTITY People PUBLIC "-//People//people xml 1.0//EN"  
"http://wrox.com/people.xml">
```

- Параметри

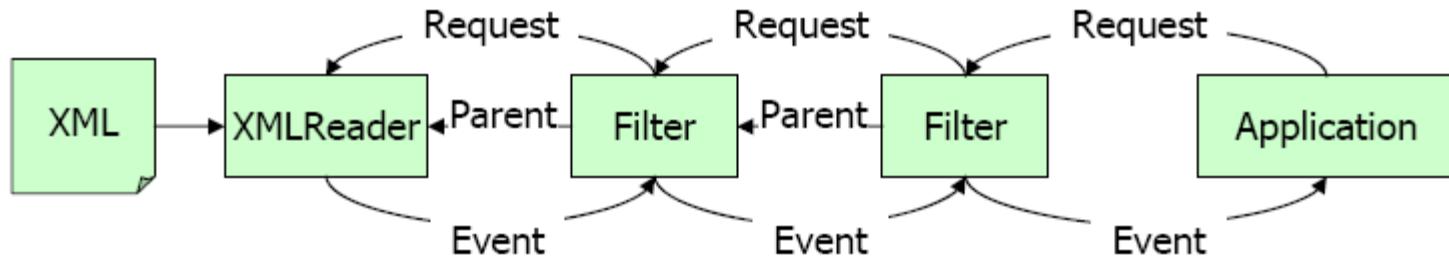
- **publicId**: Публ. идентиф. (//People//people xml 1.0//EN)
 - **systemId**: Системен идентификатор
(http://wrox.com/people.xml)

- Резултат

- **InputSource**: обектът може да се създаде въз основа на системния идентификатор или резултат върнат от база от данни, хеш таблица или каталог за избор на стойност по ключ въз основа на публичния идентификатор

XMLFilter 1/2

- Наследява **XMLReader**
- Работи по събития
- Конвейер от събития:



- Регистриране на предшестващ филтър
 - **setParent(XMLReader)**
- За удобство: **XMLFilterImpl**
 - Имплементира **XMLFilter**, **ContentHandler**, **ErrorHandler**
 - Предава събитията без промяна

XMLFilter 2/2

- Трансформации (при запазване на структурата)
 - Преименование на пространства, елементи, ...
 - Трансформации на стойности на атрибути и др.

```
public class ElementFilter extends XMLFilterImpl {...  
    public ElementFilter(XMLReader parent, String old, String new) {  
        super(parent);  
        ...}  
    public void startElement(... String name ...) {  
        if (name.equals(old))  
            super.startElement(... new ...);  
        else  
            super.startElement(... name ...);  
        ...}
```

Кога да ползваме SAX (Simple API for XML) 1/2

- Типична употреба:
 - Наш нов клас разширява `DefaultHandler`
 - Имплементираме callback методи (напр. `startElement`, ...)
- Обработка при парсването – само последното събитие е в паметта
- За комплексни структури:
 - Нужда от променливи на състоянието
 - Тежка модуларизация

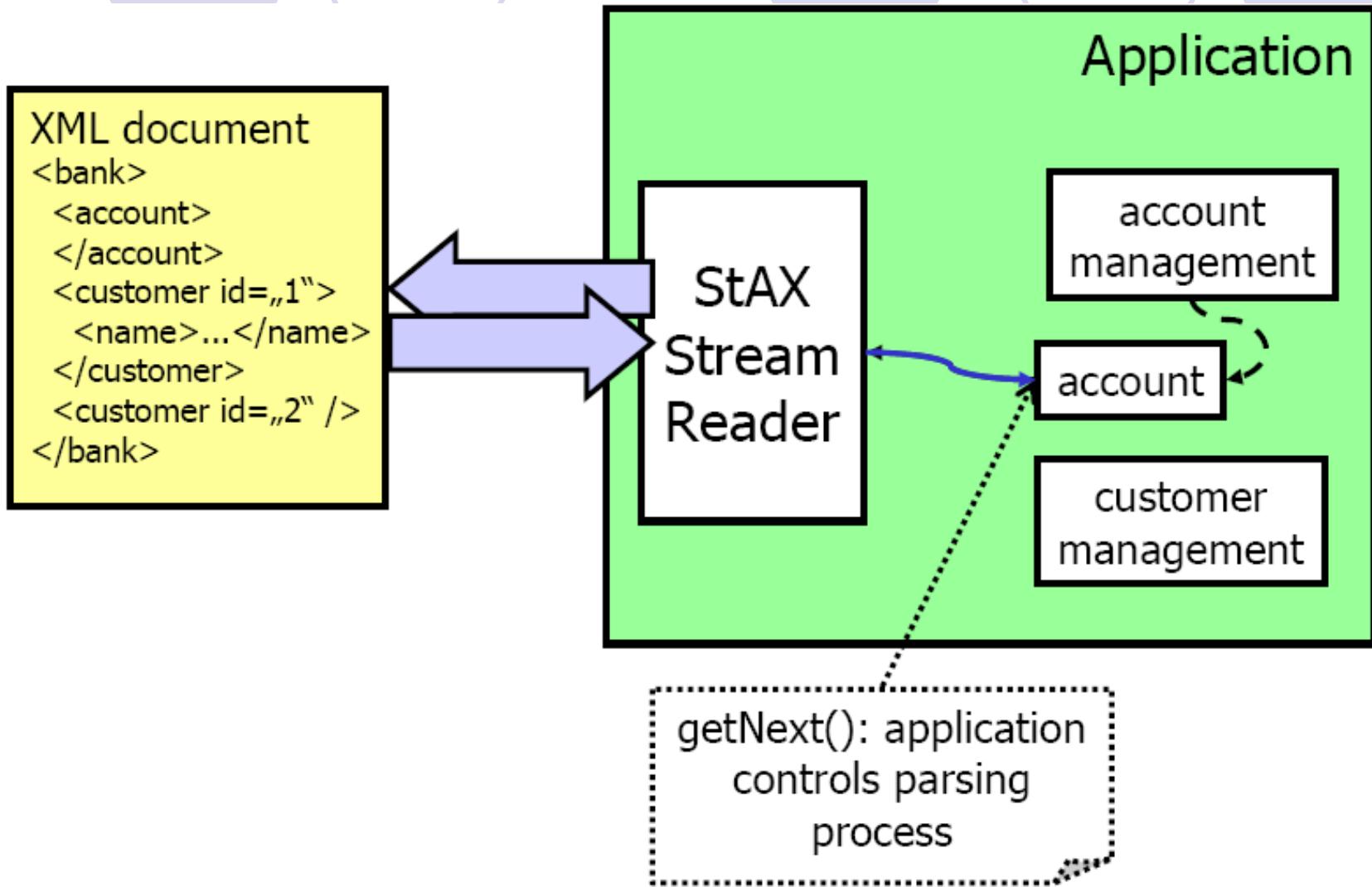
Кога да ползваме SAX (Simple API for XML) 2/2

- Идеален за прости операции над XML файлове
 - Като четене на елементи и атрибути
- Подходящ за много големи XML документи (спрямо DOM)
- Не е удобен за манипулиране на XML структурата
- Не е проектиран за генериране на XML – само за четене!

Алтернатива на SAX - Streaming API for XML (StAX)

- Цел
 - Лесен като DOM
 - Бърз като SAX
 - Икономичен спрямо паметта като SAX
 - API за четене и запис
- Приложението управлява парсера
 - Лесно е за програмиране
 - Достъп до parse-events при нужда (за работа като event driven)
 - Pull-parsing (вместо Push parsing a la SAX)
- StAX SE е част от Java 1.6+ API

Работен процес на StAX



Два приложни StAX интерфейса 1/3

- **Cursor API:** по-директен и ефикасен:

- XMLStreamReader, XMLStreamWriter
- Директен достъп до данните
- Много единични методи
- По-малък и ефективен код, по-добра производителност

- **Iterator API** (конвейерна обработка):
фильтриране, потоци, по-добра поддръжка, по-четим код и модулен дизайн

- XMLEventReader, XMLEventWriter
- Данните се съхраняват в XMLEvent immutable objects
– достъпни при следващото събитие

Два приложни StAX интерфейса 2/3

- **Cursor API:** представлява курсор, с които можем да се разхождаме по XML документ от началото до края. Курсорът сочи една конструкция в даден момент и винаги се движи напред.
- Два основни курсор интерфейси: **XMLStreamReader** и **XMLStreamWriter**.
- XMLStreamReader включва методи за достъп до цялата възможна информация на XML информационния модел, вкл. кодиране на документа, имената на елементите, атрибути, пространства от имена, текст, стартиращи тагове, коментари, инструкции за обработка, граници на документи и др.

Два приложни StAX интерфейса 3/3

- **Iterator API:** представя XML документния поток като набор от дискретни обекти-събития. Тези събития се извличат от приложението чрез парсера в реда, в който се четат във входния XML документ.
- Базовият итераторен интерфейс се нарича **XMLEvent**.
- Основният парсерен интерфейс за четене на събития е **XMLEventReader** и основният интерфейс за писане на събития е **XMLEventWriter**.
- **XMLEventReader** имплементира **java.util.Iterator**

Създаване на StAX XMLStreamReader

- При работа със StAX:
 - `import javax.xml.stream.*`
- Както при SAX и DOM, първо се получава фабрика чрез извикване на статичен метод
 - `XMLInputFactory factory = XMLInputFactory.newInstance();`
- За фабrikата можем да задаваме различни свойства
 - `factory.setProperty("javax.xml.stream.isValidating", "true");`
- XML се подава през `InputStream` или `Reader`
 - `FileReader fileReader = new FileReader("somefile.xml");`
 - Може да изхвърли `FileNotFoundException`
- Едва сега чрез фабриката създаваме `XMLStreamReader`
 - `XMLStreamReader reader =
factory.createXMLStreamReader(fileReader);`
 - Може да изхвърли `XMLStreamException`

Използване на StAX парсер

- StaX парсерът (reader) се държи като `Iterator`
 - Методът `boolean hasNext()` указва дали има друго събитие за четене
 - `int next()` прочита следващото събитие
 - Връщаният резултат `int` задава типа на това събитие
 - Възможни стойности са `START_ELEMENT`, `END_ELEMENT`, `ATTRIBUTE`, `CHARACTERS` (content), `COMMENT`, `SPACE`, `END`
- След `next()`, парсърът е стигнал до „текущ елемент“ и може да бъде разпитван относно него
 - Например, `getLocalName()` връща името на текущия елемент
- Важно: парсерът се движи само напред; лесно можем да пропуснем ценна информация
- По-лесно е да се анализира документа, ако знаем неговата структура
 - Валидиращ парсер може да провери структурата спрямо DTD
 - `isValidating` е свойство на Java StAX парсера

Използване на getLocalName()

- `getLocalName()` връща името на маркера (тага) като `String`
- понеже не можем да ползваме `switch` за `String`, трябва да го сравняваме отделно чрез `equals`:
 - `String name = reader.getLocalName();
if (name.equals(someTag)) { ... }
else if (name.equals(someOtherTag)) { ... }
else if (name.equals(someOtherTag)) { ... }
...
...`

Константи

- **int next()** премества към следващото събитие и връща **int** за указване на типа на събитието:
 - START_DOCUMENT
 - END_DOCUMENT
 - START_ELEMENT
 - END_ELEMENT
 - ATTRIBUTE
 - CHARACTERS
 - COMMENT
 - SPACE
 - DTD
 - PROCESSING_INSTRUCTION
 - NAMESPACE
 - CDATA
 - ENTITY_REFERENCE
- Тези константи са дефинирани в **XMLStreamReader** обекта

Методи

- Съществуват множество методи, дефинирани в `XMLStreamReader`; някои от тях са:
 - `boolean hasNext()` – `true` ако има друго събитие
 - `int next()` – придвижва се до следващото събитие и връща типа му (`int`)
 - `int nextTag()` – придвижва до `start` или `end` маркер и връща типа му
 - `getLocalName()` – взима името на текущия елемент или `entity reference`
 - `getAttributeCount()` – взима броя на атрибутите на текущия елемент
 - `getAttributeLocalName(index)` – името на атрибути
 - `getAttributeValue(index)` – стойността на атрибути
 - `getElementType()` – връща текста на `START_ELEMENT`; след извикване, текущ елемент става `END_ELEMENT`
 - `getText()` – връща текстовата стойност на `CHARACTERS, COMMENT, ENTITY_REFERENCE, CDATA, SPACE, or DTD`

Създаване на StAX writer

- Всичко за StAX е в `javax.xml.stream`
 - `import javax.xml.stream.*`
- Получаваме метод-фабрика:
 - `XMLOutputFactory factory = XMLOutputFactory.newInstance();`
- За запис в XML файл, използваме `OutputStream` или `Writer`
 - `FileWriter fileWriter = new FileWriter("somefile.xml");`
 - Може да изхвърли `IOException`
- Създаваме writer за XML
 - `XMLStreamWriter writer =
factory.createXMLStreamWriter(fileWriter);`
 - Може да изхвърли `XMLStreamException`

Методи

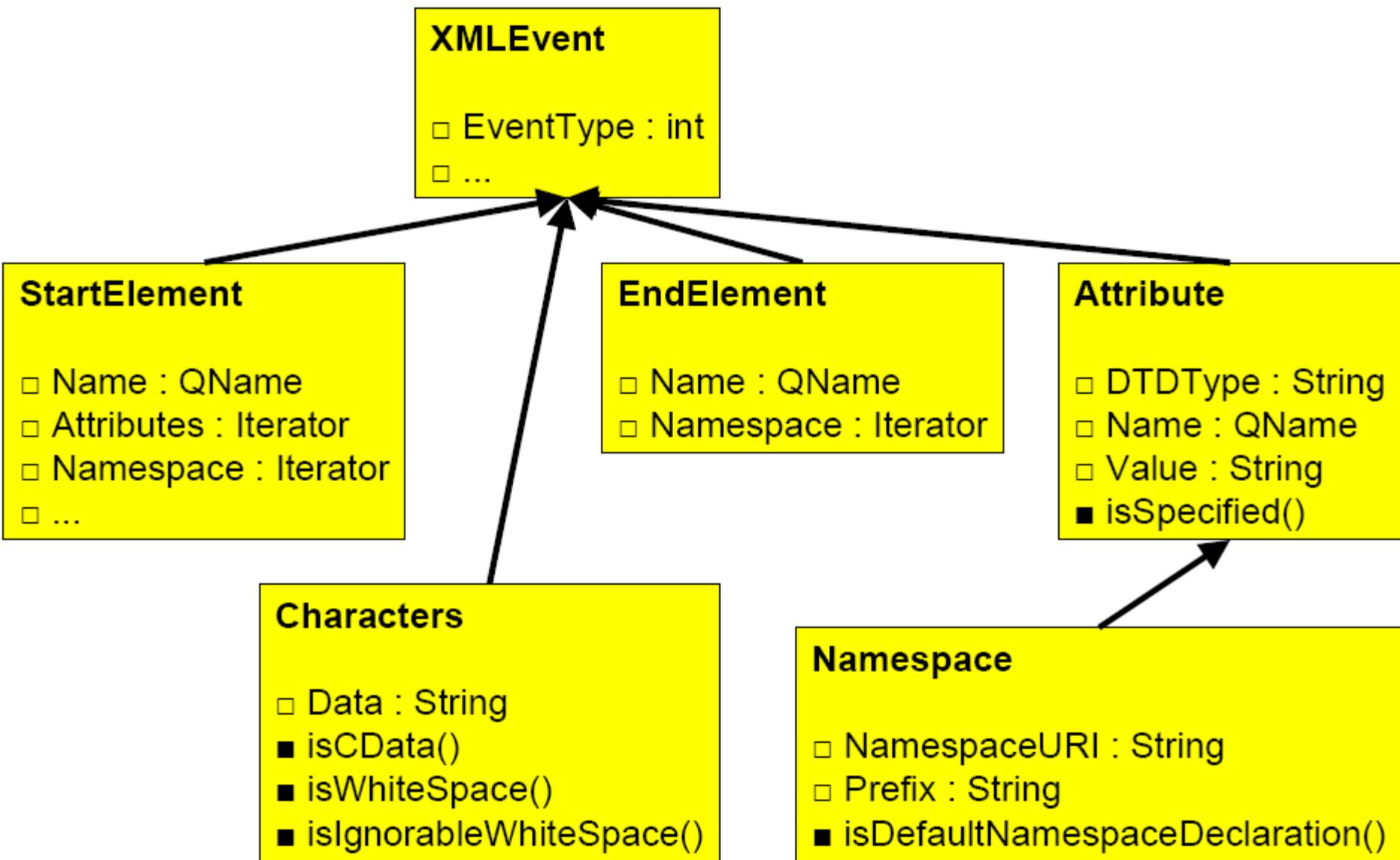
- Дефинирани са множество методи за **XMLStreamWriter**, нап.:
 - `writeStartDocument(version)` – записва XML хедър (оглавление)
 - `writeStartElement(name)` – записва стартов маркер
 - `writeAttribute(name, value)` – записва атрибут
 - `writeCharacters(value)` – записва текст, кодирачи символи като <, > и &
 - `writeComment(value)` – записва коментар
 - `writeDTD(value)` – записва цалата DTD дефиниция
 - `writeEndElement()` – записва краен маркер
 - `flush()` – принуждава записа на буфериран изход
 - `close()` – затваря **XMLStreamWriter**

Преглед на Iterator API – Read

XMLEventReader

- java.util.Iterator:
 - public boolean **hasNext()**;
 - public Object **next()**;
- Следващо събитие с преместване върху него:
 - public XMLEvent **nextEvent()**
throws XMLStreamException;
- Следващо събитие без преместване
 - public XMLEvent **peek()** throws
XMLStreamException;

Йерархия на класа XMLEvent



Преглед на Iterator API – write

XMLEventWriter

- **public void add (XMLEvent e)**
throws XMLStreamException;
- За всички writers
 - **public void flush() throws XMLStreamException;**
- За всички readers и writers
 - **public void close() throws XMLStreamException;**

StAX - обобщение

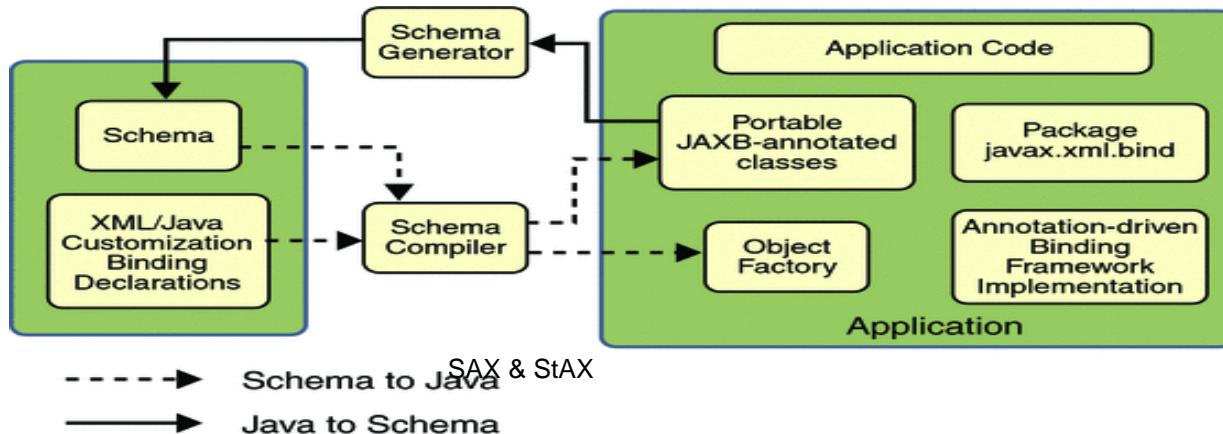
- Предимства на StAX:
 - По-лесен за ползване от SAX, особено поради липсата на callbacks
 - Може да пише в XML файлове, како и да чете от тях
- Недостатъци на StAX
 - Налага ползването на **if-then-else** за разпознаване какво се парсва
 - Както при SAX, движението е само напред
- Сравнение с DOM:
 - StAX е по-бърз, по-ефикасен и по-прост
 - DOM позволява манипулирането на дърво в паметта

JAXB (Java Architecture for XML Binding)

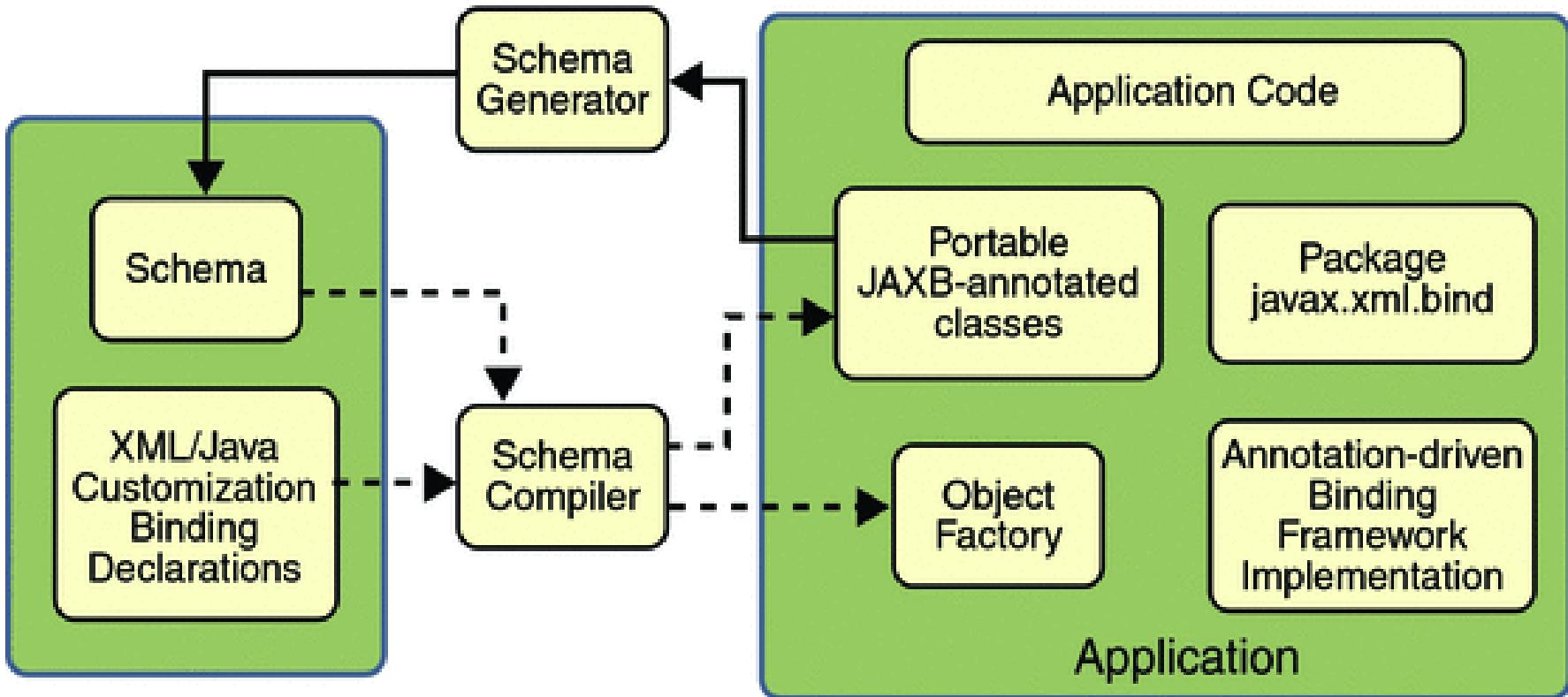
- Java Architecture for XML Binding (JAXB) разрешава на Java проектантите да съпоставят Java класове на XML.
- JAXB адресира два въпроса: възможността да се разполагат (marshal) Java обекти в XML документи и обратното - (unmarshal) XML обратно до Java обекти.
- Т.е., чрез JAXB данни могат да се запазват и извличат от паметта в произволен XML формат без имплементиране на XML зареждане.
- JAXB е особено полезна, когато спецификацията е сложна и променяща се.
- JAXB е API в Java EE платформата и е част от Java Web Services Development Pack (JWSDP).

JAXB

- Schema compiler: Свързва изходна схема към набор от програмни елементи, получени от схемата; свързването се описва от XML-базиран свързващ език.
- Schema generator: Съпоставя набор от съществуващи програмни елементи към производна схема (описва се от програмните анотации).
- Binding runtime framework: Осигурява unmarshalling (XML четене) и marshalling (XML write) операции за достъп, манипулиране и валидиране на XML съдържание.



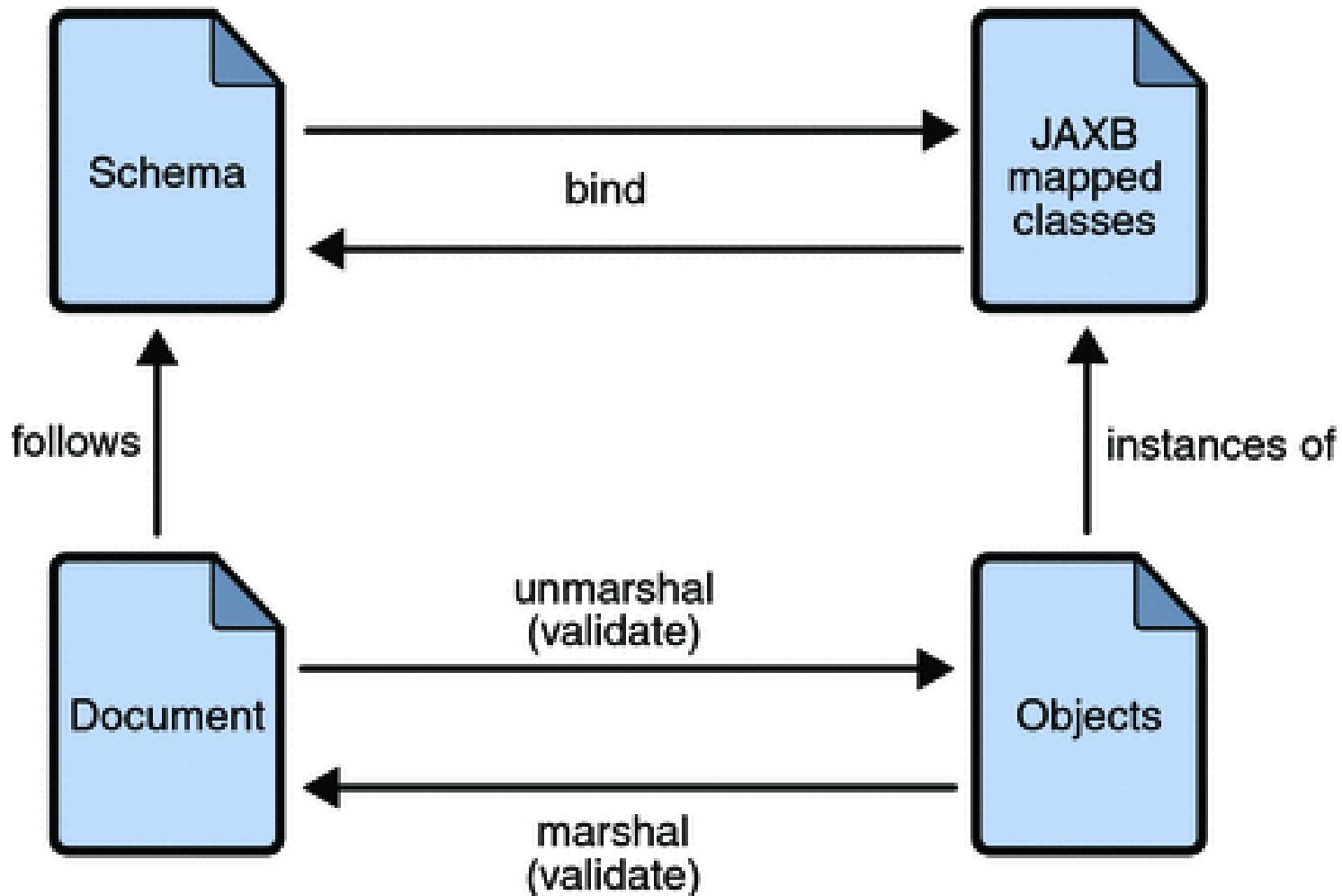
JAXB архитектура



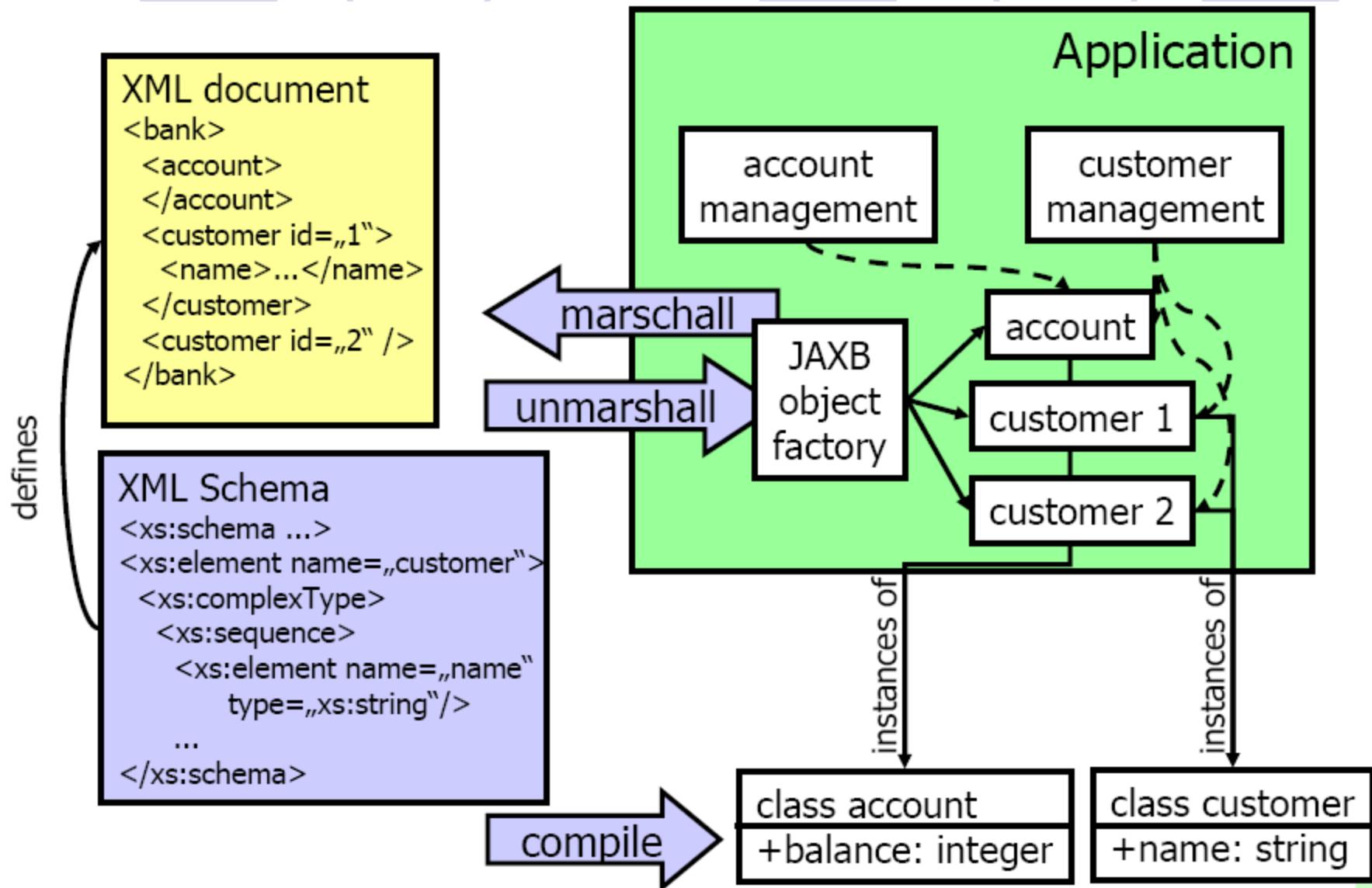
-----> Schema to Java

—————> Java to Schema

Работен процес на JAXB - Java Architecture for XML Binding (JAXB) 1/2



Работен процес на JAXB - Java Architecture for XML Binding 2/2



Повече за JAXB

- [*https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html*](https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html)
- [*http://www.oracle.com/technetwork/articles/javase/index-140168.html*](http://www.oracle.com/technetwork/articles/javase/index-140168.html)

DOM спрямo StAX

DOM	StAX
whole document in memory (tree)	only recent part (event) in memory
first read whole document	read and process data at one time
free navigation in tree	serial processing
high memory usage	low memory usage
computing intensive	high throughput
maximum flexibility	processing of well-known data structures

SAX спрямo StAX

SAX	StAX
Parser controls flow	Application controls flow
Parser sends data, whether applicator is ready or not	Application asks explicitly for data
Push parsing	Pull parsing
low memory usage	low memory usage
only read	read and write

Заключение: StAX, SAX и DOM

	StAX	SAX	DOM
API Type	pull	push	tree
Usage	easy	complex	easy
XPath-Support	no	no	yes
Efficiency (Memory, CPU)	good	good	bad
only forward parsing	yes	yes	no
XML read	yes	yes	yes
XML write	yes	no	yes
create, modify, delete	no	no	yes

Свързване и обединение на XML съдържание – XLink, Xpointer и XInclude

Същност

Типове

Елементи

Обхождания

Примери

Въведение

- Доброто структуриране на XML документа предполага по-високо ниво на грануларност на информацията.
- Това води до необходимост от свързване на едни секции на документа с други.
- Секции от съдържанието трябва да могат да реферират към съдържание във външни документи по начин, по-ефективен от хипервръзките в HTML.
- Ограничения при използването на традиционните HTML хипервръзки – напр. задължителната посока на връзката от съдържащия я документ към външен ресурс или фрагмент в документа.
- Това води до развитие на стандарти за свързване и обединение на XML съдържание.

Три стандарта за свързване и обединение на XML документи

- **XLink** – език за дефиниране и описание на различни видове хипервръзки между Уеб ресурси, в частност XML документи. Възприет е като стандарт през 2001 година.
- **XPointer** – език за дефиниране на хиперлинкове в XML документ, които обаче реферират различни фрагменти от XML документа. XPointer дефинира фрагментни идентификатори за URI референции, сочещи към възли или части от тях в XML ресурси. Стандарт от 2002 година.
- **XInclude** - общ механизъм за сливане на XML документи, посредством задаването на маркери (тагове) в главния документ за автоматично включване на други документи или части от тях. Стандарт от 2006 г. насам.

Xlink, XPointer и XInclude

- Xlink, XPointer и XInclude използват:
 - **XPath** за адресация и навигация в XML документи
 - стандарта **XML Base** (<http://www.w3.org/TR/xmlbase/>), който дефинира атрибута **xml:base** за задаване на база за относителни URI връзки към външни за документа ресурси, подобно на елемента HTML BASE.
- XPointer е предназначен за използване с XML Link Language, или съкратено XLink. Използва относително ограничено в световен мащаб.
- Както XML Schema и XPath, така и XLink и XPointer са **спомагателни езици (XML accessories)**, които разширяват свойствата на езика XML. Специално за XLink и XPointer, разширението е в посока дефиниране на различни видове хипервръзки между документи и в самия документ.
- XML Inclusions (XInclude) е по-известен и въвежда общ механизъм за сливане на XML документи за използване от софтуерни приложения. XInclude дефинира език за конвертиране на входни XML данни в друго крайно представяне. Ето защо XInclude, заедно с CSS и XSL, се отнася към семейството на **XML преобразувателите (XML transducers)**.

Какво е XLink?

- XLink е предназначен за описание и създаване на връзки между Уеб ресурси и е специално разширение на XML за хипермедиа.
- XLink преодолява недостатъците на хипервръзките между HTML документи.
- Съкращение от: **XML Linking Language (XLink)**
- Версия: 1.0 (*by W3C XML Linking Working Group*)
- Спецификация: <http://www.w3.org/TR/xlink/>
- Namespace:
`xmlns:xlink="“http://www.w3.org/1999/xlink”"`

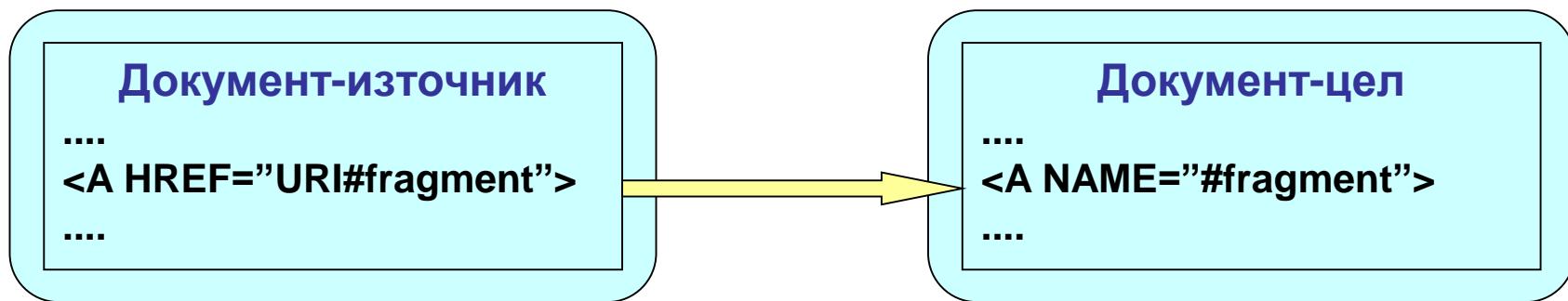
HTML хипервръзка 1/2

В HTML, връзката се изгражда на базата на елемента за котва с маркер **<A>** и се разпознават по това име, което не може да послужи за универсален механизъм за свързване в XLink. Това е така, защото котвата може да се използва по два начина:

- За да създадем връзка в един документ (източник) към друг (цел), чрез използване на атрибута HREF;
- За създаване на показалец (*bookmark*) в документа-цел с помощта на име атрибута NAME (ако целта не е целият документ-цел, а фрагмент от него).

HTML хипервръзка 2/2

- Така реално елементът HTML котва не представлява самата връзка, макар че е наричан с това име. За една хипервръзка в общия случай трябва да се създадат две котви, и то на две различни места, като дефиницията на връзката трябва да е непременно в документа-източник.



Абстрактна семантика на връзката

- XLink задава на езиково ниво абстрактна семантика на връзката, като източникът и целта се задават относително, а не абсолютно.
- Както хипервръзките в HTML, така и в XLink може да се дефинират прости и еднопосочни връзки.
- Освен това: ->

XLink може да се използва за:

- дефиниране на връзки между повече от два ресурси – целта може да е повече от една;
- дефиниране на връзки не само в документа източник, но и в документа цел;
- задаване както на вътрешни връзки (вградени в документите), така и на дефинирани на външно място, отделно от свързаните ресурси – използва се за случаите, когато не е възможно да се дефинират връзки в ресурси като графика, видео и аудио;
- свързане на метаданни с XLink връзки.

XLink концепции 1/2

- Според спецификацията (IETF RFC 2396), **XLink връзката** се посочва изрично от XLink елемент на свързване (*linking element*), който утвърждава съществуването ѝ.
- Една връзка (линк) е елемент и асоциира явно набор от ресурси или части от ресурси, за които се казва, че участват в тази връзка.
- Ресурс** е всяка адресируема единица информация или услуга - файл или фрагмент от него, изображения, програми и резултати от заявка. Биват:
- Локални (local) ресурси** – вътрешни за документа, представляващи елемент на свързване на дадена връзка или неговия родител;
- Отдалечени (remote)** – представляват както външни за документа ресурси, така и вътрешни ресурси (напр. елементи), достъпни по¹⁰URI.

XLink концепции 2/2

- Идентификаторът на отдалечен източник, участващ във връзката, се нарича **локатор** (*locator*).
- Използването на връзка за минаване по нея за каквато и да е цел се нарича преминаване или **траверс** (*traversal*). Източникът, от който е започнал траперсът, се нарича **начален ресурс**. Дестинацията на преминаването през връзката е **краен ресурс**.
- Информацията за това как се преминава през двойка ресурси, включително и посоката на преминаване и евентуална информация за поведение при преминаването, се нарича **дъга** (*arc*).

Три типа дъги

за създаване на споделени връзки в бази от връзки (*linkbases*).

за свързване на бинарни ресурси

- Дъги, имащи локален начален ресурс и отдалечен краен ресурс, се наричат **изходящи** (*outbound*). Пример за връзка с такъв тип дъга, имаща за източник локален ресурс, е HTML елементът за котва с маркер <A>;
- Дъги, имащи отдалечен начален ресурс и локален краен ресурс, се наричат **входящи** (*inbound*) – в HTML не е възможно да се дефинират връзки с такива дъги, имащи за цел локален ресурс;
- Дъги, за които нито началният, нито крайният ресурс са локални, се наричат **дъги за трета страна** (*third-party*).

Два типа връзки

- **Прости (*simple*)** – притежават само една дъга, с локален начален ресурс и отдалечен краен ресурс. Очевидно тази дъга е от тип изходяща (*outbound*). HTML хипервръзките са прости;
- **Разширени (*extended*)** – могат да свързват произволен брой ресурси, които да бъдат каквато и да е комбинация от отдалечени и локални ресурси. В резултат на това тяхната структура може да бъде доста сложна, включително елементи за посочване на отдалечени ресурси, елементи с локалните ресурси, елементи за определяне на правилата за travерс на дъга, и елементи за определяне на етикети на ресурси и дъги.

Макар и простите връзки да представляват концептуално под клас на разширените връзки, те са с различен синтаксис и не могат лесно да се конвертират до разширени връзки.¹³

Прости XLink връзки 1/2

```
<?xml version="1.0"?>  
<catalog xml:base="http://samples.org/"  
         xmlns:xlink="http://www.w3.org/1999/xlink">  
  
<head>  
    <title>Virtual Catalog</title>  
</head>  
  
<body>  
    <p>See <myLink xlink:type="simple"  
          xlink:href="products.xml">  
        Our products</myLink>!</p>  
    <li xml:base="/newproducts/">
```

за интерпретиране
на относителните
адреси, зададени в
xlink:href атрибути

Прости XLink връзки 2/2

```
<item>
  <myLink xlink:type="simple" xlink:href="p1.xml">
    New product 1</myLink>
  </item>
  <item>
    <myLink xlink:type="simple" xlink:href="p2.xml">
      New product 2</myLink>
    </item>
  </li>
</body>
</catalog>
```

Важно:

XLink не дефинира елементи, а само атрибути и начина на използването им от съдържащите ги елементи.

Атрибути на връзката 1/3

- **href** – URI на ресурс;
- **type** – глобален атрибут от XLink namespace, задаващ тип на връзката; възможни стойности са: “*arc*”, “*extended*”, “*locator*”, “*resource*”, “*simple*”, “*title*” или “*none*”;
- **role** – URI на ресурс, описва ролята на елемента;
- **arcrole** – URI на ресурс, описва ролята на връзката;
- **title** – удобно за потребителя заглавие на елемента;
- **show** – описва поведението на представяне на **href** ресурса при travерс на връзката, т.е. как приложението трябва да представи крайния ресурс на приста връзка (същият атрибут се използва и при дъги) →

Атрибути на връзката 2/3

- **show** – описва поведението на представяне на `href` ресурса при travерс на връзката, с възможни стойности:
 - “**new**” - съдържанието на документа ще се покаже в нов прозорец, подобно на HTML атрибута `TARGET=_blank` на елемента котва;
 - “**replace**” – съдържанието на крайния ресурс ще замени това на началния документ в същия прозорец, подобно на HTML атрибута `TARGET=_self` на елемента котва;
 - “**embed**” – съдържанието на крайния ресурс ще се вгради в същия прозорец заедно с това на началния ресурс, както алтернативно съдържание се представя в HTML с атрибута `ALT` на елемента `IMG`;
 - “**other**” – няма ограничения върху поведението на приложението при travерс на връзката, но то трябва да потърси друго маркиране в елемента на свързване за определяне на поведение;
 - “**none**” - няма никакви ограничения върху поведението на приложението при travерс на връзката.

Атрибути на връзката 3/3

- **actuate** – описва поведението на съществяване на travерс към **href** ресурс; може да има стойностите:
 - “**onLoad**” – приложението трябва да премине през връзката веднага щом зареди началния ресурс;
 - “**onRequest**” – travерс на връзката се извършва при външно събитие, напр. щракване с мишката върху нея или изтичане на времето за изчакване на редирект;
 - “**other**” и “**none**” - същите както за атрибута **show**.

Елемент, задаващ проста XLink връзка

```
XML Document
<doc ...
  xmlns:xlink=".../xlink">
...
<mytag
  xlink:type="simple"
  xlink:role="role_uri"
  xlink:href="ref_uri"
  xlink:show="new"
  xlink:actuate="onLoad">
...
</mytag>
...
</doc>
```

Дефиниция на роля

Рефериран ресурс

Инструкция за отваряне на връзката в нов прозорец при преминаване през нея

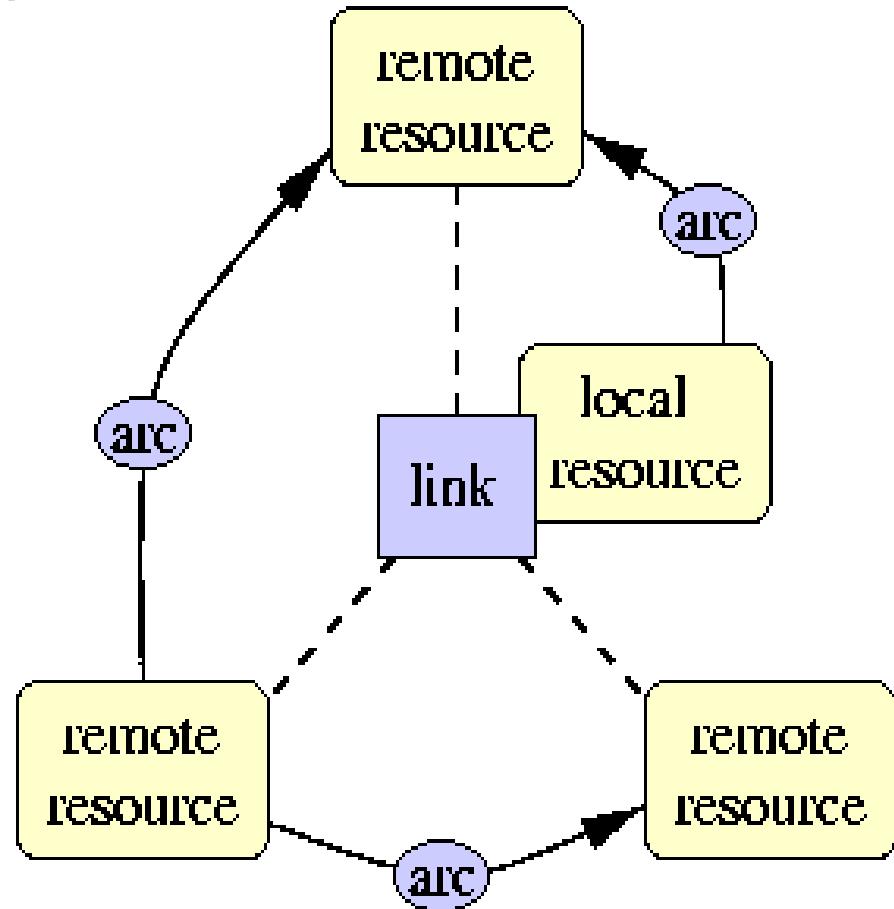
Инструкция за преминаване през връзката при зареждане на документа

Разширена XLink връзка 1/2

- асоциира произволен брой ресурси от отдалечен и/или локален тип;
- тя е единственият тип връзка, който може да има входящи дъги;
- разширените XLink връзки може да се съхраняват във външен документ – т.нар. база от връзки (*linkbase*), което води до по-лесна поддръжка и решава проблема с връзките към документи, които не могат да се модифицират (например с бинарно съдържание).

Разширена XLink връзка 2/2

- Разширените връзки могат да съдържат дъги от различен тип;
- Пунктирани линии представят асоциирането на ресурсите с общия свързващ елемент на връзката, който е определен от стандарта Xlink не като име на елемент, а чрез съдържаните от него атрибути.



Елемент с разширена връзка 1/2

Освен атрибути, елементът на разширена връзка в XLink може да съдържа и елементи от следните типове:

- **локатор** тип елементи, които адресират отдалечени ресурси, участващи в линка - задават се посредством атрибута **type="locator"** и представлят отдалечен за връзката ресурс. Те могат да имат и следните атрибути:
 - **href** - URI на ресурса (задължителен)
 - **role** - URI на ресурс, който описва ролята на элемента
 - **title** - описание на елемента за четене от потребители
 - **label** - осигурява етикет, към който елемент от тип дъга може да реферира
- елементи от тип **ресурс**, които съдържат локалните ресурси, които участват в линка - задават се посредством атрибута **type="resource"** и представлят локален за връзката ресурс.

Елемент с разширена връзка 2/2

- елементи от тип дъга, които осигуряват правила за преминаването между участващите ресурси на връзката - посредством атрибута **type="arc"**, описват връзките между ресурси. Дъгите могат да бъдат изходяща (*outbound*), входяща (*inbound*) и отдалечена (*remote*) – между отдалечени ресурси. Такъв елемент може да ползва атрибутите **show**, **actuate** и **title** със значение, идентично с това атрибутите на елемент, описващ пристъпка. Други възможни атрибути на една дъга може да са:
 - **from** - определя етикета на изходните за траверса ресурси
 - **to** - задава етикета за крайните за траверса ресурси
 - **arcrole** - URI на ресурс, който описва ролята на дъгата
- елементи от тип заглавие - задават се посредством атрибута **type="title"** и задава по-подробно описание от атрибута **title**, напр. при нужда от вложени описания.

Attr.\Type value	sim- ple	extended	locator	arc	resource	title
type	R	R	R	R	R	R
href	O		R			
role	O	O	O		O	
arcrole	O			O		
title	O	O	O	O	O	
show	O			O		
actuate	O			O		
label			O		O	
from				O		
to				O		

XLink атрибути

Типове на елемент
(по колони) и
глобални атрибути
за тях (по редове):

- **required (R)**
- **optional (O)**

Пример за разширена връзка

- Следващият пример описва разширена връзка с елемент на свързване между 4 ресурса (три ваканционни почивки и един адрес на физическо лице) и два локатора с адреси на система за банков трансфер през Уеб – един на физическото лице и един на туристически оператор, предлагащ почивките.
- Дъгата **charge_to** задава информация за travesc, реализиращ банков трансфер от сметката на лицето към сметката на оператора (номерата на сметките са указаны в URL), а дъгата **deliver_to** задава изпращането на ваучера за почивката.

```
<?xml version="1.0"?>
<purchase xmlns:xlink="http://www.w3.org/1999/xlink">
  <order xlink:type="extended" >
    <holiday xlink:type="resource" xlink:label="h1">
      Caribbean Holiday</holiday>
      <holiday xlink:type="resource" xlink:label="h2">
        Greek Holiday</holiday>
        <holiday xlink:type="resource" xlink:label="h3">
          Bulgarian Holiday</holiday>
          <account xlink:type="locator" xlink:label="a1"
xlink:href="http://www.money4nothing.com/cgi/
investbank?bankNo='12-345-67890"
xlink:title="Ivan Ivanov"/>
          <account xlink:type="locator" xlink:label="b1"
xlink:href= "http://www.money4nothing.com/cgi/
delawerebank?bankNo='09-876-54321"
xlink:title="Book Your Travel ASAP"/>
```

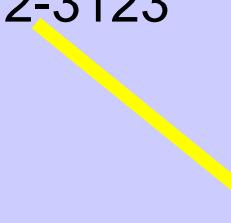
....

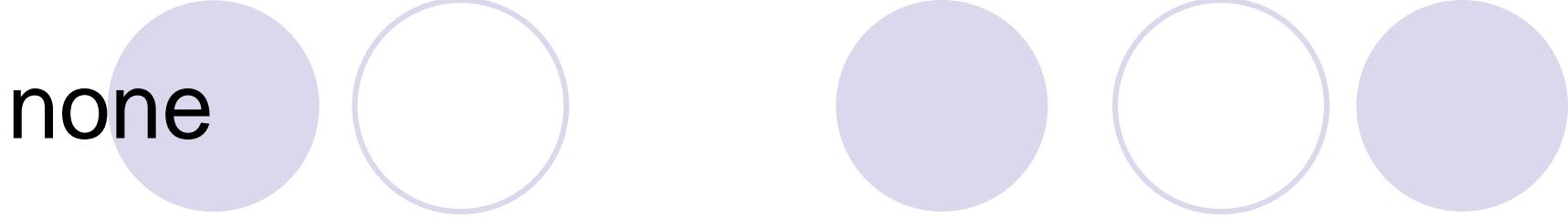
```
<location xlink:type="resource" xlink:label="d1">
    <attn>Ivan Ivanov</attn>
    <street>123, Dolno blato</street>
    <place>Sofia</place>
    <zip>1234</zip>
</location >
<charge_to xlink:type="arc" xlink:from="a1" xlink:to="b1"
xlink:title="Ivan pays for holiday by a bank transfer"/>
<deliver_to xlink:type="arc" xlink:from="h2" xlink:to="d1"
xlink:title="Ivan gets holiday voucher at his home"/>
</order>
</purchase>
```

Друг пример

```
<doc xmlns:xlink=".../xlink">
<order xlink:type="extended" >
  <car xlink:type="resource" xlink:label="c1">Porsche </car>
  <car xlink:type="resource" xlink:label="c1">Ferrari </car>
  <account xlink:type="locator" xlink:label="a1"
    xlink:href="http://www.chargeme.com/cgi/citibank-12-3123"
    xlink:title="Brian Temple"/>
  <location xlink:type="resource" xlink:label="d1">
    <attn>Matthias Hauswirth</attn>
    <street>...</street> ... <zip>80303</zip>
  </location >
  <charge_to xlink:type="arc" xlink:from="c1" xlink:to="a1"
    xlink:title="Brian pays ☺"/>
  <deliver_to xlink:type="arc" xlink:from="c1" xlink:to="d1"
    xlink:title="Matthias gets ☺"/>
</order>
</doc>
```

Bank
Account





none

- Елемент с атрибут XLink *type*=“*none*” няма определено значение.
- Използва се за включване и изключване на XLink в отделни елементи.

none пример

DTD

```
...
<!ELEMENT ref (stuff)>
<!ATTLIST ref
  xlink:type (simple|none) "none"
  xlink:href CDATA #IMPLIED >
```

“ref” елемент с връзка

Markup

```
...
<ref xlink:type="simple" xlink:href="http://x.com">
  <stuff>stuff</stuff>
</ref>
...
<ref>
  <stuff>stuff</stuff>
</ref>

```

“ref” елемент без връзка

XML

Какво е XPointer?

- XPointer е спомагателен език, който не е базиран на XML и служи за определяне на местоположения (локации) във вътрешността на XML документи
- Създаден с цел да бъде използван заедно с XLink
- XPointer изразите се добавят в края на URI като идентификатор за фрагмент, за да посочват конкретна част от XML съдържанието, а не целия документ
- XPointer Scheme - W3C Working Draft 19 December 2002
- Спецификация: <http://www.w3.org/TR/xptr-xpointer/>

XPointer като разширение на XPath

- XPointer синтаксисът се основава на синтаксиса на Xpath.
- Той използва четирите фундаментални типа данни на XPath (булев, множество от възли, число и низ), като към тях добавя два нови типа: точка (*point*) и обхват или зона (*range*), както и функции за работа с тези типове.
- XPointer добавя също и някои полезни съкращения за често използвани форми на изрази на XPath.
- XPointer представлява разширение на XPath, предназначено за свързване на секции от документ, като задава връзка между XPath изрази и URI връзки.

Проблем на традиционните хипервръзки в HTML

- В HTML документ, връзката се задава към именуван ресурс, напр. елемент-котва или IMAGE;
- за всяка връзка трябва да се поставя котва;
- адресирането е статично - чрез фиксиран идентификатор на фрагмент, което означава, че при промяна на името на фрагмента трябва да се промени и името във връзката;
- могат да бъдат свързани само отделни възли на документа, но не и техни части.

XPointer преодолява тези ограничения

- посредством разширяване на езика XPath, предназначено за употреба на XPath изрази в URI адреси
- чрез езика е възможно свързването да става динамично, на базата на изчисляване на XPath изрази, като се използва адресиране по дефинираните от XPath оси
- това прави тази технология гъвкава и независима от промени в адресирания документ.
- Освен това, езикът предоставя големи възможности за адресиране на части от съдържанието на елемента

Xpointer пример

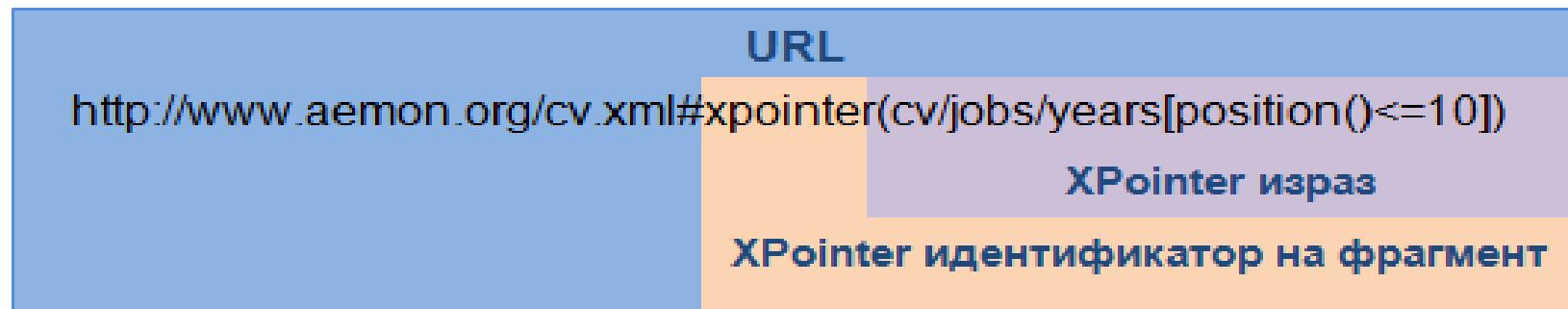
```
#xpointer(/chapter[3]/elem[@name="foo"])
```

Адресира:

```
<chapter>
    <elem name="foo"> ...<elem>
</chapter>
```

Xpointer изрази

- Чрез използването на XPointer в URL се дефинират XML указатели, които се използват за идентифициране на позиции и фрагменти в XML ресурси.
- Това става на база на замяната на опционалния идентификатор за фрагмент с XPointer израз.
- Примерът адресира чрез URL един XML документ, представляващ автобиография, като от документа се извличат първите 10 елемента с име **years**, вложени в йерархията **cv/jobs/years**.



XPointer в браузер?

XPointer би могъл да се ползва за показване на HTML съдържание, стига браузерите да поддържат езика, т.е. да интерпретират XPointer връзки. Тогава при задаване на извеждане на първия елемент с име **product**, намерен в съдържанието на документа, браузерът ще зареди целия документ и ще трябва да покаже въпросния елемент от него.

```
<a href =  
"http://www.sth.org/cat.xml#xpointer("//product[1])">  
    The name of a product from the catalog.  
</a>
```

Засега поддръжката на XPointer в браузери е ограничена.

Важно:

За разлика от традиционното адресиране, чрез XPointer можем да реферираме към повече от един елемента.

Последователност от XPointer изрази

- Възможно е също така за бъде зададена последователност от XPointer изрази без задаване на разделители (празните пространства между разделителите са допустими), във вида

URL#xpointer(израз)xpointer(израз)...xpointer(израз)

- При последователност от XPointer изрази, най-напред се изчислява първият израз и ако се върне фрагмент от адресирания документ, то изчислението спира. В противен случай, изразите се изчисляват до връщането на фрагмент или до последния израз.

XPointer израз в XLink връзка

- задаване на XPointer израз в приста XLink връзка към първият срещнат елемент с име **task**, който е наследник на елемента-корен **howto** в документа **manual.xml**:

```
<beforeStarting xlink:type="simple"
  xlink:href="manual.xml#xpointer(/howto/task[position()=1])">
  To be read before starting!
</beforeStarting>
```

- XPointer допуска използване на пространства от имена в изразите, като за целта префиксът на пространството трябва да се зададе предварително. Например

xmlns(html=http://www.w3.org/1999/xhtml) xpointer("//html:div[7])

дефинира седмият намерен по ред елемент div от пространството с префикс **html** и адрес равен на **http://www.w3.org/1999/xhtml**.

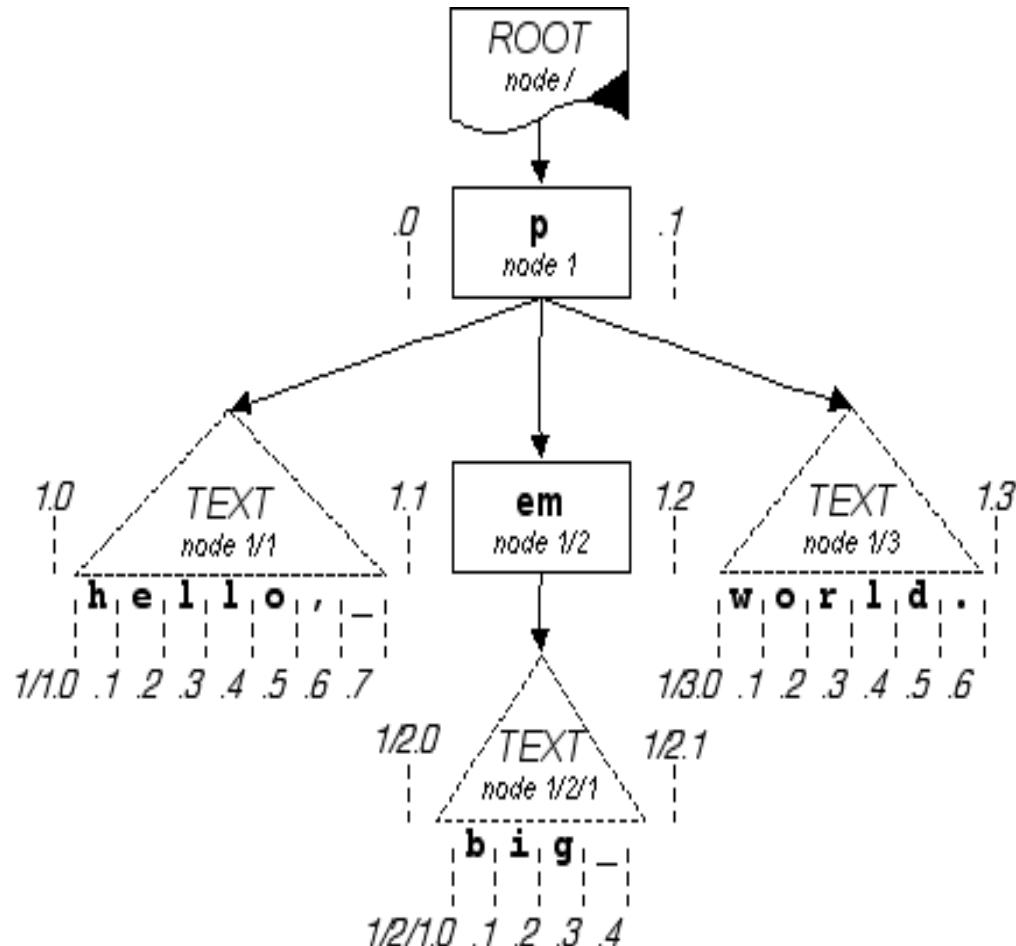
XPointer точка (point) и обхват (range)

точката (*point*) е позиция в XML съдържание

зона или обхват (*range*) - XML съдържанието между двойка крайни точки

Примери:

- point(1.0) is just inside the beginning of the first TEXT element.
- point(1.2) is between the end of the em element and the following text node (which contains "world.").
- point(.0) immediately precedes the root node.
- point(1/2/1.1) immediately follows the "b" in the middle text node.



XML Inclusions (Xinclude)

- XML Inclusions, или съкратено Xinclude, е универсален механизъм с общо предназначение за изграждане на модулност в XML.
- Спецификацията на XInclude е препоръка на W3C от 2004 г., с втора версия от 2006 г.
- Тя задава XML базиран синтаксис на общ механизъм за сливане на XML документи, посредством задаването на маркери в главния документ за автоматично включване на други документи или части от тях.
- XInclude дава по-добро решение на проблема за конструирането на XML документ от множество документи в сравнение с други подходи като например използването на външни XML единици (external entities).

Проблеми на външните XML единици

- външната XML единица не може да бъде пълноправен, самостоятелен и независим XML документ, понеже в нея не се разрешава задаването на XML декларация или използване на DOCTYPE.
- външната XML единица не може да включва други външни единици
- външната XML единица трябва задължително да бъдат добре конструиран XML - това забранява включването на други формати, напр. на код в XML документ
- грешката при зареждане на външна единица е фатална грешка и възстановяването е строго забранено
- може да бъде включена само цялата външна единица, но не и само част от документа.
- външните единици трябва да бъдат декларирани в DTD документ, което усложнява валидацията.

XInclude и XLink

- XLink не уточнява конкретен модел за обработка, а само улеснява откриването на връзки и разпознаването на свързани метаданни от софтуерните приложения.
- За разлика от него, XInclude задава специфично XML в XML преобразуване. Той определя конкретен модел на обработка за обединяване (сливане) на информационни множества.
- XInclude обработка се извършва на най-ниско ниво от XInclude процесор, който подава резултатния за сливането документ към други приложения. При това сливане, XInclude не определя механизъм за DTD валидация на резултата

Синтаксис на Xinclude – езикът задава само два елемента

- Те принадлежат на пространството от имена <http://www.w3.org/2003/XInclude> (с префикс "xi,,) и се наричат **include** и **fallback**.
- Спецификацията предоставя XML Schema и DTD, описващи XInclude.

Елемент **xi:include**

Елементът **xi:include** е маркираща инструкция на включване на външен документ, като определя кой документ да се включи и как. Има атрибути:

- **href** - URI референция към документа за включване
- **parse** - може да заема стойностите "xml" (по подразбиране) или "text" в зависимост от това дали се включва XML съдържание или текст без маркиране.
- **xpointer** - опционален XPointer израз, задаващ порция от XML документа за включване; игнорира се при включване на немаркиран текст, т.е. за `parse="text"`
- **encoding** - указва кодирането на включвания документ; при включване на XML съдържание (`parse="xml"`) този атрибут се игнорира
- **accept, accept-charset и accept-language** - тези атрибути се използват при договарянето на съдържанието в HTTP (*HTTP content negotiation*). Стойностите им се извличат от XInclude процесора и се задават в хедърите на HTTP заявката с имена Accept, Accept-Charset и Accept-Language.

Използват се за случаите, когато един URI връща различни езикови или кодови версии на един и същ ресурс.

Елемент **xi:fallback**

- При сливането на документи нищо не гарантира, че включваните един в друг документи ще са налични по време на обработката от процесора.
- В такъв случай, елементът **xi:fallback** може да се използва като механизъм за възстановяване при липсващ ресурс. Следващият пример показва как се ползва елемента **xi:fallback**. Самият той може да съдържа нов **xi:include** element със свой **xi:fallback** за възстановяване, и т.н.

Пример

```
<page xmlns:xi="http://www.w3.org/2003/XInclude">
  <header>News about the ADOPTA project</header>
  <xi:include href="http://www.adopta.eu/news.xml">
    <xi:fallback>
      <xi:include href="http://62.44.125.62/tmp/news.xml">
        <xi:fallback> Sorry, the ADOPTA server is not
available just now. </xi:fallback>
      </xi:include>
    <xi:fallback>
      </xi:include>
  </page>
```

XInclude процес на XML трансформация

- Процесът включва замяната на всеки един xi:include елемент с документа, към който той реферира.
- Включването е рекурсивно, като процесорът изгражда дърво на сливанията на база на следване на xi:include елементите във всеки нов включван документ до стигането на документ без xi:include елементи.
- Поради рекурсивната обработка, цикличните включвания се откриват и третират като фатални грешки, при които обработването трябва да се спре и цикличното включване да се избегне, напр.:
 - ако елемент **xi:include** указва връзка към самия себе си, когато **parse="xml"**;
 - ако елемент **xi:include** указва връзка към елемент или негов наследник, който вече е бил включен на по-високо ниво.

parse="text"

- При опит да се включват недобре конструирани XML документи се стига до фатална грешка. При **parse="text"** съответният документ се третира като обикновен текст, дори и да е в XML формат, напр.:

```
<doc xmlns:xi="http://www.w3.org/2003/XInclude">  
    A trial with inclusion of XML document as plain text.  
    <xi:include href="sample.xml" parse="text"/>  
</doc>
```

- Това позволява включването на код или на XML съдържание като PCDATA. За целта обаче процесорът заменя началния разделител за маркер с екранната последователност <

Заключение

- XInclude е W3C стандарт за изграждане на голем XML документите от малки, лесно поддържани XML документи или съдържание с друг формат.
- Ако се включваното съдържание е в XML формат, то трябва да е добре оформлено (конструирано), но не е задължително да може да се валидира.
- Както XPointer, така и XInclude да се използват заедно с XLinks. И трите езика използват стандарта XML Base за задаване на базов URL за адресираните и включваните документи, съдържащи относителни адреси.
- За съжаление все още тези стандарти не са широко използвани и поддръжката им от много от Уеб браузърите е ограничена.

Допълнителни материали

[CARR] Leslie Carr, Initial Experiences of an XLink Implementation;
<http://journals.ecs.soton.ac.uk/xml4j/xlinkexperience.html>

[JIRAT] Jiri Jirat, XLink Reference;
http://zvon.org/xxl/xlink/Output/xlink_refs.html

[MALER] Eve Maler, XLink and Xpointer Overview; <http://www.oasis-open.org/cover/xlinkMaler980402.html>

[ST LAURENT] Simon St.Laurent, XLinkFilter: An Open Source Java XLink SAX Parser Filter;
<http://www.simonstl.com/projects/xlinkfilter/index.html>

[vdVLIST] Eric van der Vlist, XML Linking Technologies;
<http://www.xml.com/pub/2000/10/04/linking/index.html>

[XMLCOM] XML Resource Guide: XLink;
<http://www.xml.com/pub/Guide/XLink>

Cascading Style Sheets (CSS) версии 1, 2 и 3

Стилово форматиране и представяне
Кутиен модел
Позициониране
Примери

История
Цели на CSS
Свойства

Основни характеристики на документ

Съдържание

- Информация, съдържаща текст, графика, аудио или видео

Структура

- Подялба и последователност на информационните части

Оформление

- Онагледяване на съдържанието и структурата на документ

Метаданни

- Описание семантиката на съдържанието

Каскадни стилове (Cascading Style Sheets)

- Cascading Style Sheets (CSS) е език за стилово представяне на външния вид и форматиране на маркирано съдържание в HTML, XML и XML-базиран формат.
- CSS предоставя прост механизъм за добавяне на стилови набори към структурирани Уеб документи, чрез който авторите и читателите на документите могат да влияят на начините на представяне на съдържанието.
- Представянето, управлявано чрез CSS, може да се извърши от браузер, печатащо устройство, синтезатор на реч, телевизия и други медии.

CSS – малко история

- ✓ CSS се базира на стиловия език Document Style Semantics and Specification Language (DSSSL), разработен за дефиниране на стилове за SGML през 80-те години на миналия век.
- ✓ За разлика от DSSSL обаче, CSS позволява добавянето на повече от един стилови документа, които да управляват представянето на съдържанието в браузер.
- ✓ Понастоящем CSS е представена чрез набор от спецификации на W3C, достъпни през страницата <http://www.w3.org/Style/CSS/>.
- ✓ CSS е спецификация на W3C

Версии на CSS

1. CSS 1 спецификацията бе завършена през 1996 година – слаба поддръжка от браузерите (освен Opera – още през 1997). Internet Explorer 5.0 за Macintosh през март 2000 г. - първи браузър с почти пълна поддръжка на CSS 1.
2. CSS 2 е препоръка на W3C от май 1998, но многото проблеми с поддръжката ѝ от браузерите доведоха до преразглеждане на стандарта CSS 2 в спецификацията CSS 2.1, която е все още най-популярна за момента. След много преработки CSS 2.1 стана кандидат-препоръка в средата на 2007г.
3. CSS 3 стартира още през далечната 1998 г., но все още е в процес на развитие и се поддръжа само частично от известните браузери (<http://www.w3.org/Style/CSS/>).
4. W3C започва изготвянето и на CSS 4 от 2009 г., която за момента не се поддържа от който и да е Уеб браузер.

Предимства на CSS 1/4

- Стиловото оформление допълва структурираното маркъп съдържание.
- Разделението между двете дава възможност на няколко страници да споделят едно форматиране и така да се намали сложността и повторенията в структурното съдържание (например да се работи с уеб дизайн без таблично структуриране).
- CSS предоставя по-голяма гъвкавост и контрол върху характеристиките на представяне и внасянето на промени.
- Чрез посочване на стилове за Уеб документите, дизайнерите могат да опростят поддръжка на Уеб страниците, като обаче запазят консистентен изглед и усещане за целия сайт.

Предимства на CSS 2/4

- CSS може да позволи една и съща страница от съдържанието да се представя в различни стилове за различни методи за показване в разнообразни медии, като например на екрана, в печатна форма, речеви синтезатор и Брайлова азбука.
- Една уеб страница да се показва по различен начин в зависимост от параметрите на крайното устройство, като размер на екрана, поддръжка на цветове и др.
- Авторът на документа може да зададе един стилов набор, всеки читател може да използва различен стил на представяне, а накрая клиентското устройство (напр. браузер) обикновено има свой вграден стил.

Предимства на CSS 3/4

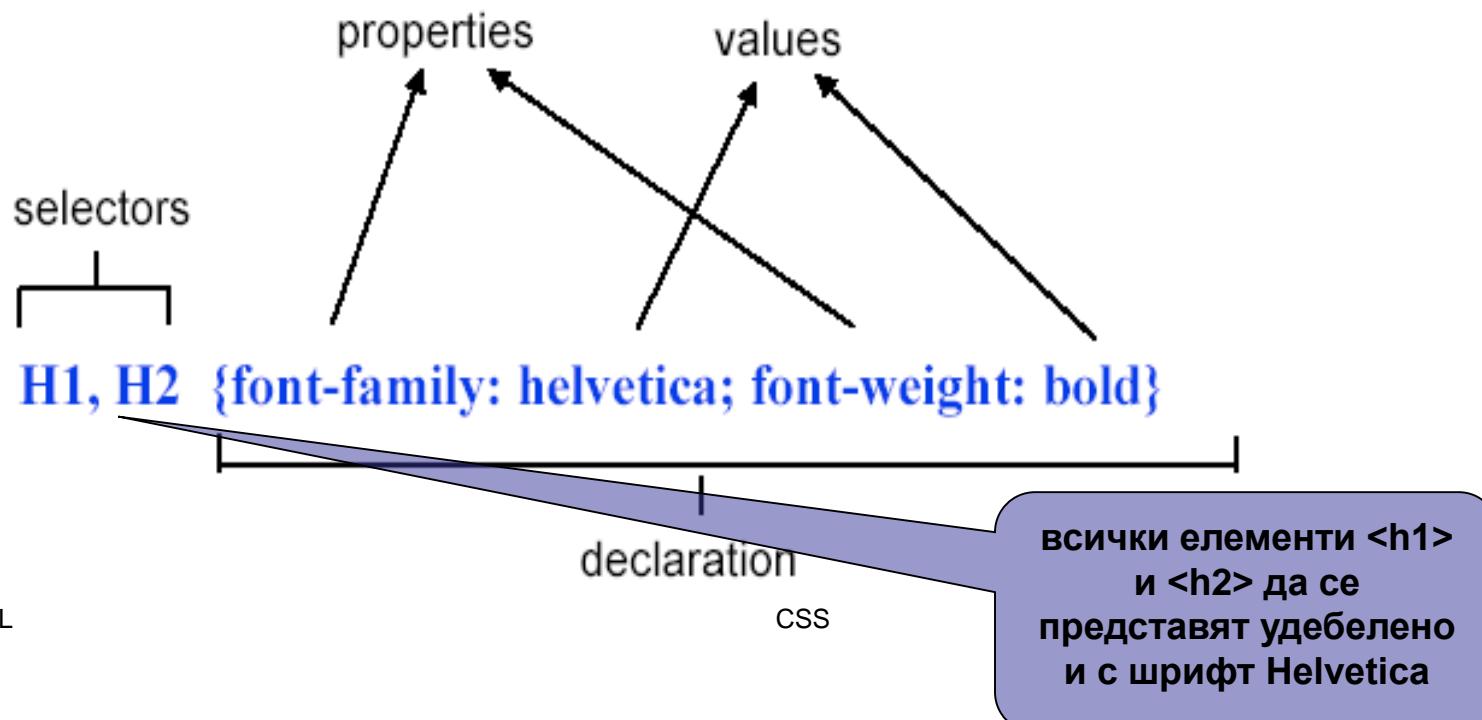
- CSS подобрява достъпността на съдържанието – хора с увреждания имат достъп до съдържанието през подходящи за тях медии и по създадено специално за тях представяне. Потребители със специфични изисквания към представянето могат да пренебрегнат зададения от автора CSS2 стил.
- CSS спецификациите са проектирани така, че да имат съвместимост отдолу нагоре.
 - CSS 2 потребителските клиенти са в състояние да интерпретират CSS1 стилови набори.
 - Същевременно, CSS 1 клиентите могат да четат CSS 2 стилове, като игнорират онези части от тях, които им са непознати.
 - Също така, потребителски агенти, които нямат поддръжка на CSS, могат да представят цялото съдържание без дефинираните CSS стилове.

Предимства на CSS 4/4

- CSS декларациите имат прост синтаксис и се съхраняват в текстови файлове, които могат лесно да бъдат манипулирани от външни приложения - едно приложение може да управлява ефективно начина на представяне в дадена медия на съдържание в HTML, XML и XML-базиран формат, единствено чрез манипулиране на стойностите в CSS декларациите.
- CSS е отворен стандарт, който осигурява независимост от платформи и производители на софтуер.
- CSS е лесно четим от потребителите и позволява бързо оформяне на стиловите набори. Той изразява стила в общата терминология на предпечатната подготовка

Представяне на CSS правила

- ✓ CSS представя стил на документа чрез комбинация от правила за представяне на определени елементи от документа (но не и на атрибути), които да се явят в медията (за CSS 1 - единствено браузер) по желан от дизайнера начин.
- ✓ Примерно CSS правило:



Структура на CSS правила

Правилото е разделено на две части:

- 1. списък от селектори - указва елемент или елементи, за които се прилага декларацията
- 2. декларация - определя как трябва да бъдат оформени елементите от списъка от селектори

CSS контекстов селектор

- Вместо разделен със запетая списък от няколко елемента се допускат и по-сложни модели, за да се показват йерархии от елементи. Така например декларацията

P EM { background: yellow }

задава контекстов селектор. Контекстовият селектор представлява низ от два или повече прости селектори, разделени от празно пространство. На такива селектори могат да се зададат стилови свойства и поради правилата на каскадния ред те ще имат предимство пред простите селектори. Горният пример указва, че подчертаният текст в рамките на параграф трябва да има жъlt фон; подчертаният текст на други места няма да бъде засегнат.

Селектори за клас

- Простите селектори могат да имат различни **класове**, с цел използване на различни стилове за даден елемент. Например за показването на съдържанието на code в различен цвят можем да използваме:
 - `code.html { color: #191970 }`
 - `code.css { color: #4b0082 }`
- Така създаваме два класа - **css** и **html** за ползване за елемента **CODE** в HTML's. Атрибутът **CLASS** се ползва в HTML за означаване на класа на елемента, напр. за
 - `P.warning { color: #191970 }`
 - ползваме
 - `<P CLASS=warning`

Още за класовете

- Класове може да се декларираат и без асоцииран елемент:
 - **.note { font-size: small }**
 - В този случай, класът **note** може да се използва с всеки елемент!
- Добра практика е да назовем класовете в съответствие с тяхната функция, а не с външния им вид. Класът **note** в горния пример би могъл да се нарича **small**, но това име ще стане безсмислено, ако авторът е решил да промени размера на шрифта.

Стилове в HTML

- За да свържете външен списък със стилове към документ, добавете елемента **LINK** в хедъра на вашия документ:
- <LINK> – an EMPTY element**
- Атрибути за елемента LINK:**
 - id** ID #IMPLIED -- SGML ID attribute --
 - href** CDATA #IMPLIED -- URL for linked resource -
 - rel** CDATA #IMPLIED -- *defines the relationship between the linked file and the HTML document. REL=StyleSheet specifies a persistent or preferred style*
 - title** CDATA #IMPLIED -- advisory title string --
 - type** CDATA #IMPLIED -- advisory Internet media type --

Използване на LINK за свързване с външна CSS дефиниция

- **<LINK TITLE="Stanford" REL="stylesheet"
 HREF="http://www.stanford.edu/stanford.dsss!"
 TYPE="application/dsssl">**
- **<LINK TITLE="Cal" REL="stylesheet"
 HREF="http://www.berkeley.edu/cal.css"
 TYPE="text/css">**
- **<h1>Welcome to my amazing home page!</h1>**
- **If your browser supports style sheets, try this
page in Cal and Stanford styles!!**

Атрибут MEDIA (от CSS2)

- Елементът **<LINK>** приема optionalен атрибут **MEDIA**, който специфицира медиите, за които се прилага стила. Възможни стойности са:
- **screen** (стойност по подразбиране), за презентиране на экран;
- **print**, за изход на принтер;
- **projection**, за презентиране на проектори;
- **aural**, за синтезатори на реч;
- **braille**, за Брайлови у-ва;
- **tty**, за телетипни у-ва (с фиксиран шрифт);
- **tv**, за телевия;
- **all**, за всякакви у-ва.
- *Многократни типове медиа се задават в списък разделени със ',' или с **all**.*

Вътрешни стилове – чрез елемента **STYLE**

- Използван за вграждане на стилове в документа, чрез елемента **STYLE**:
- <STYLE TYPE="text/css" MEDIA=screen>
<!--
 BODY { background: url(foo.gif) red; color: black }
 P EM { background: yellow; color: black }
 .note { margin-left: 5em; margin-right: 5em }
-->
- </STYLE>
- Елементът **STYLE** се разполага в **HEAD**. Изискваният атрибут **TYPE** определя CSS тип.

Импортиране на външни стилове в CSS файл

- Стилове могат да се импортират с клаузата `@import` – може да се използва както в CSS файла, така и в **STYLE** елемент:
 - `<STYLE TYPE="text/css" MEDIA="screen, projection">`
`<!--`
 `@import url(http://www.htmlhelp.com/style.css);`
 `@import url(/stylesheets/punk.css);`
 `DT { background: yellow; color: black }`
`-->`
`</STYLE>`
- Забележете, че:
 - Други CSS правила могат да бъдат включени в **STYLE** елемент, но всички `@import` клаузи трябва да се зададат в началото на CSS документа.
 - Всички правила в импортириания документ предефинират тези от импортирани стилове.

Вграден (Inlining) стил

- Начин да приложим стил към единичен таг е да използваме атрибута **STYLE**. **STYLE** може да се приложи към всеки елемент на **BODY** с изключение на **BASEFONT**, **PARAM**, и **SCRIPT**. Стойността на атрибута е една или повече двойки свойство /стойност разделени с двоеточие. Например:
- <P **STYLE**=**"color: red; font-family: 'New Century Schoolbook', serif"**> This paragraph is styled in red with the New Century Schoolbook font, if available.</P>
- Забележете, че **New Century Schoolbook** е ограждан с ‘ в атрибута **STYLE** (а не с “).
- Inlining стиловете не се препоръчват – липса на гъвкавост.

Атрибути за поддръжка на стилове

- Повечето от тях са разрешени в много елементи:
- **ID** – за рефериране към стил в даден елемент
- **CLASS** – разрешава набор от елементи да се групират по стил, зададен от класа
- **STYLE** - съдържа стилова информация за представяне на елемента

ID селектори

- ***ID селекторите*** индивидуално се определят за целите на представяне на даден елемент. Те трябва да се използват пестеливо, поради своите присъщи ограничения. ID селектор се задава чрез използване на индикатор "#", предхождащ името му. Например :
- **#svp94O { text-indent: 3em }**
ще се реферира в HTML документа по ID атрибут:
- **<P ID=svp94O>Text indented 3em</P>**

CLASS и STYLE атрибути

- **CLASS:**
 - **<P CLASS = "special">**

Стиловото правило може да зададе всички елементи от стила **CLASS "special"** например да бъдат с определен цвят.
- **STYLE:**
 - **<P STYLE = "... style sheet right here !...">**
 - Зададеният по-горе **style sheet** ще замени стила по подразбиране, описан чрез HTML **META** елемент.

HTML елементи, подпомагащи стилизирането

- **SPAN** - разграничава (маркира) за целите на стилизирането част от текста, която не притежава структурно значение. Може да има атрибути: **id, class, style**
- **DIV** - разграничава (маркира) за целите на стилизирането поредица от елементи.

SPAN и DIV примеры

- <DIV CLASS="section">
- <P>The spec states that DIV, together with the CLASS attribute, is used to denote structural elements, like "abstract" or "chapter", for which HTML doesn't provide elements.
- <P>DIV allows headers, lists, paragraphs, and other DIV elements.
- <P> Maybe we want the first three words here to be set in a special way. SPAN elements can appear wherever you expect text.
- <P>Presumably, somewhere, a style sheet would have to state how the "init" and "section" classes are to appear. Or, instead, we could have used the STYLE attribute and put a style statement in directly in with the tags.
- XML </DIV>

Свойства, стойности и групиране

- **Свойства (Properties)**
- Свойството (*property*) се задава на селектор за манипулиране на стила му. Примери за свойства: **color**, **margin** и **font**.
- **Стойности (Values)**
- Стойността (**value**) се присвоява в декларацията на дадено свойство (*property*). Например, свойството **color** може да получи стойността **red**.
- **Групиране**
- Групирането намалява повторенията на дефинициите. Например за всички хедъри можем да зададем:
 - **H1, H2, H3, H4, H5, H6 { color: red; font-family: sans-serif }**

Видове CSS свойства 1/2

- ❖ текст и шрифт – в тази група влизат свойствата, които задават какъв да бъде шрифта (**font-style**), неговата големина (**font-size**), дебелина (**font-weight**), разстоянията между буквите (**letter-spacing**) и други.
- ❖ цвят и фон – включва свойства за цвят на текст или очертание (**color**), цвят на фон (**background-color**), изображение за фон (**background-image**) и др.
- ❖ кутиен модел – съдържа свойства свързани с кутийно представяне на текст като дебелина на очертанията на кутийното представяне (**border-width**), неговия стил (**border-style**), цвят (**border-color**), височина (**height**), широчина (**width**) и др.

Видове CSS свойства 2/2

- ❖ позициониране – състои се от свойствата за позициониране като например разстоянието на стартовата позиция от най-горния десен ъгъл (**top**), от най-левия долен ъгъл (**bottom**), отдясно (**right**), отляво (**left**) и др.
- ❖ списъци – включва свойства за представяне на списък като например:
 - ❖ стил (**list-style**)
 - ❖ тип (**list-style-type**)
 - ❖ изображение за булет (**list-style-image**) и
 - ❖ позициониране (**list-style-position**).

Видове CSS елементни селектори 1/3

- универсален – означават се с знака * се прилагат към всички елементи. Пример: *** { }**

- типов – прилага се или поединично, или за повече елементи (разделени със запетая). Пример:

**firstName, lastName, country { margin-top:100px;
position:absolute;}**

- пряк наследник – прилага се за елементи, които са преки наследници на други. Пример:

parent_element > child_element {....}

- наследник – прилага се за всички елементи, които са наследници на даден елемент. За разлика от горния вид селектор, наследниците може да не са преки наследници.

Пример: **element_parent element_descendant {....}**

Видове CSS елементни селектори 2/3

- "брат/сестра,, един след друг – пример: за XML документа

<parent>

```
<brother>...</brother>  
<sister> </sister>
```

</parent>

- CSS правило:

```
brother + sister{ margin-top:100px;  
position:absolute;  
}
```

- ще се приложи за елемента **sister**,
ако той веднага следва след **brother**.

XML

CSS

The **adjacent sibling combinator** (+) separates two selectors and matches the second element only:
1) if it immediately follows the first element, and
2) both are children of the same parent element.

Видове CSS елементни селектори 3/3

- "брат/сестра,, където и да са – пример:
за XML документа

<parent>

<brother>...</brother>

....

<sister> </sister>

</parent>

- CSS правилото:

brother ~ sister{ margin-top:100px;

position:absolute;

}

- ще се приложи за елемента **sister**, ако
XML той следва където и да е слѣд **brother**.
CSS

the difference is that the second selector does **NOT** have to immediately follow the first one means It will select all elements that is preceded by the former selector.

Видове CSS елементни селектори - обобщение

- Универсален: *** { }**
- Типов: **element_1, element_2, ... , element_n {....}**
- Пряк наследник: **parent_element > child_element {....}**
- Наследник: **element_parent element_descedant {....}**
- Брат/сестра: **brother ~ sister { ... }**
- Съседни брат/сестра:
brother + sister { ... }
- Клас за елемент:
my_element.my_class{ ... }

CSS селектори за атрибути

- Задаване:

a[target] { background-color: yellow; }

– селектира всички елементи `<a>` с атрибут `target`

- Атрибут с дадена стойност:

a[target="_blank"] { background-color: yellow; }

- избира `<a>` елементите с атрибут `target="_blank"`

- Атрибут, съдържащ дадена дума:

[title~="flower"] { border: 5px solid yellow; }

- всички елементи с атрибут `title`, съдържащ `flower`

- Атрибут, започващ с дадена (цяла!) дума – разл. от **[class^="top"]**:

[class]="top" { background: yellow; }

- всички елементи с атрибут `class`, започващ с `top`

- Атрибут, завършващ с дадена стойност:

[class\$="test"] { background: yellow; }

XML

33

Наследяване (Inheritance)

- Много от CSS свойствата дефинирани за даден елемент могат да бъдат наследявани от неговите под-елементи.
- Така веднъж дефинирани за даден елемент, те са в сила и за неговите под-елементи.
- Ако искаме обаче да заменим свойствата за даден под-елемент, то това може да стане чрез създаване на специфично свойство конкретно за него.

Наследяване - пример

- За CSS дефиницията
- h1, h2, h3 { color:#000000;**
- margin-top:100px;**
- position:absolute;**
- font-weight:bold; }**
- искаме да добавим свойство **font-style** само за **h2**:
- h2 {font-style:italic;}**
- или да предефинираме свойството **color** за тези елементи **h2**, които следват веднага след **h1**:
- h1 + h2 {color:#000FFF;}**

Псевдо- класове и елементи

- Отнасят се за случаи, наподобяващи структурни елементи, без всъщност да представляват такива.
 - Котва (*Anchor*) **псевдо-класове** – представлят статуса на връзка
 - Типографски **псевдо-елементи** – представлят се като елемент, който се показва по време на изпълнение

Котва (*Anchor*) псевдо-класове

- Отнасят се до елемента **A** (котва) и се използват за различно показване на връзките (links), посетените връзки (visited links) и активните връзки (active links), въответно с имена на псевдо-клас:
- **link**
- **visited**
- **active**.
- Пример: по-голям шрифт и син цвят за активни линкове и по-малък шрифт и зелен цвят за посетени такива:
- **A:link { color: red }**
A:active { color: blue; font-size: 125% }
A:visited { color: green; font-size: 85% }

First Line & Letter псевдо-елементи

- CSS1 включва възможността за различно представяне на първата линия от дадено съдържание чрез **first-line** псевдо-елемент – може да се ползва при всеки **block-level елемент като напр. P, H1, и т.н.:**
- **P:first-line { font-variant: small-caps; font-weight: bold }**
- Псевдо-елементът **first-letter** се използва за така наречените бити букви (drop caps) и други ефекти. Първата буква от текста ще се представи със зададения стил. **first-letter** може да се ползва при всеки **block-level елемент**. Пример:
- **P:first-letter { font-size: 300%; float: left }**
указва бита буква, три пъти по-голяма от текста.
Изврътане с останалия текст (left | right | none).

Други примери

• Псевдо-класове

- A:**link** { color: red } /* unvisited link */
- A:**visited** { color: blue } /* visited link */
- A:**active** { color: green } /* link currently being pressed */

• Псевдо-елементи

- P:**first-line** { font-variant: small-caps; } /* The first-line pseudo-element will make the first line as formatted of a P element appear as if it were inside a P:first-line element, with the result effect on style*/
- p:**first-child** { background: #ff0000 } /* sets a background color for the <p> element that is the **first** child of its parent */
- p:**last-child** { background: #ff0000 } /* sets a background color for the <p> element that is the **last** child of its parent */
- p:**first-of-type** { background: #ff0000 } /* sets a background color for the first <p> element of its parent */
- P:**first-letter** { font-size: 200% } /* first-letter is as if it surrounds the "first letter" of the element. */

• Комбинации

- A.**footnote:visited** { color yellow } /* class and pseudo-class */
- P.**urgent:first-line** { color red } /* class and pseudo-element */

CSS1 пример

<html>

 <head>

 <STYLE TYPE="text/css">

h1, h2, h3 {

 font-family: helvetica; font-weight: bold;

 font-size: 36pt; line-height: 14pt;

 font-family: helvetica; font-variant: normal;}

h2 {font-size: 28pt; font-style: italic; color: blue; font-stretch: condensed}

h3 {font-size: 20pt; font-style: oblique; color: red; font-variant: small-caps}

 </STYLE>

 </head>

 <body>

 <h1> Heading One </h1>

 <h2> Heading Two </h2>

 <h3> Heading Three </h3>

 <body>

XML

 </html>

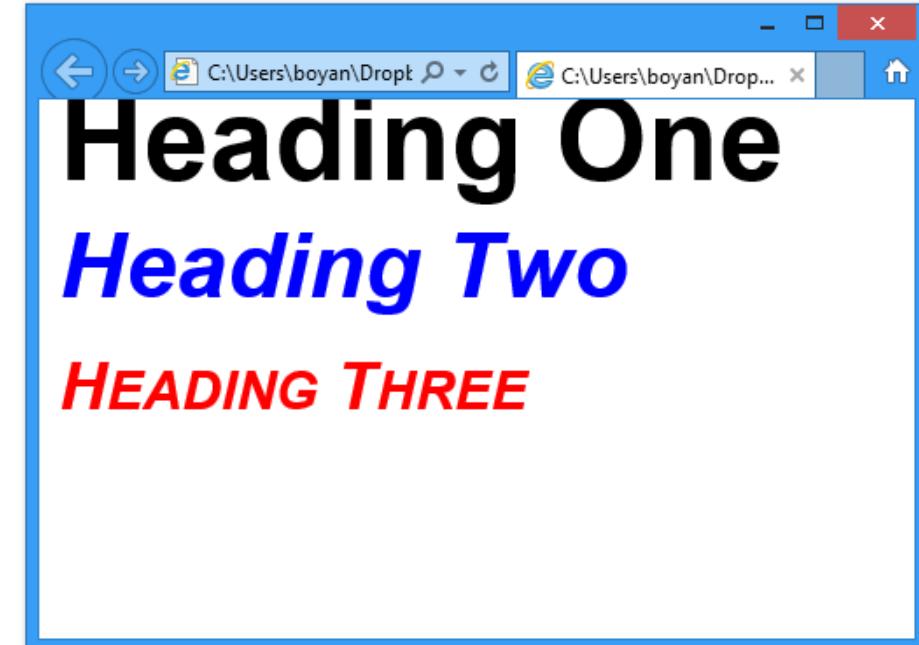
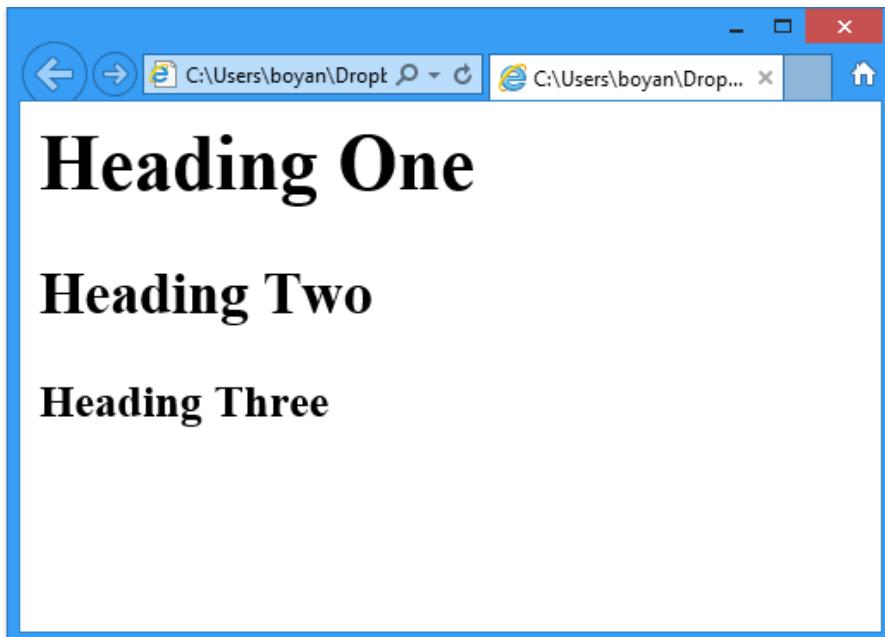
CSS

40

CSS1 пример в браузър

HTML без CSS

HTML с CSS



Каскаден ред – клауза ! important

- При противоречиви правила (за стил на даден елемент) се избира правило по следния ред:
- **! important** правилата са с по-висок приоритет при еднаква тежест на дефинициите и съдържат клаузата **! important**. *Както авторът, така и читателят могат да задават important правила, затова в този случай авторските правила предефинират (отменят) тези на читателя.* Пример за използване на **! important**:
- **BODY { background: url(bar.gif) white; background-repeat: repeat-x ! important }**

•**repeat** Adding this tag just makes your background image tile like it does with the regular background tag.

•**no-repeat** Adding this tag makes your background image appear in its regular size in the upper left hand corner of your page, the image shows only once.

•**repeat-y** Adding this tag makes your background image tile vertically on your page, along the left margin.

•**repeat-x** Adding this tag makes your background image tile horizontally on your page, along the top margin.

Каскаден ред – произход на правилата (Author's vs. Reader's)

- При конфликт, авторските правила предефинират (отменят) тези на читателя.
- Авторските и читателските правила предефинират тези на браузъра.
- Авторите трябва да използват ! **important** правила много внимателно, за да не отменят някое важно читателско правило – например за цвят на текст върху даден фон, с цел добра четимост на текста.

Избор на правила по специфичност

- При конфликтни правила избор на стил се прави и на база на специфичността на правилото.
- Колкото едно правило е по-специфично, толкова по-голям приоритет ще има то самото!
- Примери:
 - Брой на **ID** атрибутите в даден селектор.
 - Брой на **CLASS** атрибутите в даден селектор.
 - Брой на **HTML имена на елементи** в даден селектор.
 - Брой на псевдо-класове/елементи
 - "Pseudo-classes" and "pseudo-elements"

Още за специфичността

- Един селектор е по-специфичен от друг, ако:
 - - има повече **ID** атрибути,
 - - има повече **CLASS** атрибути,
 - - има повече **имена на елементи**.
- Примери:
 - **UL UL** по-специфичен от **UL**.
 - **UL.urgent** по-специфичен от **UL UL UL UL**
 - **UL UL.urgent** по-специфичен от **UL.urgent**.

Cascading Style Sheets (CSS2)

- W3C Recommendation since 2001,
<http://www.w3.org/TR/REC-CSS2/>
- CSS2 е изградена върху CSS1 и, с много малко изключения
- Всички валидни CSS1 стилови дефиниции са валидни CSS2 стилове.
- CSS2 поддържа специфични стилове за различни медии, така че авторите могат да представят техните документи за визуални браузъри, звукови устройства, принтери, брайлови устройства, преносими устройства, и др.

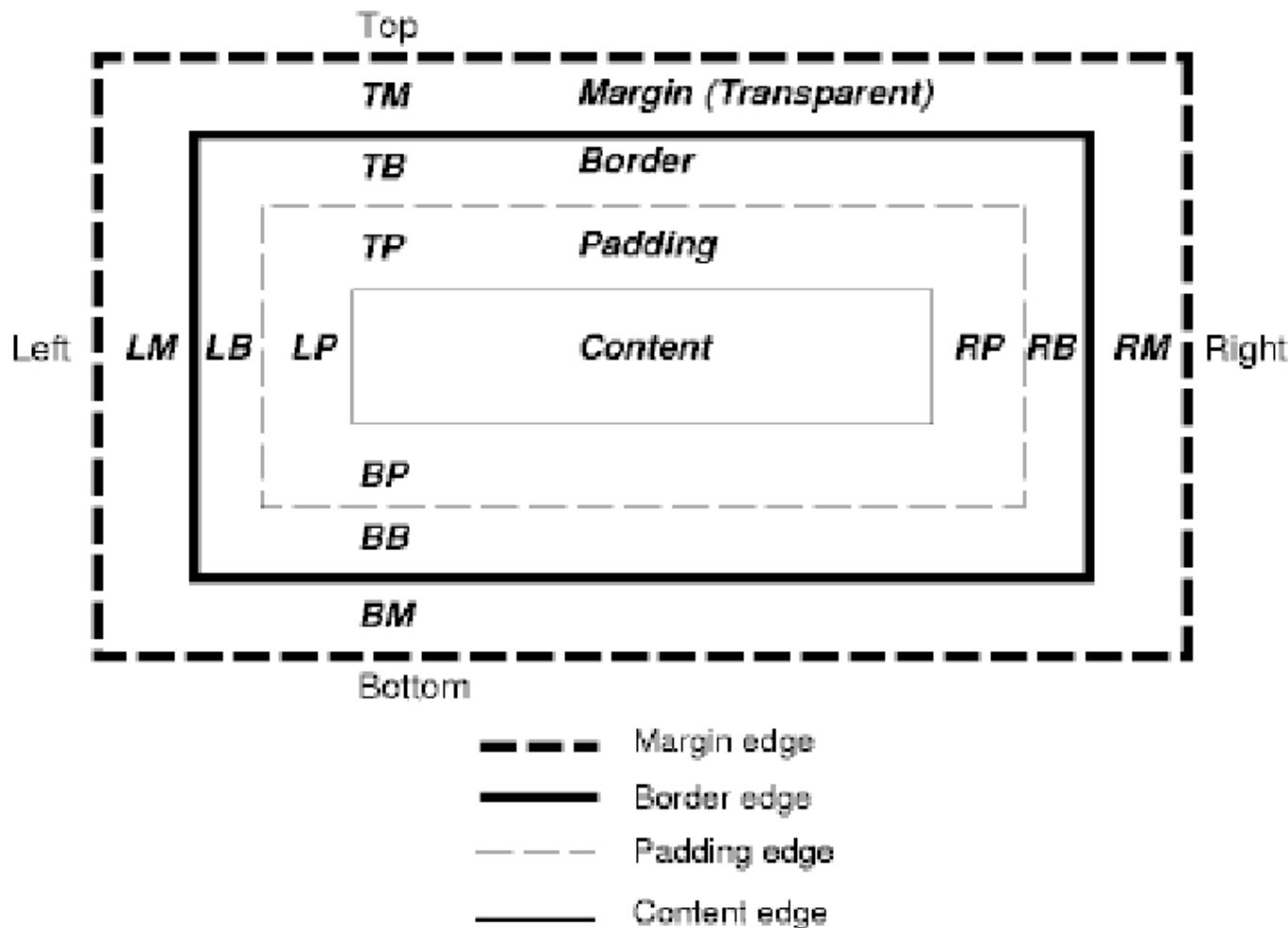
Cascading Style Sheets (CSS2)

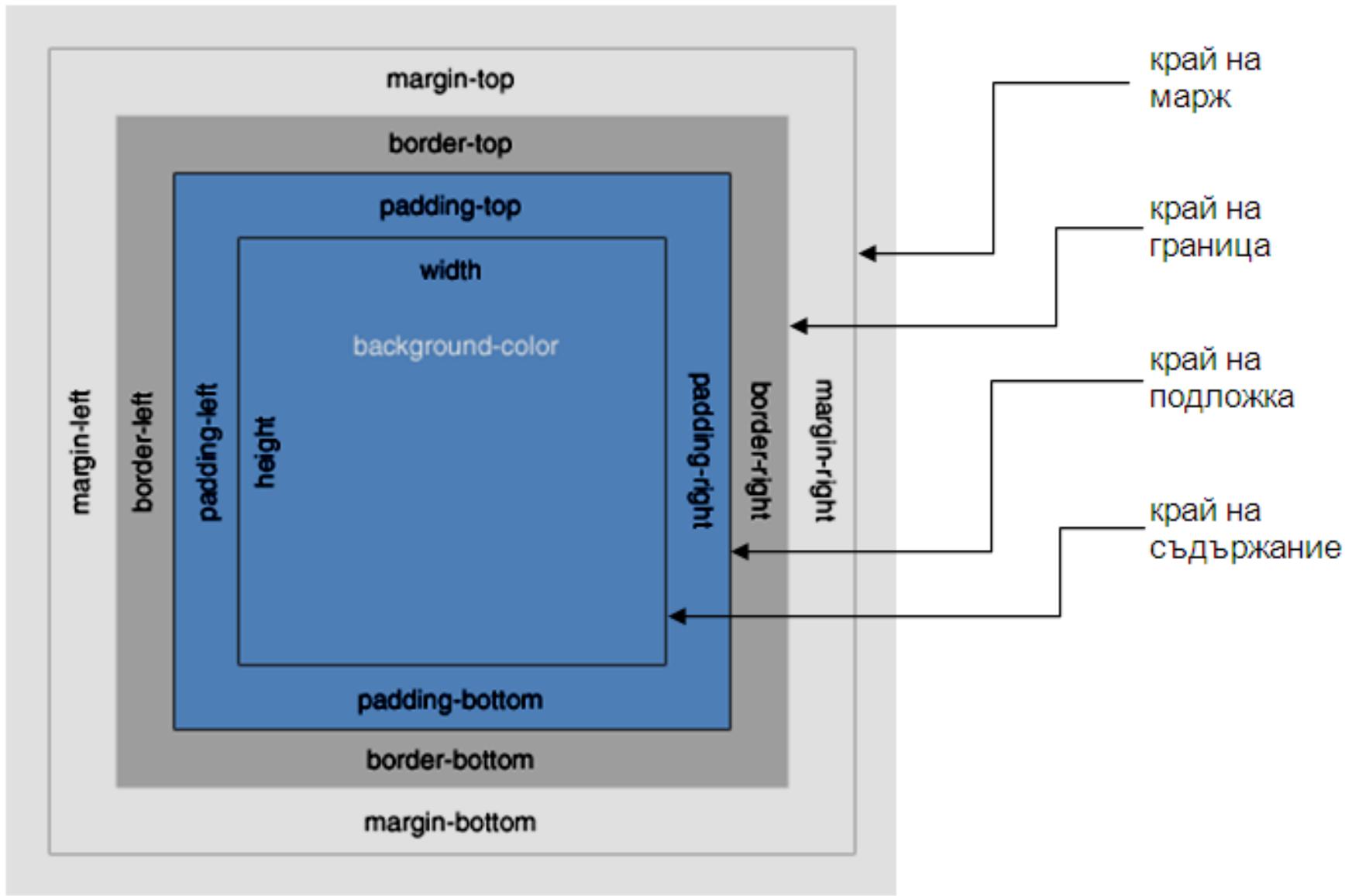
- Поддържа:
 - Различни медии
 - Схеми за позициониране на съдържанието
 - Зареждане на шрифтове (downloadable fonts)
 - Разполагане на таблици (table layout)
 - Интернационализация
 - Автоматично броене и номериране
 - Нови свойства на потр. интерфейс
- Работи както с XML, така и с HTML документи

Кутиен модел (усъвършенстван в CSS2)

- При показване на документ, CSS третира всеки елемент в документа като правоъгълна кутия
- Всяка кутия се състои от четири компонента:
 - съдържание
 - подложка (*padding*), заобикаляща съдържанието
 - граница (*border*), и
 - допустими граници или маржове (*margin*).
- Маржовете на всяка кутия са прозрачни, а границите ѝ могат да имат стилове (непрекъсната или пунктирена л-я).
- Подложката представлява зоната между границата и съдържанието. Цвят на фона се прилага към области вътре в рамката на границата, която включва подложката и съдържанието.

Елементи на CSS кутия: content, padding, border, and margin





Блоково и поредово представяне

1/2

- Всяка кутия може да съдържа други кутии, отговарящи на елементи, които са вложени в нея.
- В CSS съществуват два основни вида кутии: блокови (*block*) и поредови (*inline*).
- Блоковите кутии представлят съдържанието поблочно - всеки параграф се показва с пренасяне на нов ред преди и след параграфа
- Съдържанието на поредовите кутии може да се движи заедно с обкръжението си без пренасяне на нов ред, както например част от текста с удебелен шрифт в средата на параграфа.

Блоково и поредово представяне

2/2

- В обичайното поточно представяне на съдържание, блоковите кутии се нареджат една след друга вертикално, докато поредовите кутии се разполагат една след друга хоризонтално. Например, параграфът е стек от блокови кутии-линии, всяка от които се състои от серия от поредови кутии.
- Има и други видове кутии, като например таблиците, които имат смесено представяне.
- В HTML или XHTML, блоковите кутии са създадени от елементи като например **<p>**, **<div>** или **<table>**, докато поредовите кутии се създават от тагове като ****, **** и ****, както и за съдържание като текст и изображения.

Свойство **Display**

Разлика между HTML и XML:

- HTML елементите са или block, или inline,
- при оформяне на XML с CSS браузърът не знае кои елементи трябва да се показват по блокове и кои поредово => използваме CSS за XML с **display** свойство, с допустими стойности:

- **Block**
- **Inline (по подразбиране!)**
- **None (`display:none <=> visibility:hidden !`)**
- Inherit
- Table
- Table-row-group (equiv. to HTML element <TBODY>)
- Table-row (equiv. to HTML element <TR>)
- Table-cell (equiv. to HTML element <TD>)
- Други ...

faq

body

section

qna

q

a

qna

q

a

Вграждане на кутии

CSS пример: “*rolodex.css*”

- **rolodex** {display: block}
- **last** {font-family: garamond; font-weight: bold; color: blue}
- **name** {display: block; font-family: garamond; text-decoration: italic; font-weight: bold; color: green; }
- **address** {font-family: garamond; font-weight: bold; color: blue}
- **record** { display:block; width:2in;
border-top: solid red;
border-right: solid red;
border-bottom: solid blue;
border-left: solid red }

CSS пример - XML

- <?xml version="1.0"?>
- <!DOCTYPE Rolodex SYSTEM "rolodex.dtd">
- <?xml-stylesheet type="text/css" href="rolodex.css"?>
 - <Rolodex>
 - <record>
 - <name gender = "female">
 - <first>Jane</first>
 - <last>Poe</last>
 - </name>
-

```
1 Rolodex {display: block; color: red; font-family: garamond; font-weight: bold; text-decoration: underline;}
```

```
2
```

```
3 name {display: block; font-family: arial; font-style: italic; text-decoration: underline; color: green;}
```

```
4
```

```
5 address {font-family: times; font-style: italic; color: blue;}
```

```
6
```

```
7 record { display: block; width: 2in; border-top: solid red; border-right: solid red; border-bottom: solid blue; border-left: solid red; }
```

```
8
```

```
9
```

```
10
```

```
11
```

```
12
```

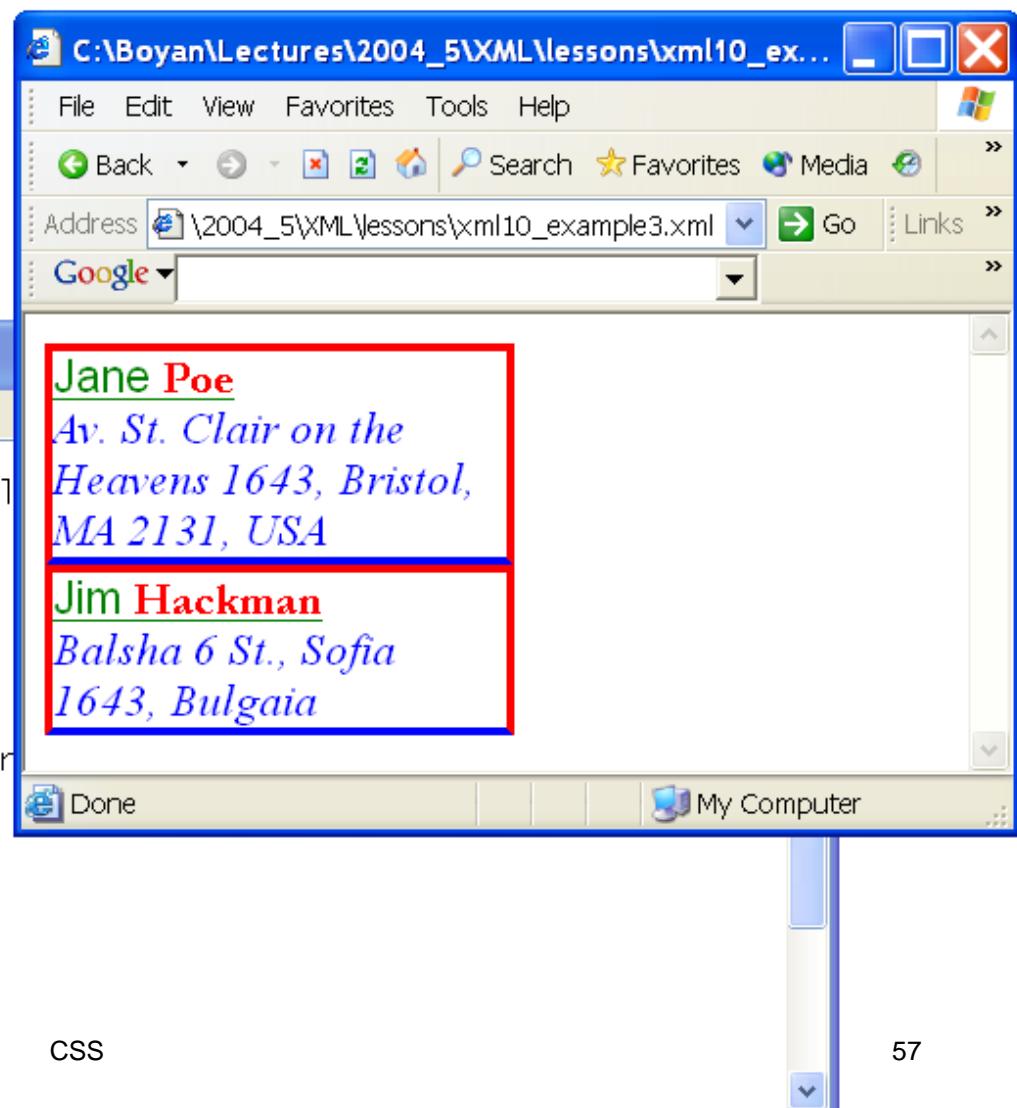
```
13
```

```
14
```

xml10_example3.xml - Notepad

```
File Edit Format View Help
```

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="example3.css"?>
<Rolodex>
  <record>
    <name gender = "female">
      <first>Jane</first>
      <last>Poe</last>
    </name>
    <address>
      Av. St. Clair on the Heavens 1643, Bristol, M4 2131, USA
    </address>
  </record>
  <record>
    <name gender = "male">
      <first>Jim</first>
      <last>Hackman</last>
    </name>
    <address>
      XML Balsha 6 St., Sofia 1643, Bulgaria
    </address>
  </record>
</Rolodex>
```



Позициониране в CSS

- След като съдържанието на всеки елемент може да се показва като кутия, процесът на оформление на представянето се свежда до вземане на решение кой тип кутия (с блоково или поредово представяне) да се използва за всеки елемент в документа.
- Освен това, трябва да се определи местоположението - тоест позиционирането - на тази кутия.
- Спецификацията на CSS 2 дефинира чрез стойността на свойството **position** няколко вида на позициониране - нормалното (статично) позициониране на потока, плаващо (относително) и абсолютно позициониране.

Схеми за позициониране 1/3

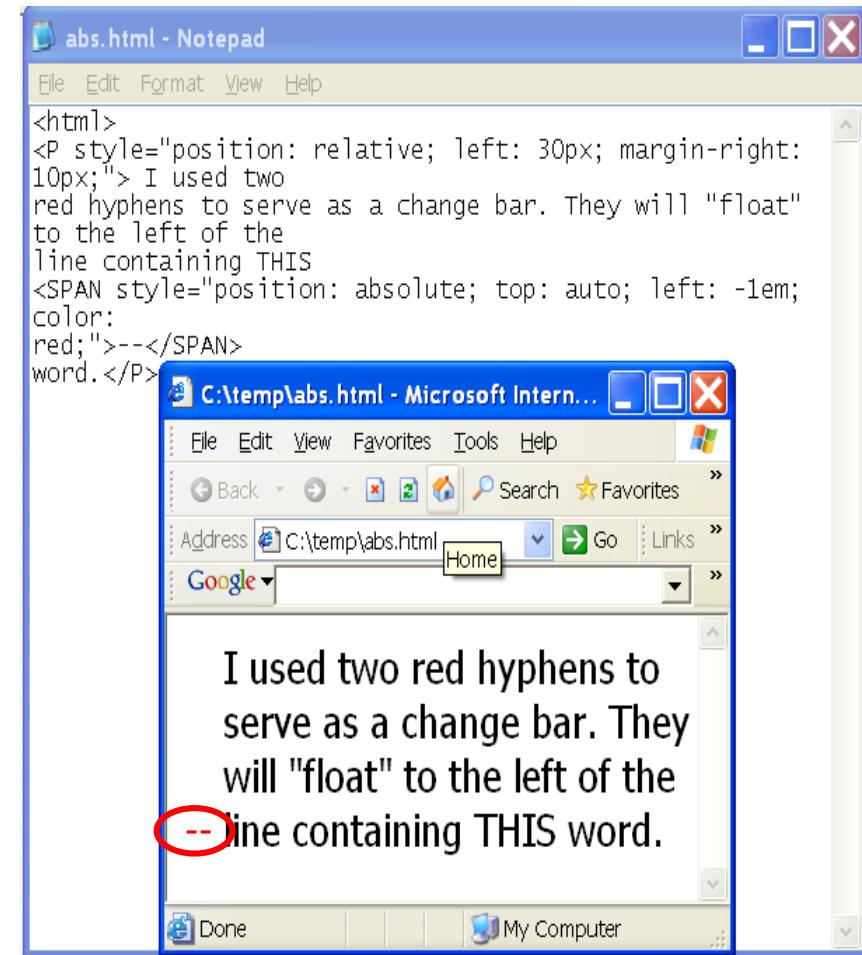
- **Статично позициониране - position:static** задава статично позициониране. То се контролира от браузера и е най-бързо.
- **Относително позициониране** - подобно на статичното, като обаче горната и лявата позиции могат да бъдат предефиирани от CSS. То е вариант на нормалния поток и се задава със стойност **position: relative**.

Схеми за позициониране 2/3

- **Абсолютно позициониране (`position: absolute`)** имаме тогава, когато на кутията е присвоено фиксирано отместване от местоположението ѝ, определено спрямо фиксирания поток. То не оказва влияние върху това как са определени *Sibling* кутиите.
- В зависимост от нивото на вложеност, абсолютно позиционираният елемент може да закрие други елементи.

Пример за относително с абсолютно позициониране

- **<P style="position: relative; left: 10px; margin-right: 10px;"> I used two red hyphens to serve as a change bar. They will "float" to the left of the line containing**
- **--**
- **THIS word.</P>**
- Резултат →



Друг пример

Използваме стила

- <STYLE TYPE="text/css">
- .strut { font-size: 24pt; position: relative; color: green; }
- .anno { font-size: 10pt; position: absolute; top: 0px }
- </STYLE>
- В примера
- <P style="position: relative">This is some plain HTML text. We would like to place an external annotation on it, which is still hard to do, but we can at least make a span
-
-
-
- Replace:
-
- here is candidate replacement text
-
- <u>that is marked for replacement</u> as this example illustrates.

Резултат

example2.html - Notepad

```
<html>
<STYLE TYPE="text/css">
.strut { font-size: 24pt; position: relative; color:
green; }
.anno { font-size: 10pt; position: absolute; top: 0px }
</STYLE>
```

```
<P style="position: relative;">
text. We would like to p
it, which is still hard t
a span
<span class=strut>
<span class=anno><span st
font-size:10pt; color: re
Replace:</span>&nbsp;&nbs
text</span>
</span><u>that is marked
example illustrates.
```

C:\temp\example2.html - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Home Search Favorites Media

Address C:\temp\example2.html

Google

This is some plain HTML text. We would like to place an external annotation on it, which is still hard to do, but we can at least make a span that is marked for replacement as this example illustrates.

Replace: here is candidate replacement text

Done

CSS

XML

My Computer

63

Схеми за позициониране 3/3

- **Плаващо позициониране** - със стойност **position:float** - създава кутия, като позволява друго съдържание да тече (плава) около тази кутия.
- **Фиксирано позициониране** - подкатегория на абсолютното позициониране. Елемент, за който имаме **position:fixed**, винаги има изгледа като съдържащия блок. За непрекъснати медии, като например компютърен еcran, фиксираните елементи няма да се движат, когато документът се превърта (скролира). За пейджър медиите, фиксираният елемент ще се повтаря на всяка страница.

Още примери

- **W3 School: CSS School**

- <http://www.w3schools.com/css/>

- **CSS Basic:**

- **CSS Introduction**
 - **CSS Syntax**
 - **CSS How To**
 - **CSS Background**
 - **CSS Text**
 - **CSS Font**
 - **CSS Border**
 - **CSS Margin**
 - **CSS Padding**
 - **CSS List**

- **CSS Advanced:**

- **CSS Dimension**
 - **CSS Classification**
 - **CSS Positioning**
 - **CSS Pseudo-class**
 - **CSS Pseudo-element**
 - **CSS Media Types**



Семантичен Уеб. Resource Description Framework (RDF)

Въведение в семантичен Уеб.

RDF

RDF/XML

Turtle, N3, RDFa

Примери

Източници

- RDF Primer, W3C Recommendation 10 February 2004
- Framework for the Semantic Web: *An RDF Tutorial*, by S. Decker et al, Stanford Univ.
- Introduction to Semantic Web,
от Ivan Herman, W3C, 2011

Интернет и Уеб

- Роля на World Wide Web (Световна паяжина), съкр. WWW, Уеб или Световната мрежа
- Нов начин на общуване между хората, общностите и дори начините за комуникация между компютрите
- Услуги за е-бизнес, електронна търговия, е-икономика, електронно правителство, електронна демокрация, електронно обучение, ...
- Информационен обмен и взаимодействие между различни актори – лица, организации, Уеб приложения, интелигентни агенти...

Развитие на Уеб

- Тим Бърнърс-Лий, Церн, 1991 г. - разработва основите на Уеб
- Проектира прости средства за пренос на взаимосвързани документи със структурирана информация до всякакви компютри, свързани в Интернет и работещи с различни операционни системи :
 - ✓ език за маркиране на хипертекст (Hypertext Markup Language, или съкр. HTML), и
 - ✓ протокол за трансфер на хипертекст (Hyper Text Transfer Protocol, или съкр. HTTP)
- По-късно - спецификация на URI (Uniform Resource Identifier) - нотация за уникално идентифициране на обекти в целия Интернет.

Уеб 1.0 (традиционн Уеб) 1/3

- **Web 1.0** - термин за етап от еволюцията на World Wide Web; обхваща периода от 1993 до 2001г.
- **Бизнес модел:**
 - top-down подход за изграждане и използване на WWW - статични страници с хипертекст (Hyperlinks е стандарт на WWW от 1993г.)
 - страници на малцина автори на съдържание (webmasters), зареждани от голям брой потребители с глобален достъп;
 - фокус върху презентацията, а не върху създаването на съдържание;
 - печалби от броя посещения (most visited webpages)

Уеб 1.0 (традиционн Уеб) 2/3

- **Технически характеристики:**
- статичен хипертекст – без динамика в браузера
- липса на редактиране на страниците от външни потребители
- използване на Framesets - рамката (frame) е начин за представяне на няколко Уеб страници и/или медия елементи в един прозорец (или таб) на браузъра:
 - характерен за HTML 3 и 4;
 - липса на поддръжка от много браузъри, лоша индексация от търсачките, трудни връзки към рамкираните страници, лошо скролиране при ниска резолюция;
 - изключени от HTML 5

```
<frameset cols="65%, 35%">
  <frame src="URL OF FRAME PAGE 1">
  <frame src="URL OF FRAME PAGE 2">
  <noframes> Sorry but your browser do not support frames ☺
  </noframes> Семантичен Уеб. RDF
</frameset>
```

Уеб 1.0 (традиционн Уеб) 3/3

- **Още технически характеристики:**
 - използване на таблици (`<table>`) за подравняване на съдържанието на страницата
 - отделяне на съдържание с прозрачни 1x1 pixel изображения в GIF format
 - патентовани нестандартни HTML елементи като `<blink>` и `<marquee>`
 - онлайн книги за гости
 - изпращане на HTML форми като ел. поща от статичен хипертекст
 - сървърни технологии като PHP, Ruby, Perl, Python, JSP, and ASP.NET

Уеб 2.0 (социален Уеб) 1/3

- **Web 2.0:**

- ✓ термин за втория етап от еволюцията на World Wide Web;
- ✓ от началото на века до наши дни;
- ✓ въведен от Tim O'Reilly на Web 2.0 conference през 2004

- **Бизнес модел:**

- ✓ top-down + bottom-up подход за изграждане и използване на WWW - динамични хипертекст страници с авторско съдържание и на самите потребители;
- ✓ добавена стойност от споделянето на информация и сътрудничеството между организации и хора;
- ✓ фокус върху създаването на съдържание и персонализираната презентацията

Уеб 2.0 (социален Уеб) 2/3

- **Технически характеристики:**

- ✓ динамичен хипертекст
- ✓ съдържание от външни потребители
- ✓ модел "Network as platform" - потребителски интерфейси за достъп до разл. услуги като напр. публични сайтове с галерии на потребителя (Flickr, Picasa Web Albums, ...), частни и споделени хранилища за данни (DropBox), споделени документи (Google Docs), представяне на геогр. обекти върху карта (Google Mail API), ...
- ✓ оперативен обмен на данни (interoperability)
- ✓ Rich Internet Application (RIA)

Уеб 2.0 (социален Уеб) 3/3

- **Нови технологии:**

- ✓ Клиентски (client-side/web browser):

- XML или JSON (JavaScript Object Notation)
- asynchronous JavaScript (Ajax)
- Adobe Flash
- Adobe Flex
- JavaScript/Ajax frameworks като jQuery
- HTML5 - изиска по-малко изчислителни ресурси отколкото Adobe's Flash; по-малко ел. мощност (батерия при мобилни устройства); замразяване на публичните мобилни Adobe's Flash приставки (plugins)

- ✓ Сървърни технологии

Социални феномени в Уеб 2.0

- **Podcasting** - от broadcast и (i)Pod - сваляне на онлайн видео или аудио съдържание от настолни или мобилни компютри
- **Blogging** - web log - поддържане на личен журнал, публикуван в Уеб на дискретни порции (т.нар. posts), показвани в ред, обратен на хронологичния
- **Tagging** - добавяне на метаданни (описания с ключови думи и термини) към съдържание или части от него, с цел да се ползват при търсене и разглеждане (browsing)
- **Folksonomy (social tagging)** - много потребители добавят метаданни като кл. думи към споделени ресурси - Golder, Scott; Huberman, Bernardo A. (2006). "Usage Patterns of Collaborative Tagging Systems". *Journal of Information Science* 32 (2): 198–208.
- **Social bookmarking** - социални отметки - организиране, поддръжка и търсене на отметки към онлайн ресурси
- **Social networking** - сътрудничество и съревнование в Уеб

Ограничения на днешния Уеб



Machine-to-human, not machine-to-machine

Проблем: приложенията не разбират значението

- “*My mouse is broken. I need a new one...*”



Използване на онтологии

“My mouse is **broken**” vs. “My mouse is **dead**”

XML

Семантичен Уеб. RDF

13

Looking for a “Blue Car with Red Doors”



Red Car with
Blue Doors



Navy Sedan with
Crimson Hatches



Blue Car and
a Red Door



Car,
Blue Chair,
Red Door

Simple word-matching



navy = blue
crimson = red
sedan = car
hatch = door

Thesaurus-based search



"blue car red door"



Red Car with Blue Doors



Navy Sedan with Crimson Hatches



Blue Car and a Red Door



Car, Blue Chair, Red Door

navy = blue
crimson = red
sedan = car
hatch = door

Semantic Matching



"blue car red door"



hasColor(car, blue)
hasColor(door, red)
hasPart(car, door)

?



Red Car with Blue Doors



Navy Sedan with Crimson Hatches



Blue Car and a Red Door



Car, Blue Chair, Red Door

navy = blue
crimson = red
sedan = car
hatch = door

Semantic Matching



"blue car red door"



hasColor(car, blue)
hasColor(door, red)
hasPart(car, door)



Red Car with Blue Doors

hasColor(car, red)
hasColor(door, blue)
hasPart(car, door)

Navy Sedan with Crimson Hatches

hasColor(sedan, navy)



Blue Car and a Red Door

hasColor(car, red)
hasColor(door, blue)



Car, Blue Chair, Red Door

hasColor(chair, blue)

Подход към семантичния Уеб

“The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications”



<http://www.w3.org/sw/>



Семантичният уеб е инициатива с цел разширяване на текущия Уеб и улесняване на Уеб автоматизацията чрез достъпни в Уеб ресурси и 'Мрежа от доверие' ('Web of Trust') - универсално достъпна платформа, която позволява данните да бъдат споделяни и обработени както от автоматизирани средства, така и от хора.

Уеб 3.0 (семантичен Уеб) 1/2

- Еволюция и переход от сегашното състояние на Световната мрежа към семантичен Уеб
- ✓ Семантични услуги – базирани на онтологии за представяне на знанието за дадена предметна област:
 - Семантично анотиране на съдържание
 - Семантично търсене
 - Семантично разглеждане
 - Семантично препоръчване
 - ...

Уеб 3.0 (семантичен Уеб) 2/2

- Семантични езици – базирани на XML
- ✓ **Resource Description Framework (RDF)** - за описание на модела на метаданните относно Уеб ресурси
- ✓ **RDF Schema (RDFS, RDF(S), RDF-S или RDF/S)** - набор от класове с определени свойства представени в RDF, за описание на онтологии и RDF речници за структуриране на RDF ресурси
- ✓ **SPARQL Protocol + RDF Query Language** - език за заявки към RDF графи
- ✓ **Web Ontology Language (OWL)** - фамилия от езици за представяне на знания чрез онтологии
- ✓

Средства за изграждане на Уеб 3.0

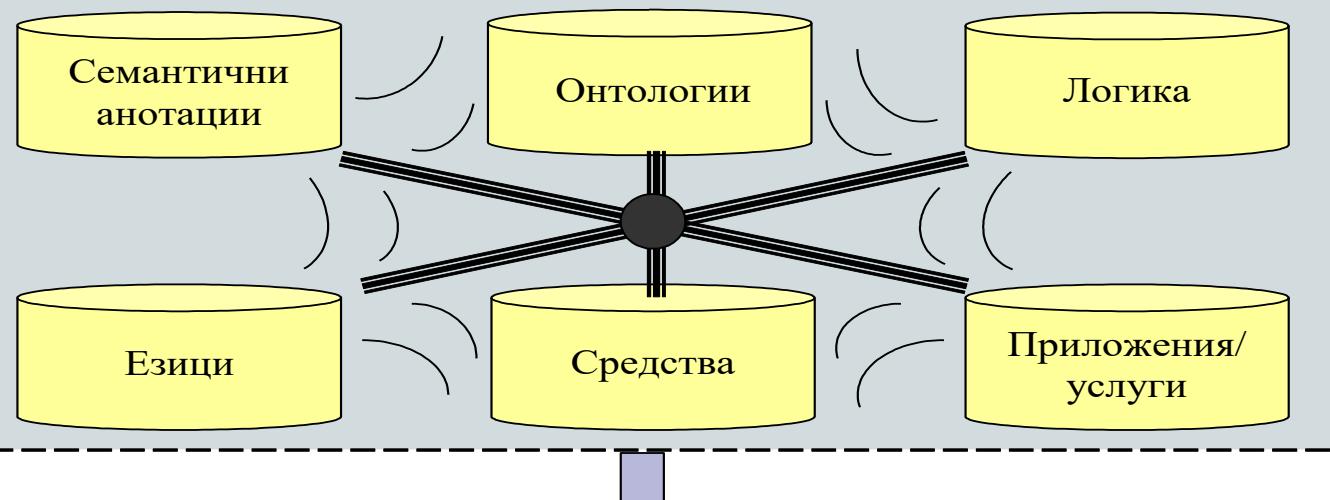
приложения

Семантичен Уеб и след него



агенти

Семантичен Уеб



WWW и след него

XML



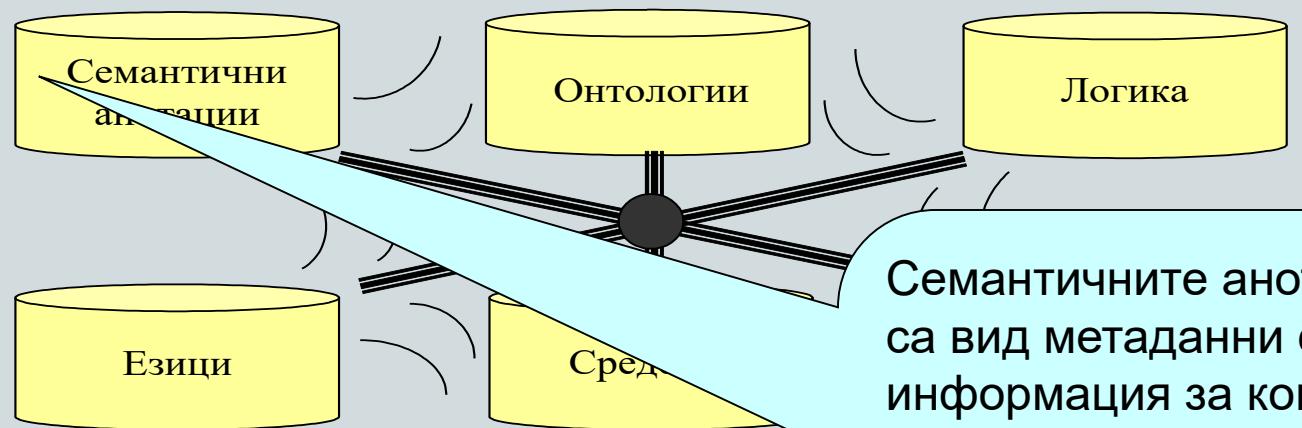
Семантичен Уеб: анотации

Семантичен Уеб и след него



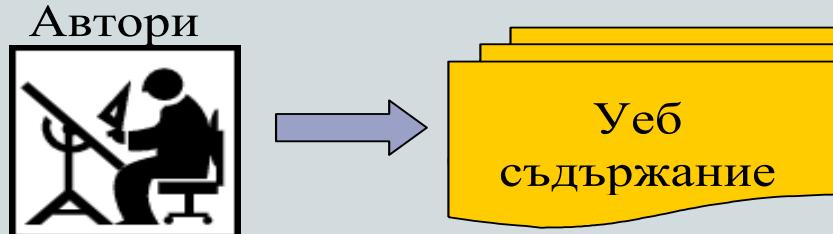
приложения

Семантичен Уеб



Семантичните анотации са вид метаданни с информация за конкретни домейн-обекти, стойности на техни свойства и взаимоотношения, в машинно-обработваема, формална и стандартизирана форма.

WWW и след него



XML

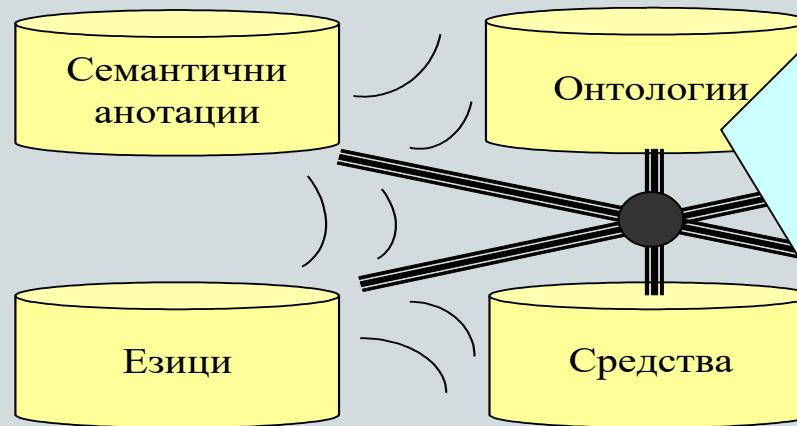
Семантичен Уеб: онтологии

Семантичен Уеб и след него



приложения

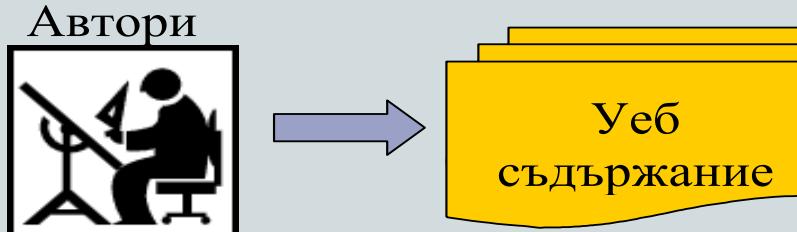
Семантичен Уеб



Онтологията прави метаданните оперативно съвместими и готови за ефективно споделяне и повторна употреба. Тя осигурява споделено и общо разбиране на домейна и може да се използва от хора и машини. Представя част от знанията за света, базирана на споразумение, и най-общо описва домейна чрез екземпляри, класове, атрибути, отношения и събития.

WWW и след него

XML



Семантичен Уеб: правила

Семантичен Уеб и след него



Автори

Семантично съдържание в Уеб

Потребители



приложения

Семантичен Уеб

Семантични анатации

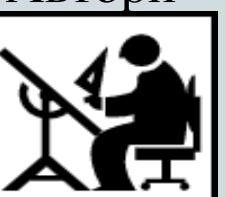
Онтологии

Логика

Езици

Логическата поддръжка на правила е позволява изводи на база на съдържание, метадани и онтологии, по експлицитни правила. Правилата се считат за основа за бъдещето развитие на семантичния Уеб като средство да се правят заключения, за изразяване на ограничения, определяне на политики, реакция на събития, трансформиране на данни, задаване поведението на агенти...

WWW и след него



Автори

Семантичен Уеб: езици

Семантичен Уеб и след него



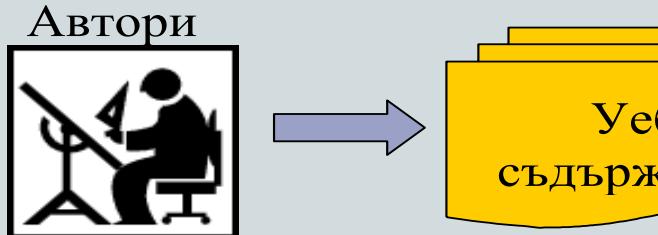
приложения

Семантичен Уеб



Езиците са необходими за машинно-обработваеми формални описания на метаданни (анотации), като напр. RDF; онтологии като напр. OWL, правила като напр. RuleML. Предизвикателството: рамка на синтаксиса (напр. XML) и семантика на тези езици по единен и съгласуван начин.

WWW и след него



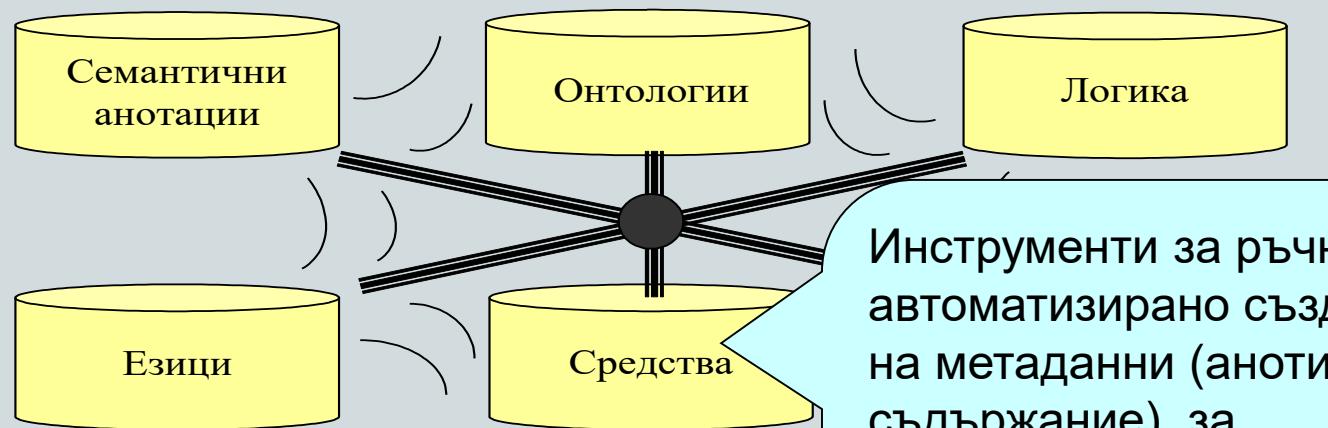
Семантичен Уеб: среди

приложения

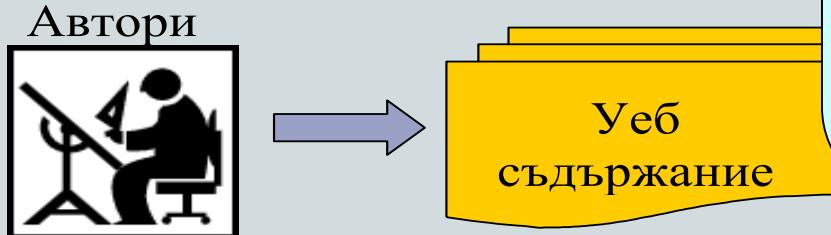
Семантичен Уеб и след него



Семантичен Уеб



WWW и след него



Инструменти за ръчно или автоматизирано създаване на метаданни (анотирано съдържание), за инженеринг на онтологии и валидиране, за придобиване на знания (правила), за разбор и обработка на езици и др.

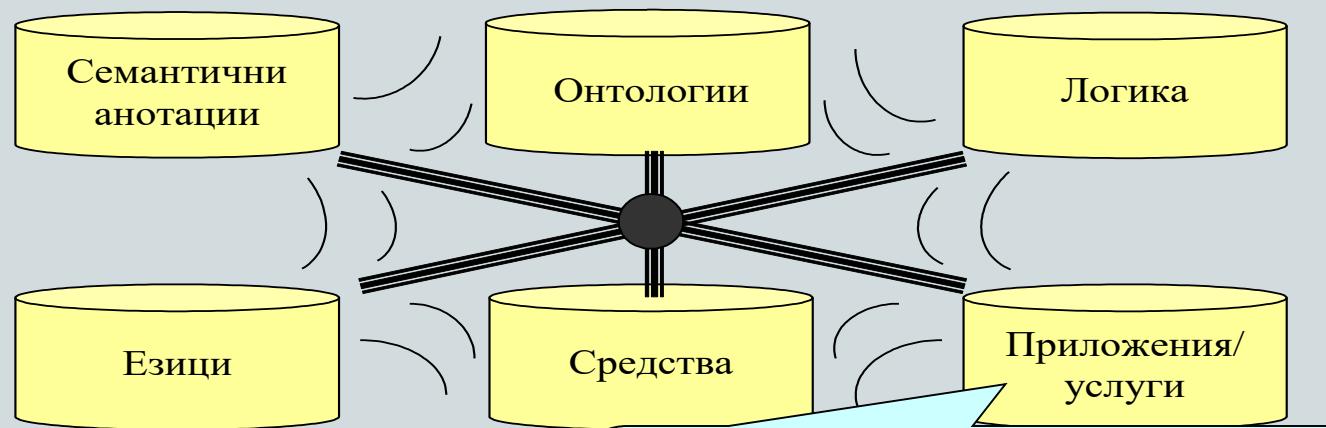
Семантичен Уеб: приложения и услуги

Семантичен Уеб и след него

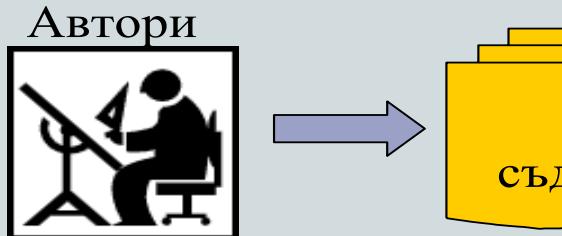


приложения

Семантичен Уеб

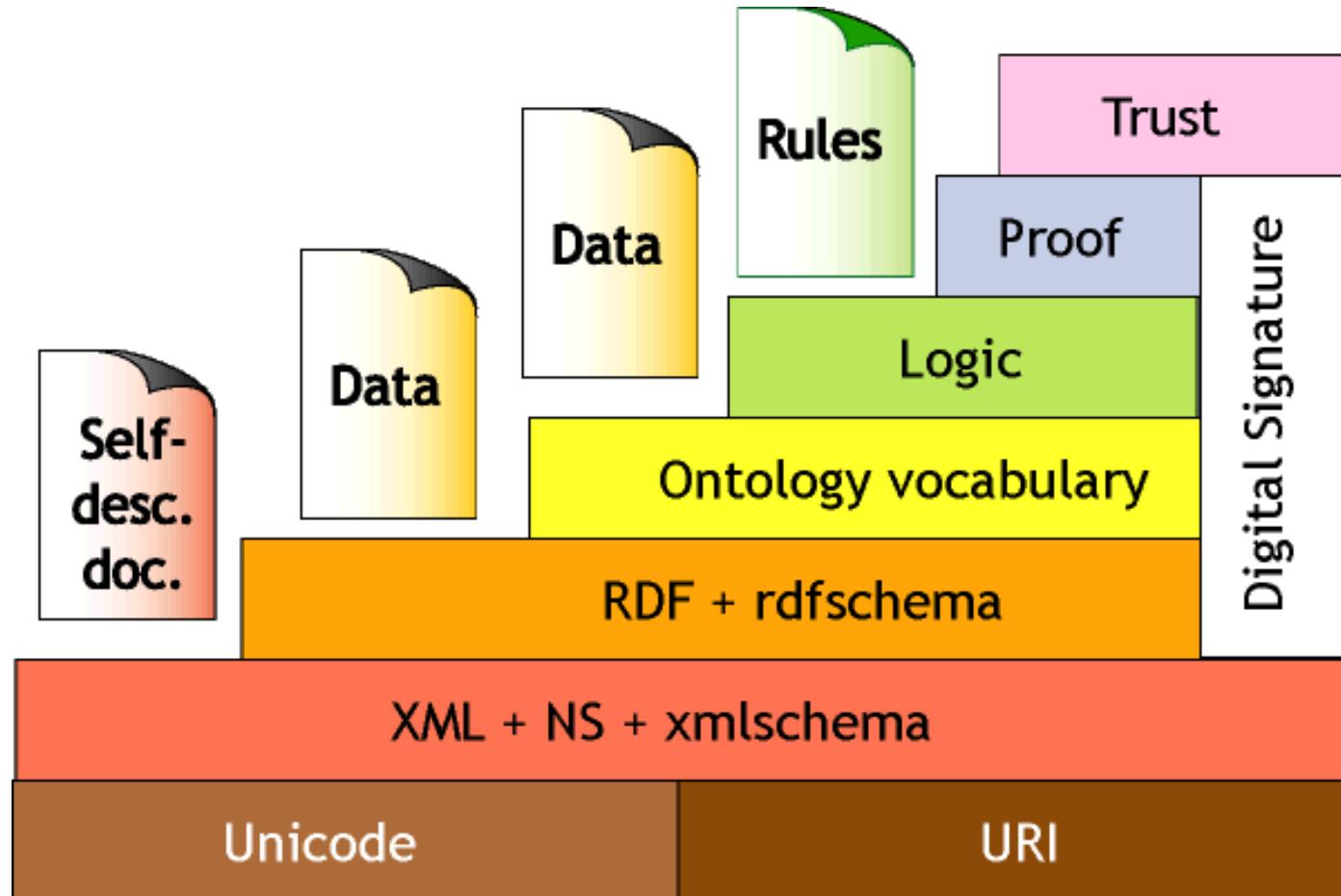


WWW и след него



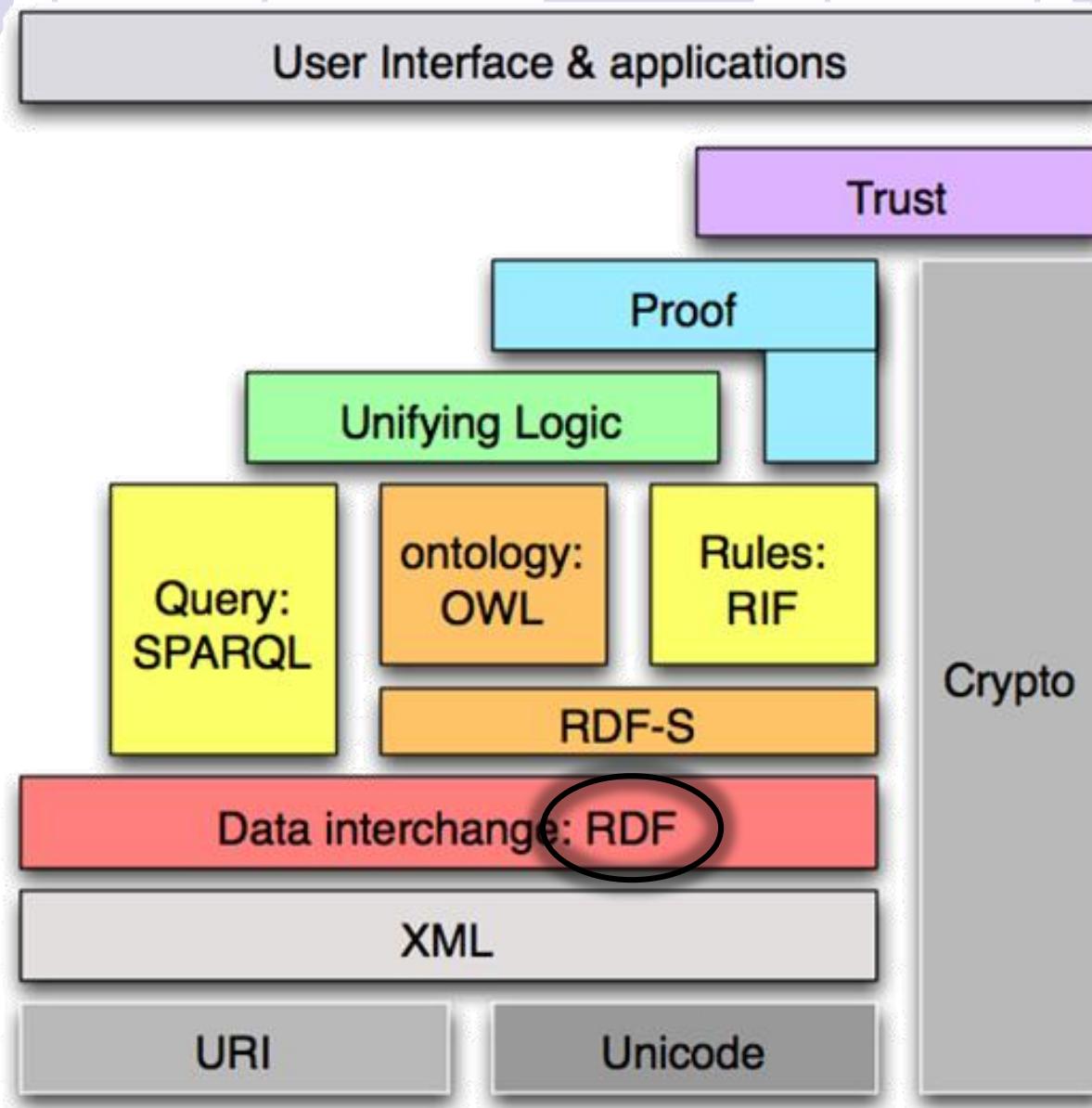
Използването на семантични метаданни, онтологии, правила, езици и инструменти дава възможност за предоставяне на мащабирими и интелигентни уеб приложения с услуги за потребители и **агенти**.

Визия на Tim Berners-Lee за семантичен Уеб (IJCAI-01)



Стек на семантичния Уеб (W3C, 2006)

Adapted from http://en.wikipedia.org/wiki/Semantic_Web_Stack



Resource Description Framework (RDF)

- Рамка за описване на ресурсите
- Модел за данни
- Представя синтаксис, който да позволи обмена и използването на информация, съхранявана на различни места
- Въпросът е да се улесни четенето и правилното използване на информация от компютри, а не непременно от хора

Идея на RDF 1/2

- Resource Description Framework (RDF) – език за представяне на информация за метаданни за **ресурси в Уеб** (напр. име, автор, дата на създаване)
- Генерализация на концепцията **Web resource** - RDF може да представи информация на ресурси, които да се идентифицират в Уеб, дори и когато *те не могат да бъдат директно извлечени* в Уеб
- Предназначени за обработка от приложения, но не за представяне пред потребители

Идея на RDF 2/2

- RDF служи за:
 - идентифициране на неща (обекти и субекти), използващи Уеб идентификатори (URI адреси), и
 - описание на ресурси чрез прости свойства и стойности.
- Това позволява чрез RDF да се представят прости твърдения за ресурси като граф от възли и дъги, представляващи ресурси, както и техните свойства и стойности.

„*Things with properties having values*“

RDF описва: неща (**things**)
имащи свойства (**properties**)
и техните стойности (**values**) –
ресурси, описвани чрез твърдения
(**statements**):

- **http://www.me-xml.edu/index.html has a creator whose value is Boyan Bontchev**
- **http://www.me-xml.edu/index.html has a creation-date whose value is November 21, 2019**
- **http://www.me-xml.edu/index.html has a language whose value is Bulgarian**

Субект-предикат-обект

(*Subject-Predicate-Object*)

http://www.my-xml.edu/index.html has a creator whose value is Boyan Bontchev

- Субект (**subject**) е описаното нещо и се задава чрез URL **http://www.my-xml/index.html**
- Предикат (**predicate**) е свойството/характеристиката на субекта – в случая "creator"
- Обект (**object**) е стойността - "Boyan Bontchev"

RDF компоненти

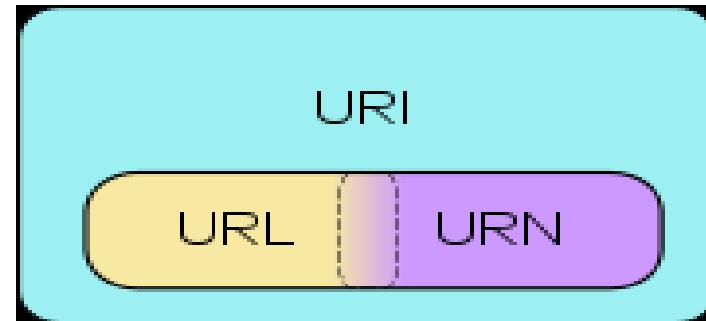
- Формален модел на данните
- Синтаксис за обмен на данни
- Вид схема (схема-модел)
- Синтаксис за машинно-разбираеми схеми
- Заявки

Необходимост от URL

- Нужда от система от *машинно-обработваеми идентификатори* - за *идентифициране на обект, предикат, или обект - в твърдение (statement)*, без възможно двусмислие
- Интернет осигурява една по-обща форма на идентификатор за тези цели, наречен URL
- RDF използва URI референции (или URRef), заедно с допълнителен идентификатор за фрагмент в края, като

`http://www.example.org/index.html#section2`

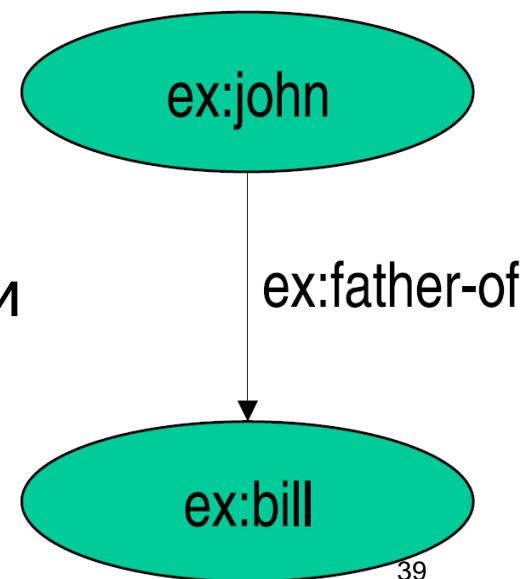
URI, URN, URL



- Uniform Resource Identifier (URI) – стринг за идентификация за име или ресурс в Интернет
- **URI = URL или URN**
- Uniform Resource Name (URN) дефинира идентичността на ресурс
 - *urn:isbn:0-395-36341-1*
- Uniform Resource Locator (URL) предоставя метод за локацията му
 - *http://www.sti-innsbruck.at/*

RDF тройки, представени като граф

- Моделът на тройките от данни може да се представи като граф
- Такъв граф в изкуствения интелект се нарича **семантична мрежа**



- Именовани и насочени графи
 - Възли (Nodes)**: ресурси, литерали
 - Етикети (Labels)**: свойства
 - Крайща (Edges)**: твърдения
(statements)

RDF графи

http://www.example.org/index.html has a creator whose value is John Smith

- **subject –**

http://www.example.org/index.html

http://www.example.org/index.html

- **predicate –**

http://purl.org/dc/elements/1.1/creator

http://purl.org/dc/elements/1.1/creator

- **object –**

http://www.example.org/staffid/85740

http://www.example.org/staffid/85740

Triples нотация

Използва наредена тройка от **subject**, **predicate**, и **object**:

- <<http://www.example.org/index.html>>
<<http://purl.org/dc/elements/1.1/creator>>
<<http://www.example.org/staffid/85740>> .
- <<http://www.example.org/index.html>>
<<http://www.example.org/terms/creation-date>>
„November 10, 2017” .
- <<http://www.example.org/index.html>>
<<http://purl.org/dc/elements/1.1/language>>
"en" .

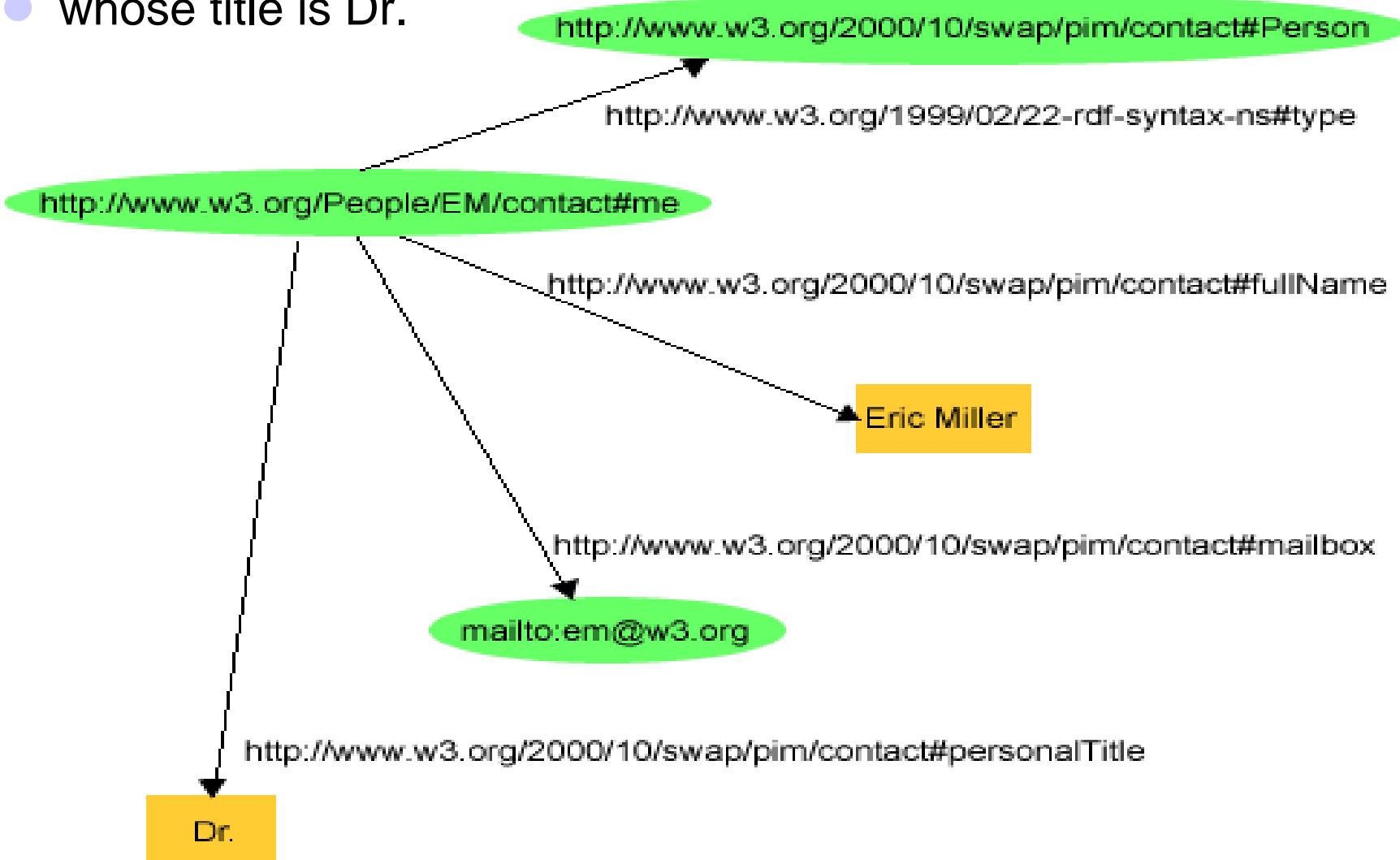
RDF пример 1/2

- Група от твърдения:

- sth.(sb.) is a Person
- identified by the type
<http://www.w3.org/People/EM/contact#me>,
- whose name is Eric Miller,
- whose email address is em@w3.org, and
- whose title is Dr.

RDF пример 2/2

- there is a Person
- identified by <http://www.w3.org/People/EM/contact#me>,
- whose name is Eric Miller,
- whose email address is em@w3.org, and
- whose title is Dr.



Популярни префикси на QName

- Префикс **rdf:**, пространство от имена с URI:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- Префикс **rdfs:**, пространство от имена с URI:
<http://www.w3.org/2000/01/rdf-schema#>
- Префикс **dc:**, пространство от имена с URI:
<http://purl.org/dc/elements/1.1/>
- Префикс **owl:**, пространство от имена с URI:
<http://www.w3.org/2002/07/owl#>
- Префикс **xsd:**, пространство от имена с URI:
<http://www.w3.org/2001/XMLSchema#>

Използване на префикси

Използвайки

- prefix **ex:**, namespace **URI: http://www.example.org/**
- prefix **exterm:**, namespace **URI: http://www.example.org/terms/** (за термините, използвани от примерната организация),
- prefix **exstaff:**, namespace **URI: http://www.example.org/staffid/** (за идентификатори на персонала в примерната организация)

получаваме

ex:index.html dc:creator exstaff:85740 .

ex:index.html exterm:creation-date "August 16, 2018" .

ex:index.html dc:language "en" .

вместо

<http://www.example.org/index.html>
<http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740> .

<http://www.example.org/index.html>
<http://www.example.org/terms/creation-date> "August 16, 2018" .

<http://www.example.org/index.html>
http://purl.org/dc/elements/1.1/language> "en" .

RDF като формат за запис на информация

RDF твърденията са подобни на други формати за запис на информация, като напр.:

- части от запис или каталог на стоки, описващи ресурси в система за обработка на данни
- редове в проста релационна база данни
- прости твърдения във формалната логика

Структури в RDF?

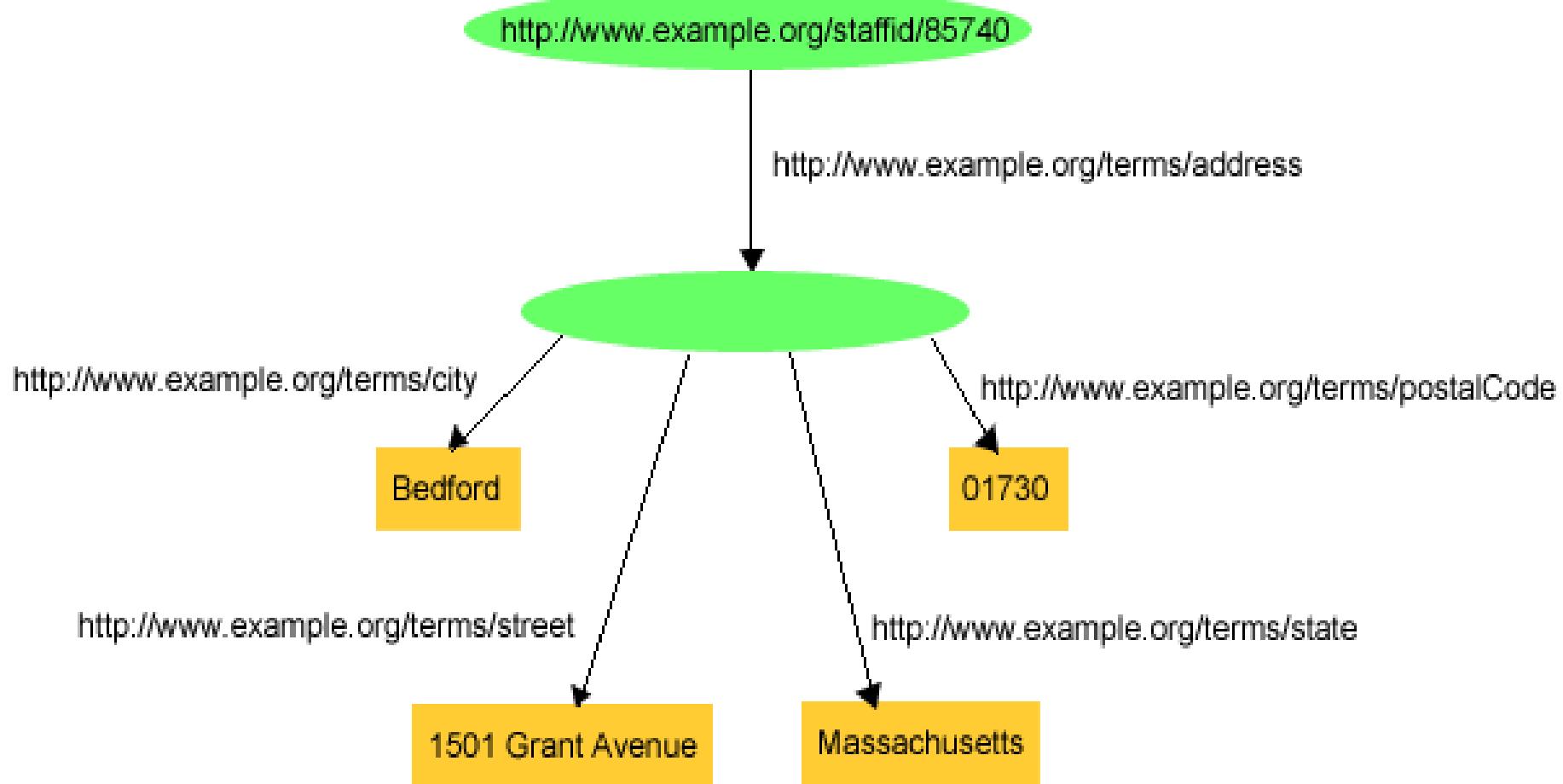
exstaff:12345

exterms:address

„5 J. Baurchier Blv., Sofia 1164, BULGARIA” .

- Как да се изрази в RDF адресът като структура, състояща се от отделни стойности за улица, град, пощенски код и държава?

RDF стойности на структурно свойство – чрез празен (blank) възел



Идентификатори на Blank възел

Тройки с идентификатори за blank възли използват формата `_:name`

```
exstaff:85740 exterms:address _:johnaddress .  
_:johnaddress exterms:street "1501 Grant Avenue" .  
_:johnaddress exterms:city "Bedford" .  
_:johnaddress exterms:state "Massachusetts" .  
_:johnaddress exterms:postalCode "01730" .
```

Литерали

- Обикновени литерали
 - напр. "any text"
 - optionalен маркер за език, напр. "Hello, how are you?"@en-GB
- Типови литерали
 - напр. "hello"^^xsd:string, "1"^^xsd:integer
 - препоръчителни типове данни:
 - XML Schema datatypes
- Явяват се само като обект в тройка, напр.

<<http://example.org/#john>,

<<http://example.org/#hasName>,

"John Smith"^^xsd:string>

XML

<http://example.org/#john>

<http://example.org/#hasName>

"John Smith"^^xsd:string

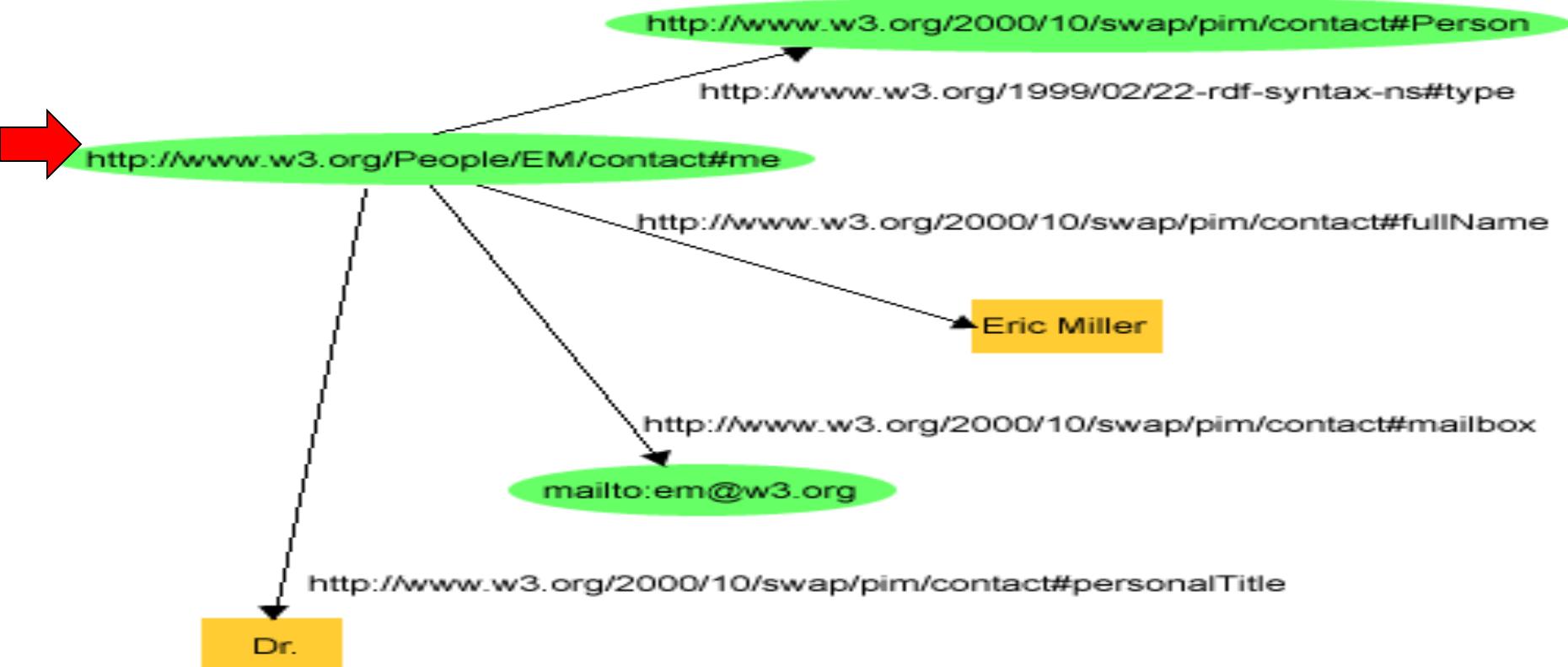
50

Типове данни (Datatypes)

- Един предифиниран тип: **rdf:XMLLiteral**
 - Използван за вграждане на XML в RDF
- Препоръчителни типове данни са XML Schema datatypes, напр.:
 - **xsd:string**
 - **xsd:integer**
 - **xsd:float**
 - **xsd:anyURI**
 - **xsd:boolean**

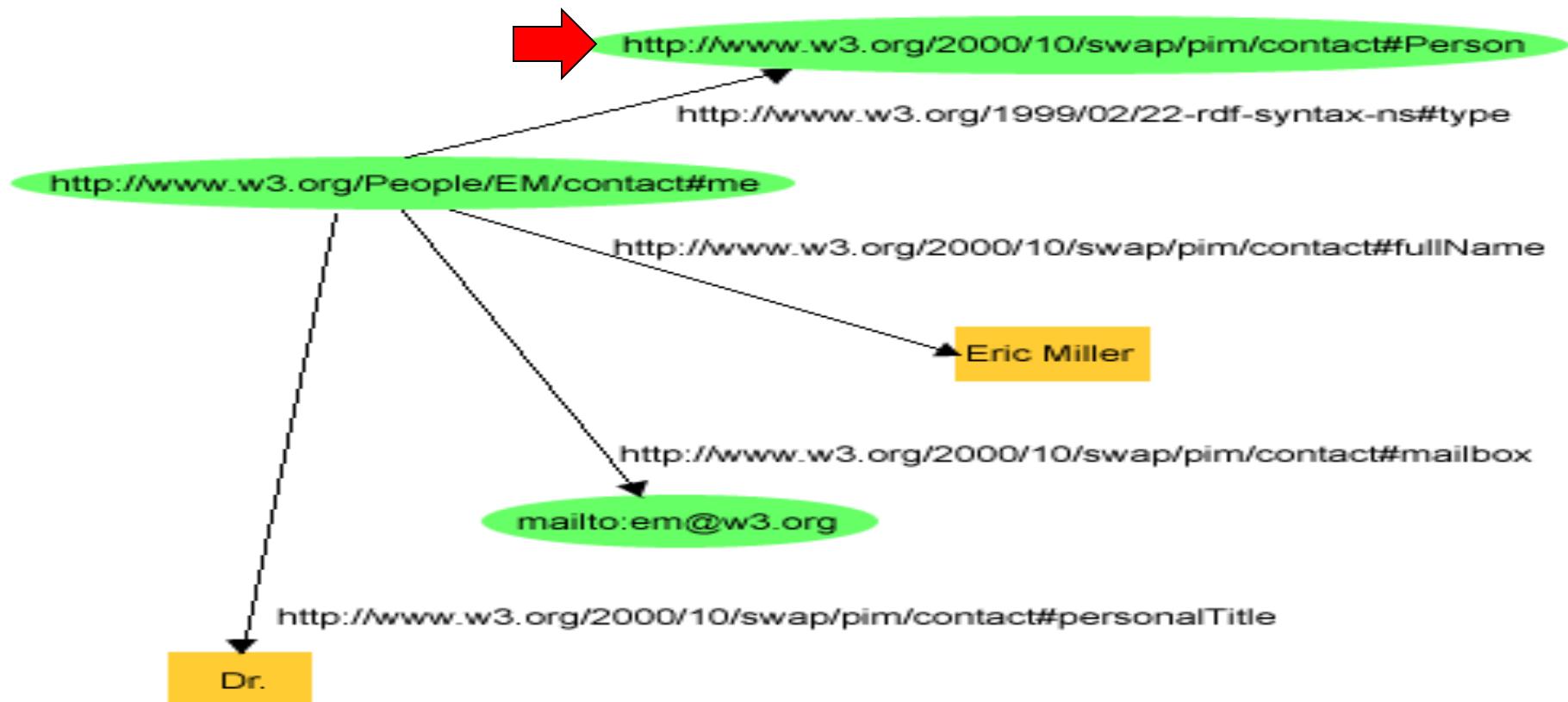
RDF използва URI за идентифициране на:

- **individuals**, напр. Eric Miller, идентифициран чрез <http://www.w3.org/People/EM/contact#me>



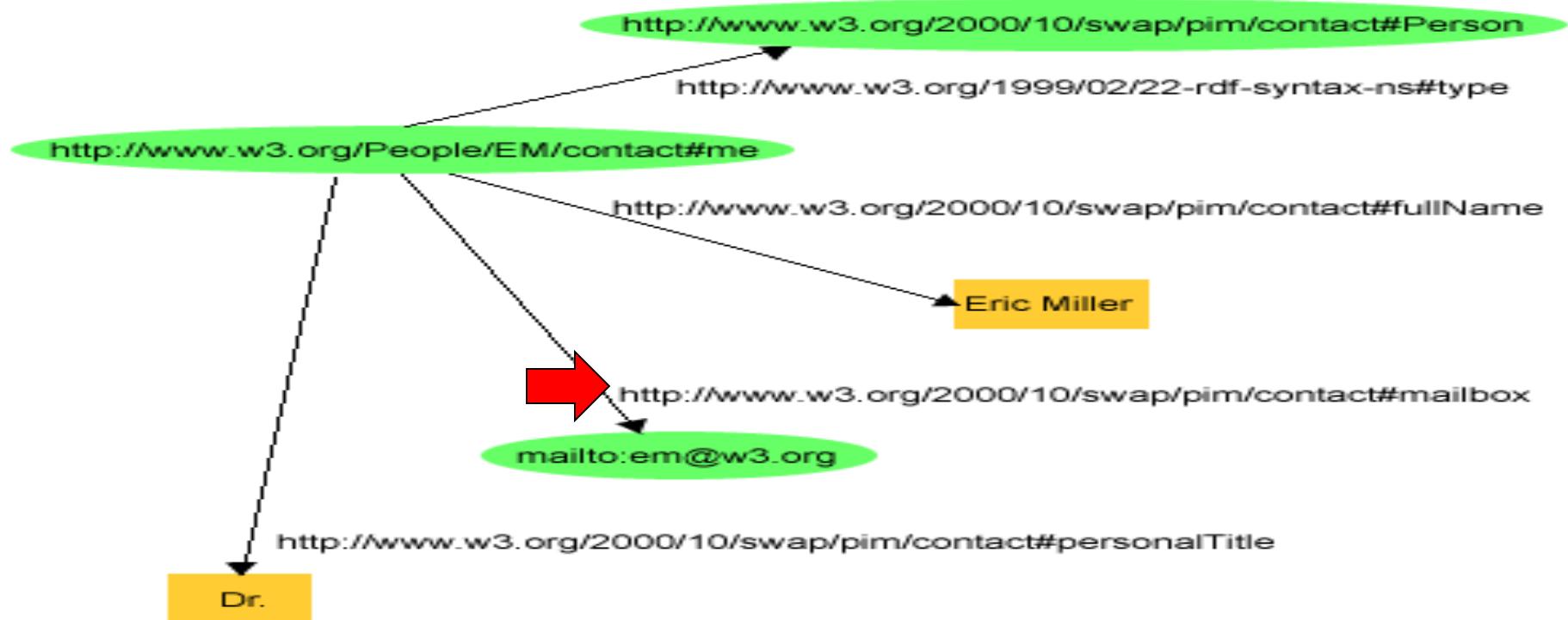
RDF използва URI за идентифициране на:

- **kinds of things**, напр. Person, идентифициран чрез <http://www.w3.org/2000/10/swap/pim/contact#Person>



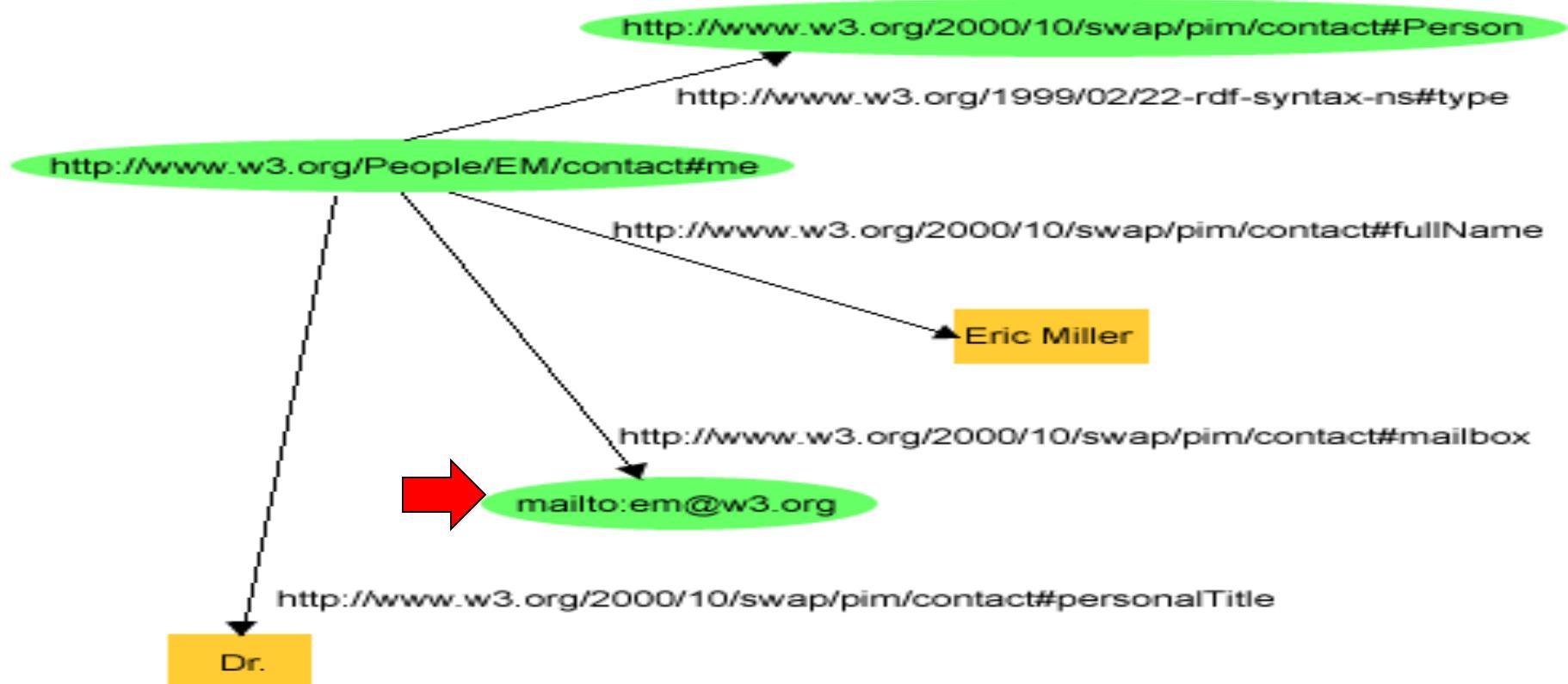
RDF използва URI за идентифициране на:

- **properties of those things**, напр. mailbox,
идентифициран чрез
<http://www.w3.org/2000/10/swap/pim/contact#mailbox>



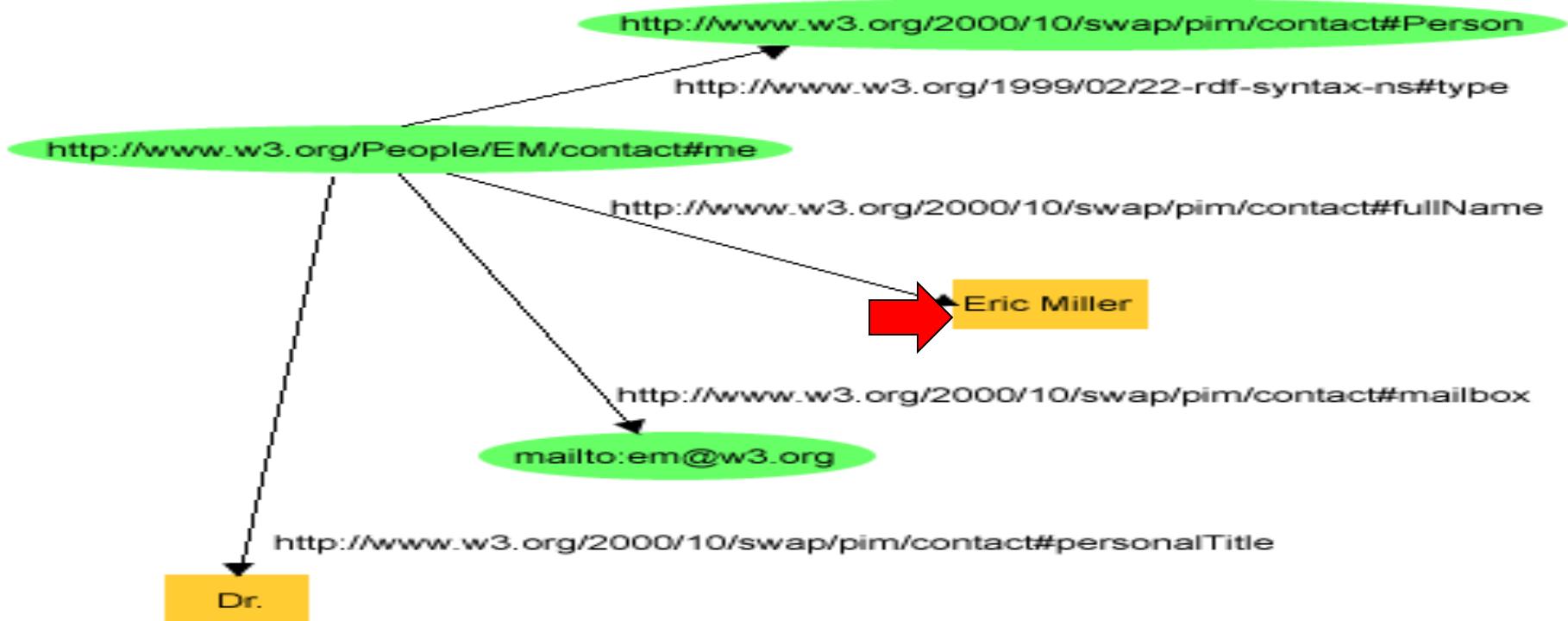
RDF използва URI за идентифициране на:

- values of those properties, напр.
`mailto:em@w3.org` чрез стойност на mailbox свойство



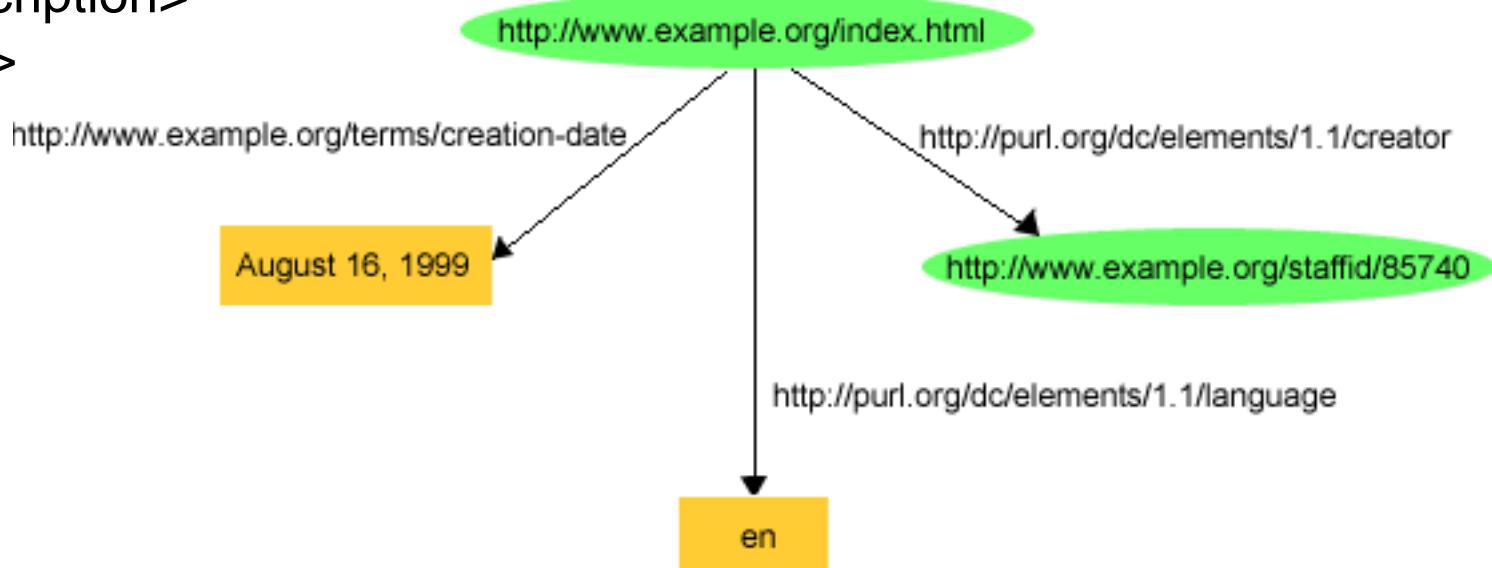
СИМВОЛНИ НИЗОВЕ

- RDF използва също символни низове като **стойностите на свойства**, напр. "**Eric Miller**", и стойности от други типове данни като числа и дати

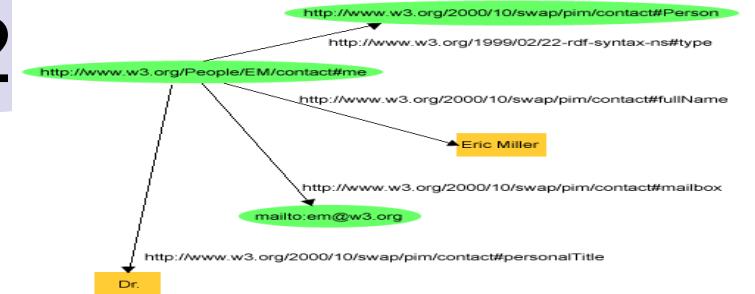


RDF/XML синтаксис за съхранение и обмен на RDF графи 1/2

1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3. xmlns:dc="http://purl.org/dc/elements/1.1/"
4. xmlns:exterm="http://www.example.org/terms/">
5. <rdf:Description rdf:about="http://www.example.org/index.html">
6. <exterm:creation-date>August 16, 1999</exterm:creation-date>
7. <dc:language>en</dc:language>
8. <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
9. </rdf:Description>
10. </rdf:RDF>



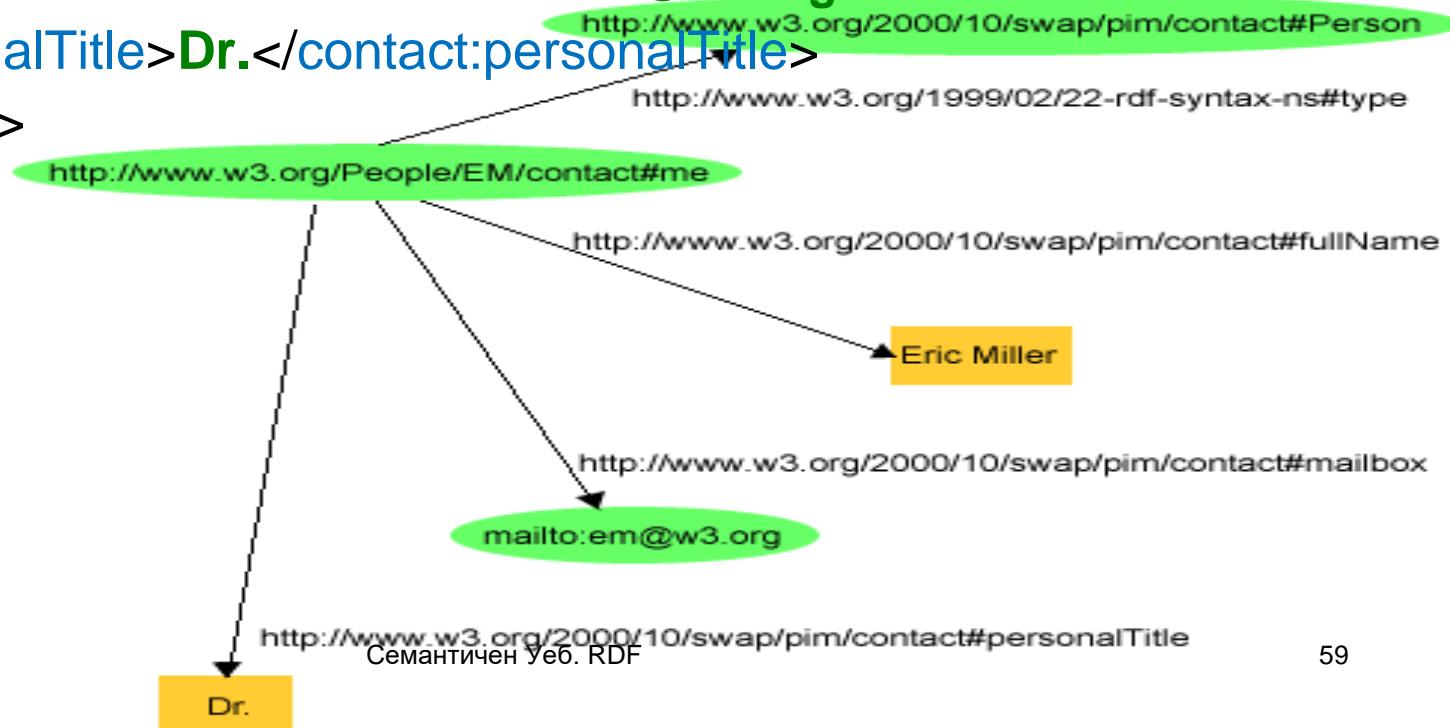
RDF/XML синтаксис за съхранение и обмен на RDF графи 2/2



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person
    rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

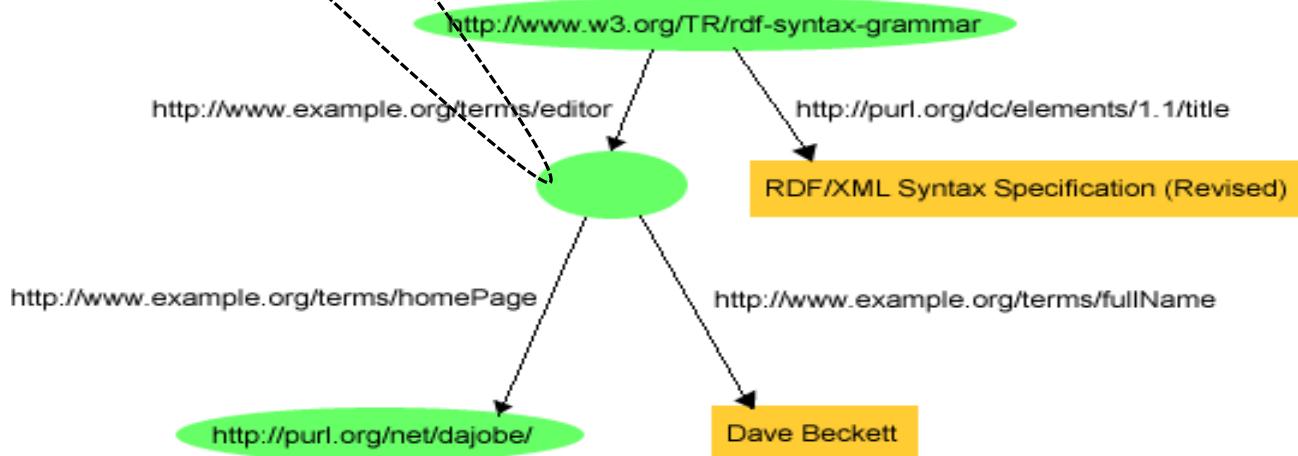
Наредени тройки от субект, предикат и обект

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
    <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
        <contact:fullName>Eric Miller</contact:fullName>
        <contact:mailbox rdf:resource="mailto:em@w3.org"/>
        <contact:personalTitle>Dr.</contact:personalTitle>
    </contact:Person>
</rdf:RDF>
```



Празни възли в RDF/XML граф

1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3. xmlns:dc="http://purl.org/dc/elements/1.1/"
4. xmlns:exterms="http://example.org/stuff/1.0/">
5. <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntaxgrammar">
6. <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
7. <exterms:editor **rdf:nodeID="abc"**>
8. </rdf:Description>
9. <rdf:Description **rdf:nodeID="abc"**>
10. <exterms:fullName>Dave Beckett</exterms:fullName>
11. <exterms:homePage rdf:resource="http://purl.org/net/dajobe/" />
12. </rdf:Description>
13. </rdf:RDF>



RDF/XML с типов литерал и XML единица (ENTITY)

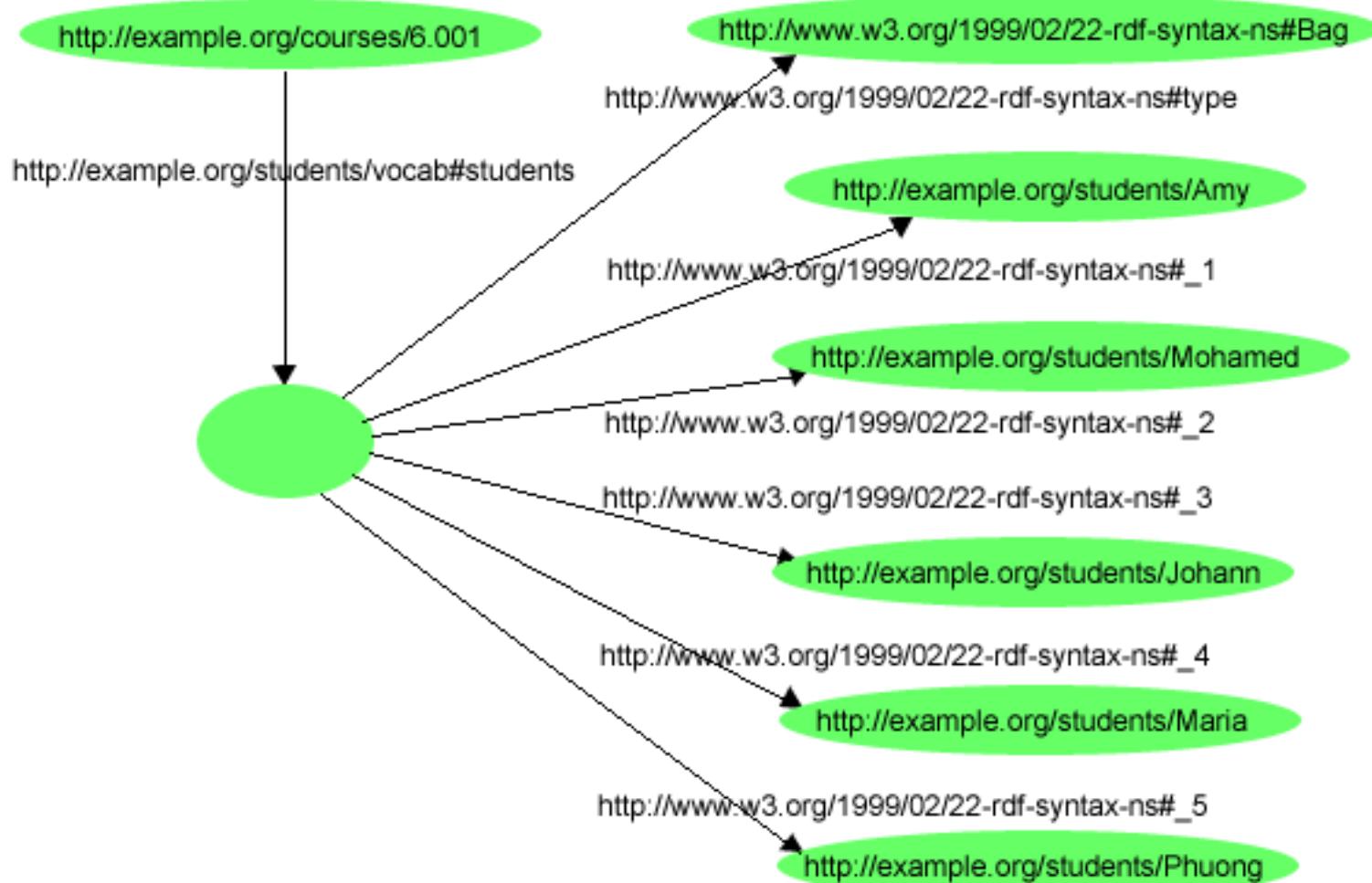
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
 "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:exterm="http://www.xmlcourse.org/terms/">
<rdf:Description
rdf:about="http://www.xmlcourse.org/test2.html">
  <exterm:creation-date rdf:datatype="&xsd;date">
    2019-11-21
  </exterm:creation-date>
</rdf:Description>
</rdf:RDF>
```

RDF контейнери

RDF контейнерът е ресурс, който съдържа членове. Членовете могат да бъдат ресурси (вкл. и празни възли) или литерали. RDF определя три вида контейнери:

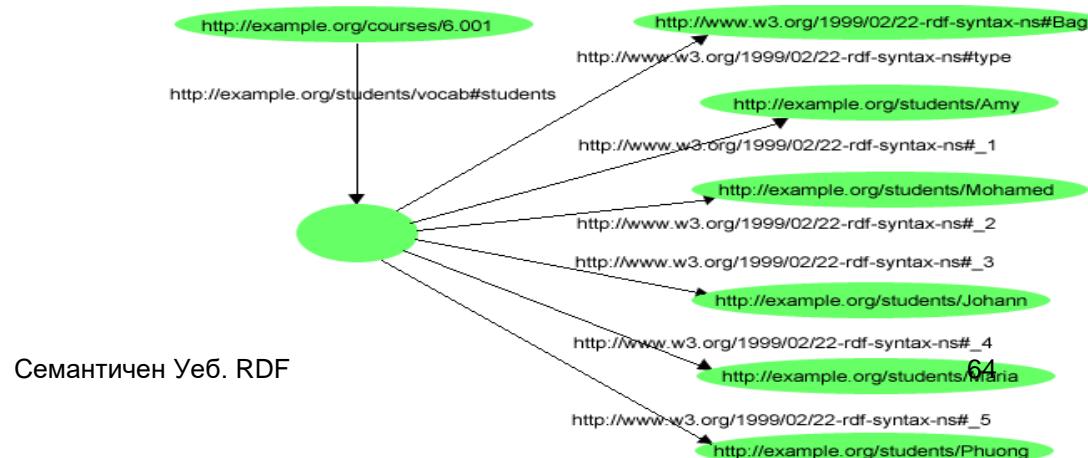
- **rdf:Bag** - група от *неподредени* ресурси или литерали, евентуално дублирани
- **rdf:Seq** - група на *подредени* ресурси или литерали, евентуално дублирани
- **rdf:Alt** - група ресурси или литерали, които представляват алтернативи, напр. списък от алтернативни интернет сайтове. Трябва да съдържа поне една алтернатива; първата е по подразбиране.

Примерен граф на контейнер от тип Bag



Описание на примерен контейнер от тип Bag

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://example.org/students/vocab#">
  <rdf:Description rdf:about="http://example.org/courses/6.001">
    <s:students>
      <rdf:Bagrdf:Bag
```



rdf:Bag описан чрез triples

Да опишем, че *дадена резолюция е одобрена от Комитета по правилата* – с характеристики на Bag контейнер, тоест че:

- комитетът включва *някои* членове (може би и повече!) и
- *редът на описание не е от значение!*

ex:resolution ex:terms:approvedBy ex:rulesCommittee.

ex:rulesCommittee rdf:type rdf:Bag .

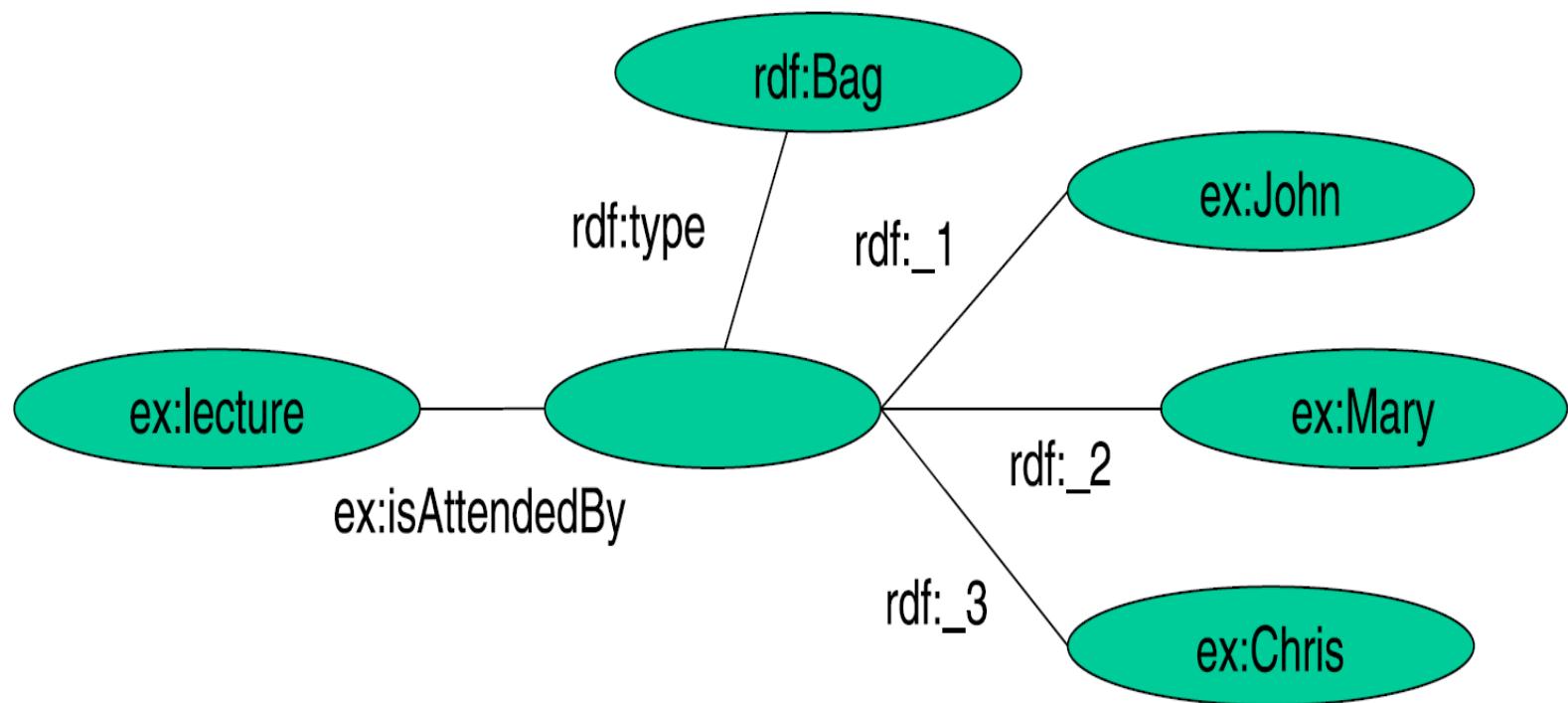
ex:rulesCommittee rdf:_1 ex:Fred .

ex:rulesCommittee rdf:_2 ex:Wilma .

ex:rulesCommittee rdf:_3 ex:Dino .

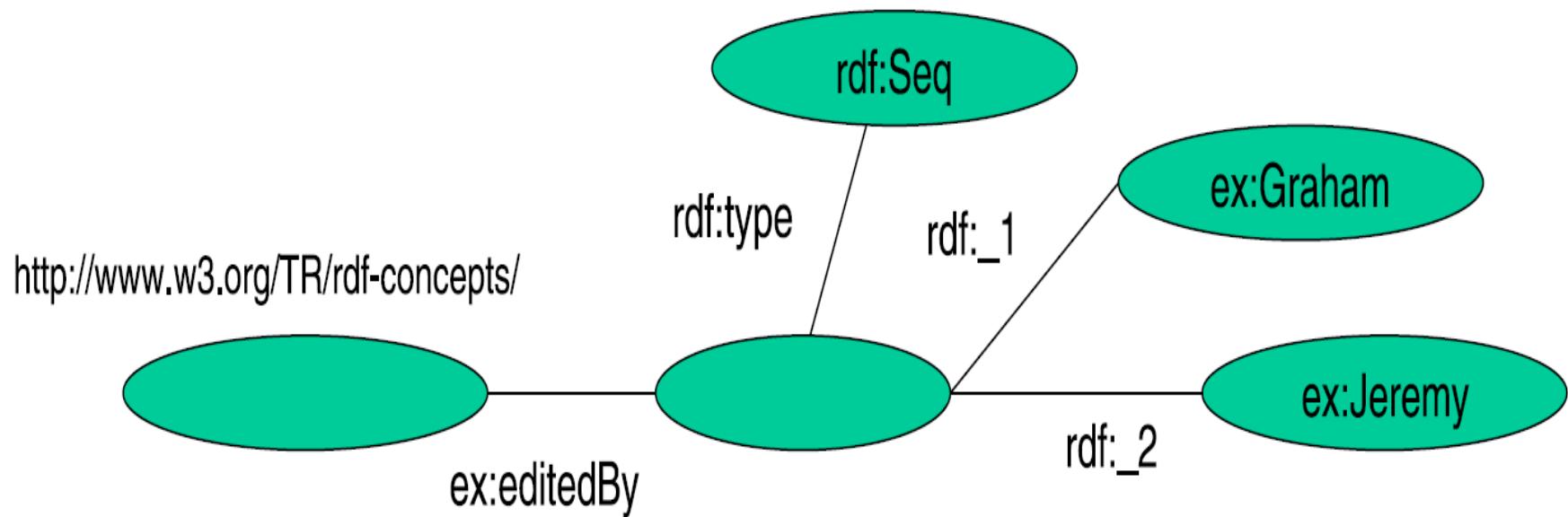
RDF представяне чрез граф на Bag контейнер

“The lecture is attended by John, Mary and Chris”



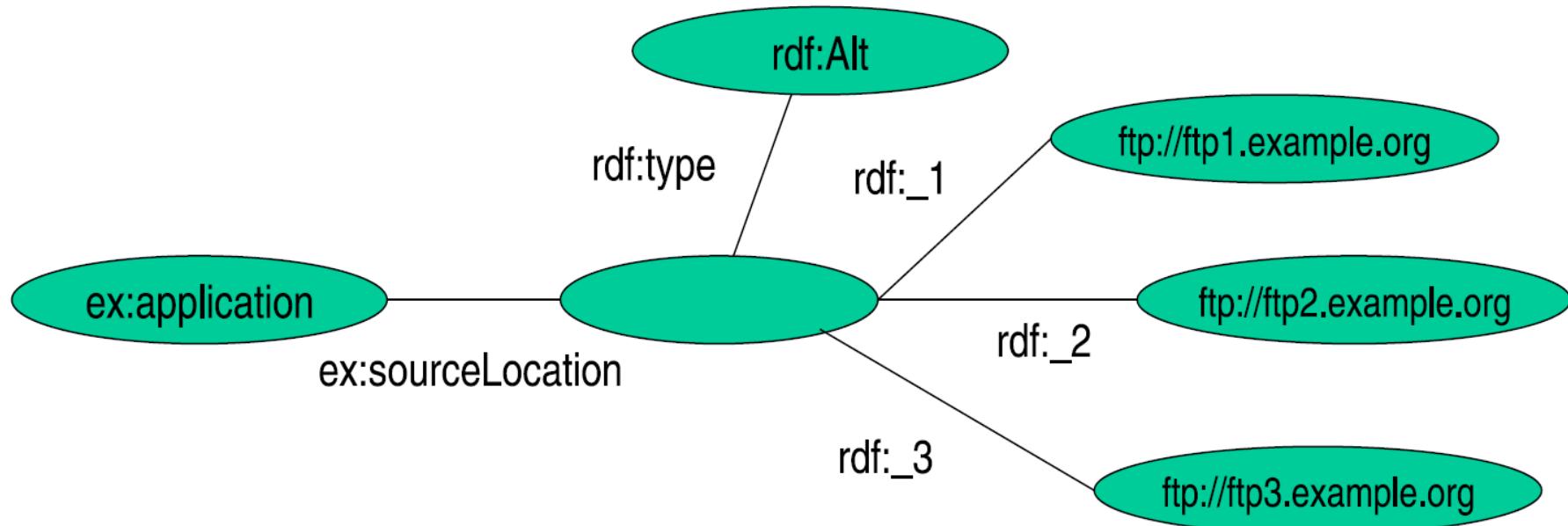
RDF представяне чрез граф на Seq контейнер

“[RDF-Concepts] is edited by Graham and Jeremy”
(в този ред)



RDF представяне чрез граф на Alt контейнер

*“The source code for the application may be found at
ftp1.example.org, **ftp2.example.org**, or
ftp3.example.org”*

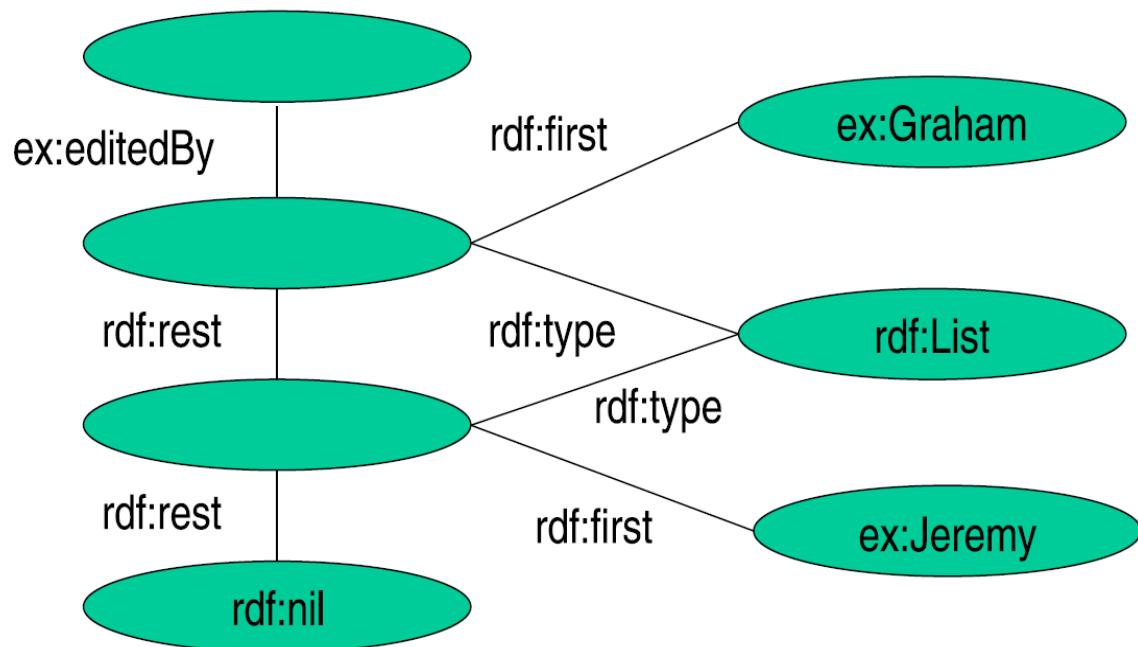


RDF колекции

RDF поддържа описания на групи, съдържащи само определени членове, под формата на RDF колекции.

<http://www.w3.org/TR/rdf-concepts/>

[RDF-Concepts] is edited by Graham and Jeremy (in that order) and nobody else



RDF коллекции – атрибут rdf:parseType="Collection"

```
<?xml version="1.0"?>  
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:cd="http://recshop.fake/cd#">  
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles">  
        <cd:artist rdf:parseType="Collection">  
            <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/George"/>  
            <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/John"/>  
            <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Paul"/>  
            <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Ringo"/>  
        </cd:artist>  
    </rdf:Description>  
</rdf:RDF>
```

Как да описваме RDF твърдения посредством RDF?

- RDF осигурява вграден речник, предназначен за описание на RDF твърдения (*statements*).
- Описание (конкретизация) на твърдение с използване на този речник се нарича *реификация* (*reification*) на RDF твърдението.
- Речникът за RDF реификация се състои от типа **rdf:Statement**, и от свойствата **rdf:subject**, **rdf:predicate** и **rdf:object**.

Пример за реификация

(<http://www.w3.org/2001/sw/RDFCore/TR/WD-rdf-primer-20030117/>)

- За твърдението

`exproducts:item10245 exterms:weight "2.4"^^xsd:decimal .`

- задаваме реификация чрез присвояване на URIref на твърдение като `exproducts:triple12345` и:

`exproducts:triple12345 rdf:type rdf:Statement .`

`exproducts:triple12345 rdf:subject exproducts:item10245 .`

`exproducts:triple12345 rdf:predicate exterms:weight .`

`exproducts:triple12345 rdf:object "2.4"^^xsd:decimal .`

RDF/XML за примера за реификация

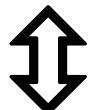
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [ <!ENTITY xsd
 "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:exterms="http://www.example.com/terms/
           xml:base="http://www.example.com/2002/04/products">
<rdf:Description rdf:ID="item10245">
    <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
</rdf:Description>
<rdf:Statement rdf:about="#triple12345">
    <rdf:subject
        rdf:resource="http://www.example.com/2002/04/products#item10245"/>
    <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/>
    <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>
    <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
</rdf:Statement>
</rdf:RDF>
```

Реификацията при представянето на знания

- Реификация: фактически твърдения за други (може би и неверни) твърдения

Mary claims that John's name is "John Smith".

```
<<#myStatement>, rdf:type, rdf:Statement>
<<#myStatement>, rdf:subject, <#john>
<<#myStatement>, rdf:predicate, <#hasName>
<<#myStatement>, rdf:object, "John Smith">
```



```
<<#john>, <#hasName>, "John Smith">
```

RDF речник (Vocabulary)

- RDF дефинира различни ресурси и свойства
- В примерите дотук: `rdf:XMLLiteral`, `rdf:type`, . . .
- RDF речник (vocabulary) е дефиниран чрез пространството от имена:
`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- Класове (Classes):
 - `rdf:Property`, `rdf:Statement`, `rdf:XMLLiteral`
 - `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf>List`
- Свойства (Properties):
 - `rdf:type`, `rdf:subject`, `rdf:predicate`, `rdf:object`,
 - `rdf:first`, `rdf:rest`, `rdf:_n`
 - `rdf:value`
- Ресурси (Resources):
 - `rdf:nil`

RDF речник - типизация

- Използване на **`rdf:type`**:

`<A, rdf:type, B>`

“A belongs to class B”

- Всички свойства принадлежат на класа **`rdf:Property`**:

`<P, rdf:type, rdf:Property>`

“P is a property”

`<rdf:type, rdf:type, rdf:Property>`

“rdf:type is a property”

RDF формати за сериализация

Съществуват няколко формализирани формати за сериализация на RDF

- RDF/XML
- Turtle
- N3

RDF/XML 1/3

- Сериализиране на RDF за използване в Уеб
 - XML като стандартен формат за обмен на данни:
 - Namespaces (напр. **rdf:type**, **xsd:integer**, **ex:john**)
 - Encoding (напр. UTF8, iso-8859-1)
 - XML Schema (напр. типове данни)
- Използване на съществуващи XML среди:
 - Проверка на синтаксиса (т.е. валидиране по схема)
 - Преобразуване (чрез XSLT)
 - Различни RDF представяния
 - Layout (XHTML)
 - Различни XML-базирани формати
- Парсване (разбор) и представяне в памет + манипулации (DOM / SAX)

RDF/XML 2/3

`<#john, #hasName, "John">`

`<#john, #marriedTo, #mary>`

```
<!ENTITY ex "http://example.org/#">

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example.org#">

    <rdf:Description rdf:about="http://example.org/#john">
        <ex:hasName>John</ex:hasName>
        <ex:marriedTo rdf:resource="#mary"/>
    </rdf:Description>

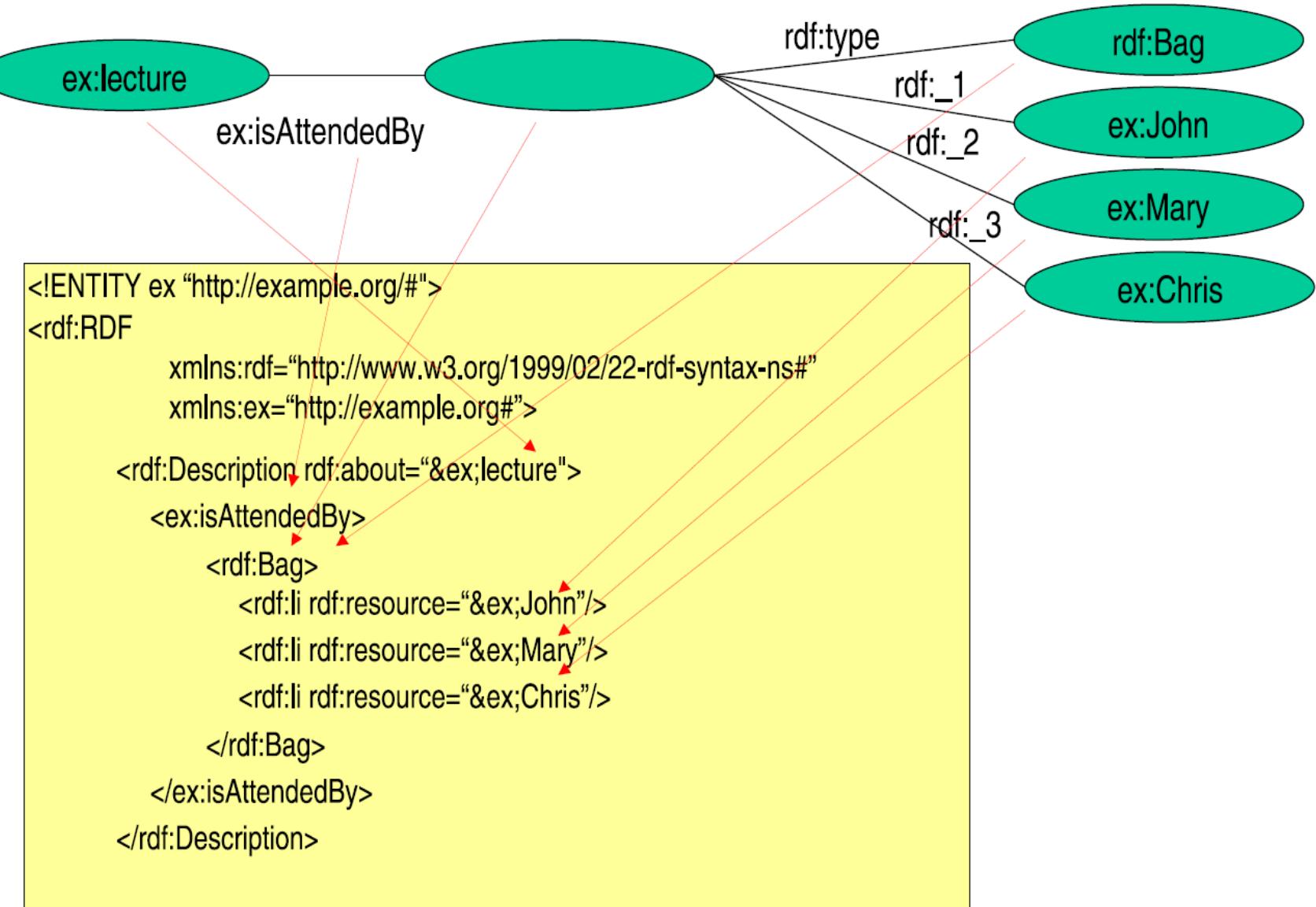
</rdf:RDF>
```

Head

Body

Foot

RDF/XML 3/3





RDF N3 синтаксис

- **Notation3**, известен повече като **N3**, е съкратена **не-XML** сериализация на RDF модели и е ориентиран към потребителя; задава контекстно-свободна граматика
- Много по-компактен и разбираем от XML нотацията за RDF
- Създаден от Tim Berners-Lee и Semantic Web community.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
    <dc:title>Tony Benn</dc:title>
    <dc:publisher>Wikipedia</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

RDF пример
в
XML нотация

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
<http://en.wikipedia.org/wiki/Tony_Benn>
  dc:title "Tony Benn";
  dc:publisher "Wikipedia".
```

RDF пример
в
N3 нотация

RDF N3 примери 1/2



- Обикновено твърдение

`:John :Loves :Mary`

subject, verb and object

- Твърдение с реификация

`{ :John :Loves :Mary } :accordingTo :Bill`

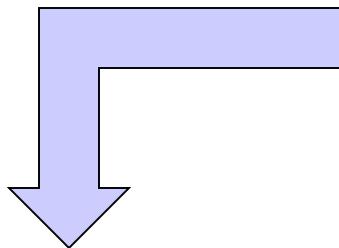
- Твърдение – цел:

`gb:I gb:want { :John :Loves :Mary }`

Префиксът **gb:** задава онтологията S-APL на [Semantic Agent Programming Language \(S-APL\)](#).

RDF N3 примери 2/2

	age	eyecolor
pat	24	blue
al	3	green
jo	5	green



```
<#pat> <#age> 24; <#eyecolor> "blue" .  
<#al> <#age> 3; <#eyecolor> "green" .  
<#jo> <#age> 5; <#eyecolor> "green" .
```

Повече за RDF N3

- <http://www.w3.org/2000/10/swap/Primer.html>
- <http://www.w3.org/2000/10/swap/Examples.html>
- <http://www.w3.org/DesignIssues/Notation3>

Turtle

- Turtle е съкращение от **Terse RDF Triple Language**
- Форма за сериализация на RDF, базирана на най-полезните конструкции на **Notation 3**
- Представяне на тройки от вида
<Subject, Predicate, Object>
- Пример:

```
@prefix person: <http://example/person/> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
person:A foaf:name "Bobsun" .  
person:A foaf:mbox <mailto:bobsun@brey.com> .  
person:B foaf:name "John" .
```

...

Turtle - пример

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix dc: <http://purl.org/dc/elements/1.1/> .  
@prefix ex: <http://example.org/stuff/1.0/> .  
  
<http://www.w3.org/TR/rdf-syntax-grammar>  
    dc:title "RDF/XML Syntax Specification (Revised)" ;  
    ex:editor [  
        ex:fullname "Dave Beckett" ;  
        ex:homePage <http://purl.org/net/dajobe/>  
    ] .
```

За самостоятельна работа: Turtle спрямо N3 и SPARQL

Turtle - Terse RDF Triple Language, Dave Beckett -
[http://www.dajobe.org/2004/01/turtle/2006-12-04/#sec-diff-n3:](http://www.dajobe.org/2004/01/turtle/2006-12-04/#sec-diff-n3)

- Turtle Compared To Notation 3
- Turtle Compared To SPARQL

+

https://en.wikipedia.org/wiki/Notation3#Comparison_of_Notation3,_Turtle,_and_N-Triples

RDFa

- RDFa = Resource Description Framework in attributes
- Разширение на RDF за HTML, XHTML, ...
- RDFa 1.1 е W3C препоръка от юни 2012г.
- Цели:
 - вграждане на обогатени метаданни в уеб документи
 - увеличаване на визуални данни в Уеб страниците в указания за машинно четене
 - да се намали разликата между интерпретацията на Уеб страницата от потребителя и програмите

RDFa произход

- В традиционна технология (обикновени HTML страници) има голяма разлика между това, което човека и програмите разбират



On the left, what browsers see. On the right, what humans see.
Семантичен Уеб. RDF

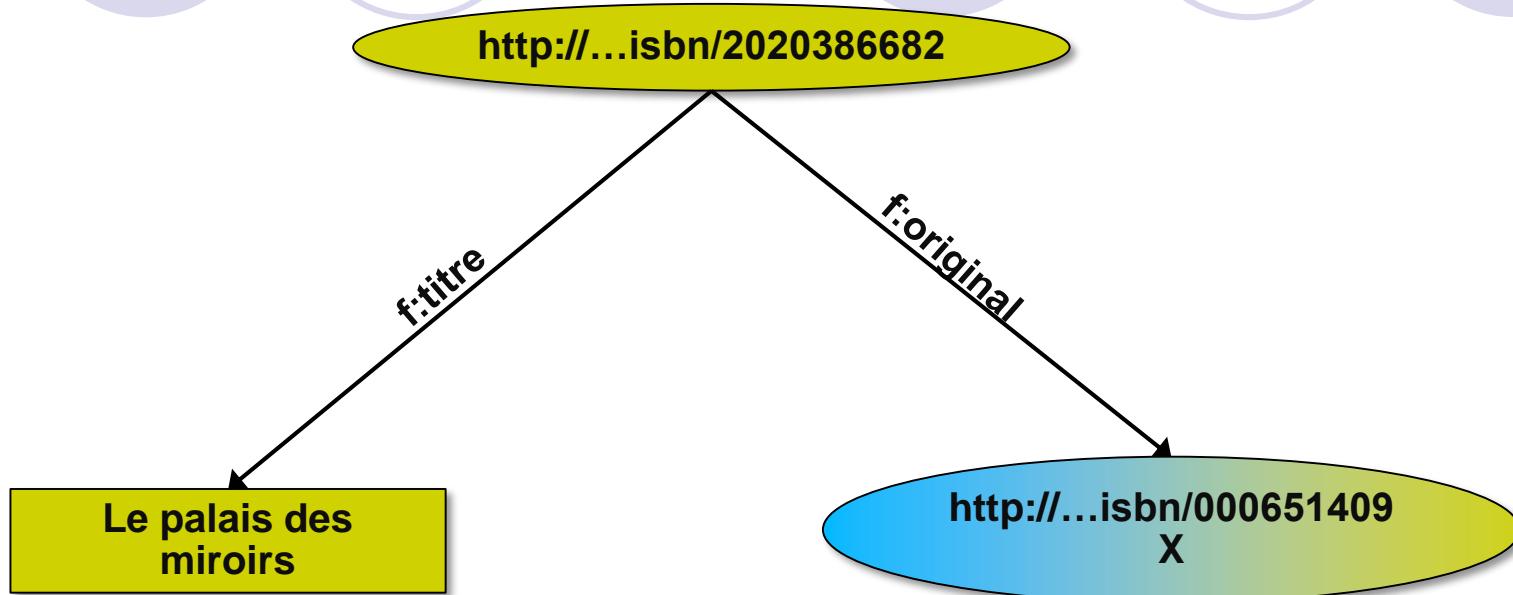
RDFa технология

- RDFa осигурява набор от атрибути, които могат да бъдат използвани за представянето на метаданни в XML (оттук и "a" в RDFa)
- RDFa задава обобщаващи свойства на XHTML meta и link елементите
- Това позволи на потребителя да анотира (пояснява) XHTML маркиране със семантични анотации
- Така RDFa улеснява и подобрява достъпността на Уеб страниците

RDFa набор от атрибути

- `about:` - URI на ресурс определен от метаданни
- `rel:` - специфицира връзка с друг ресурс
- `href, src, resource:` - специфицират партньорски ресурс
- `property:` - специфицира свойство за съдържанието на елемент

RDF пример (в RDF/XML)

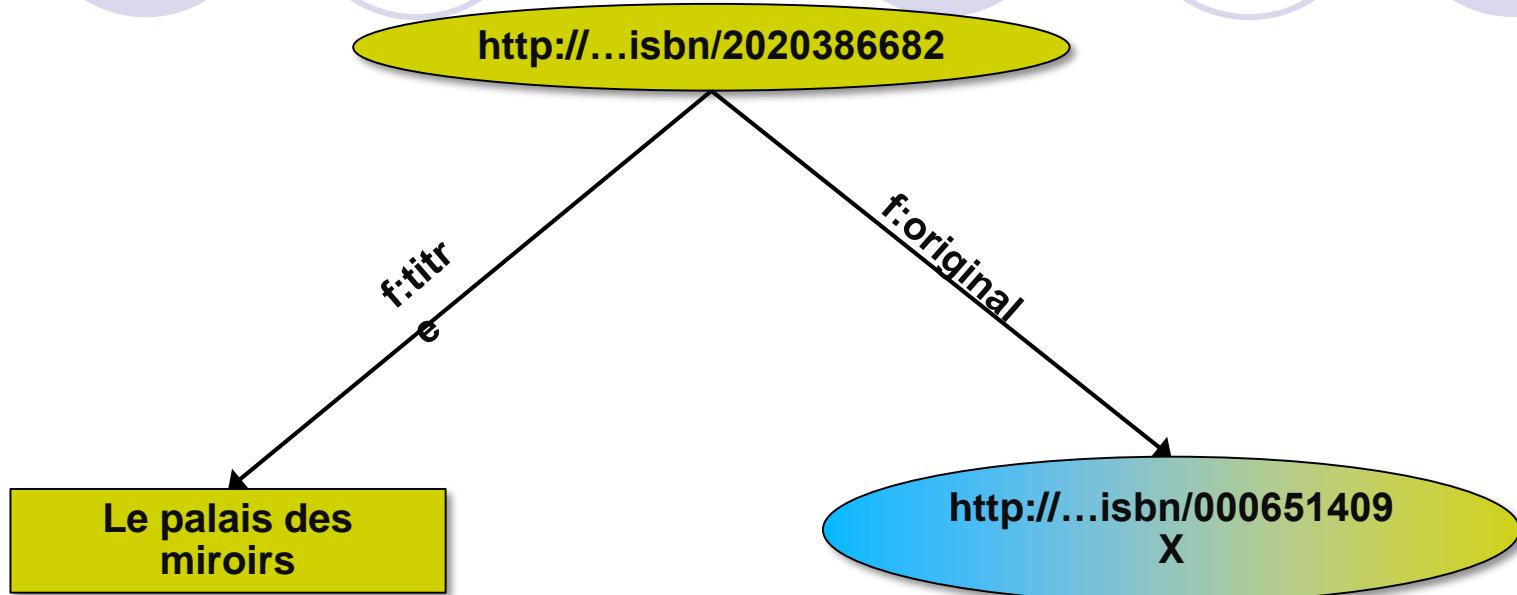


```
<rdf:Description rdf:about="http://.../isbn/2020386682">
  <f:titre xml:lang="fr">Le palais des miroirs</f:titre>
  <f:original rdf:resource="http://.../isbn/000651409X"/>
</rdf:Description>
```

(namespaces се използват за улеснение вместо URI-s)

Източник: Ivan Herman, W3C, 2011

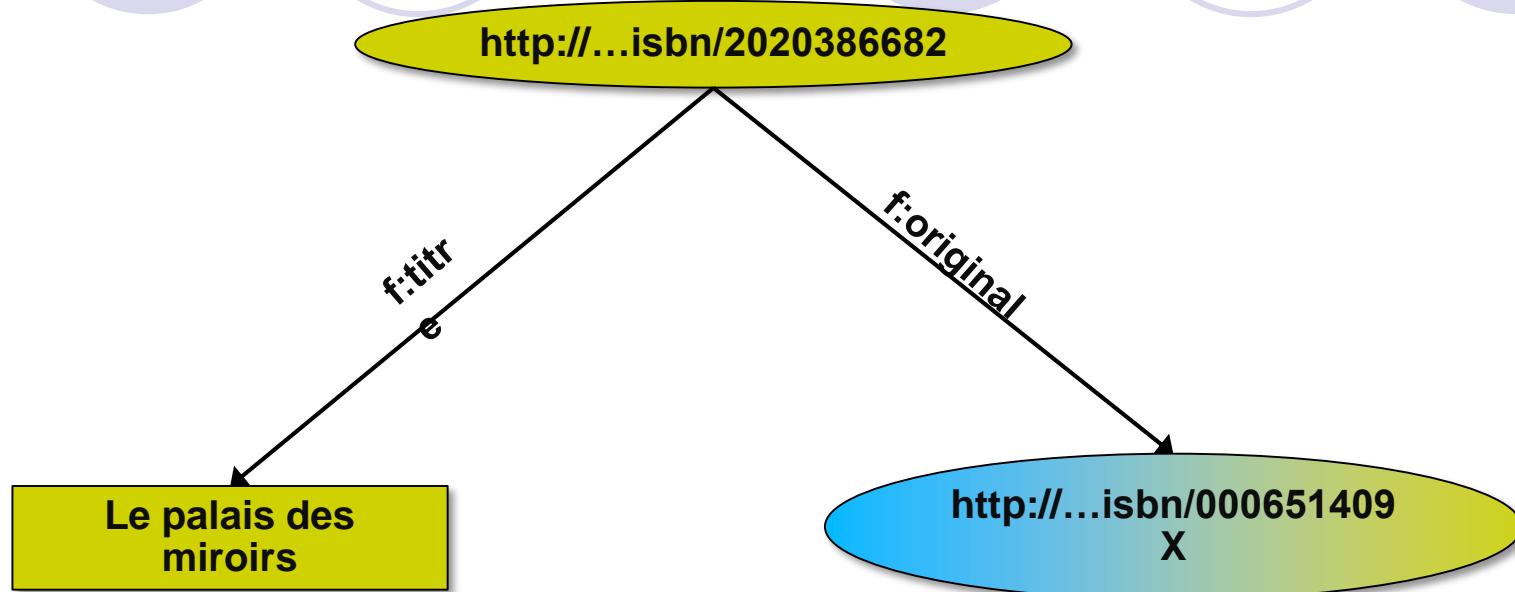
RDF пример (в Turtle)



```
<http://.../isbn/2020386682>
  f:titre "Le palais des miroirs"@fr ;
  f:original <http://.../isbn/000651409X> .
```

Източник: Ivan Herman, W3C, 2011

RDF пример (в RDFa)



```
<p about="http://.../isbn/2020386682">The book entitled  
"⟨span property="f:title" lang="fr"⟩  
Le palais des miroirs</span⟩"  
is the French translation of the  
"⟨span rel="f:original" resource="http://.../isbn/000651409X"⟩  
Glass Palace</span⟩"</p> .
```

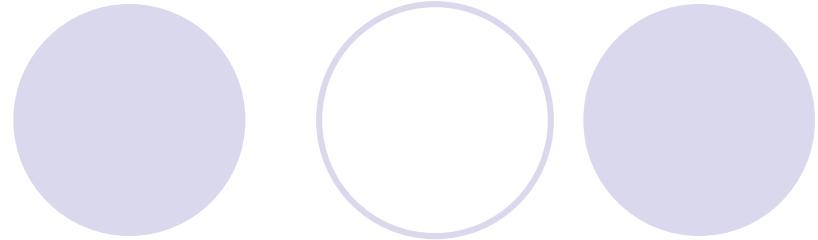
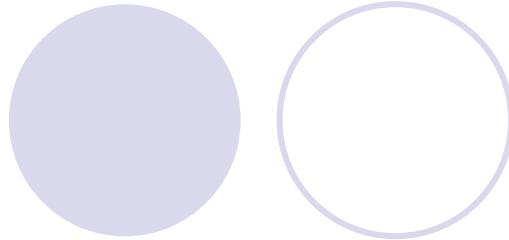
RDF - заключение

Предимства:

- Повторно използване на съществуващите стандарти / инструменти
- Осигурява някаква степен на свобода (напр. чрез контейнери)
- Стандартен формат

Недостатъци:

- Многословен
- Нетривиална реконструкция на RDF граф.



RDF речници

RDF Schema (RDFS)

Ресурс, клас, свойство

Домейн и обхват

Примери



Интегриране на данни 1/2

- Пример:

- “The author of a document is Paul”
- “Paul is the author of a document”
- “A document is authored by Paul”
- “The **author** of a **document** is Paul”

- Представяне в XML:

```
<author>
  <url> http://doc_url </url>
  <name> Paul </name>
```

```
<document>
  <author>
    <name> Paul </name>
  </author>
  <url> http://doc_url </url>
</document>
```

```
<document
  href = "http://doc_url"
  author = "Paul"
/>
```

Интегриране на данни 2/2

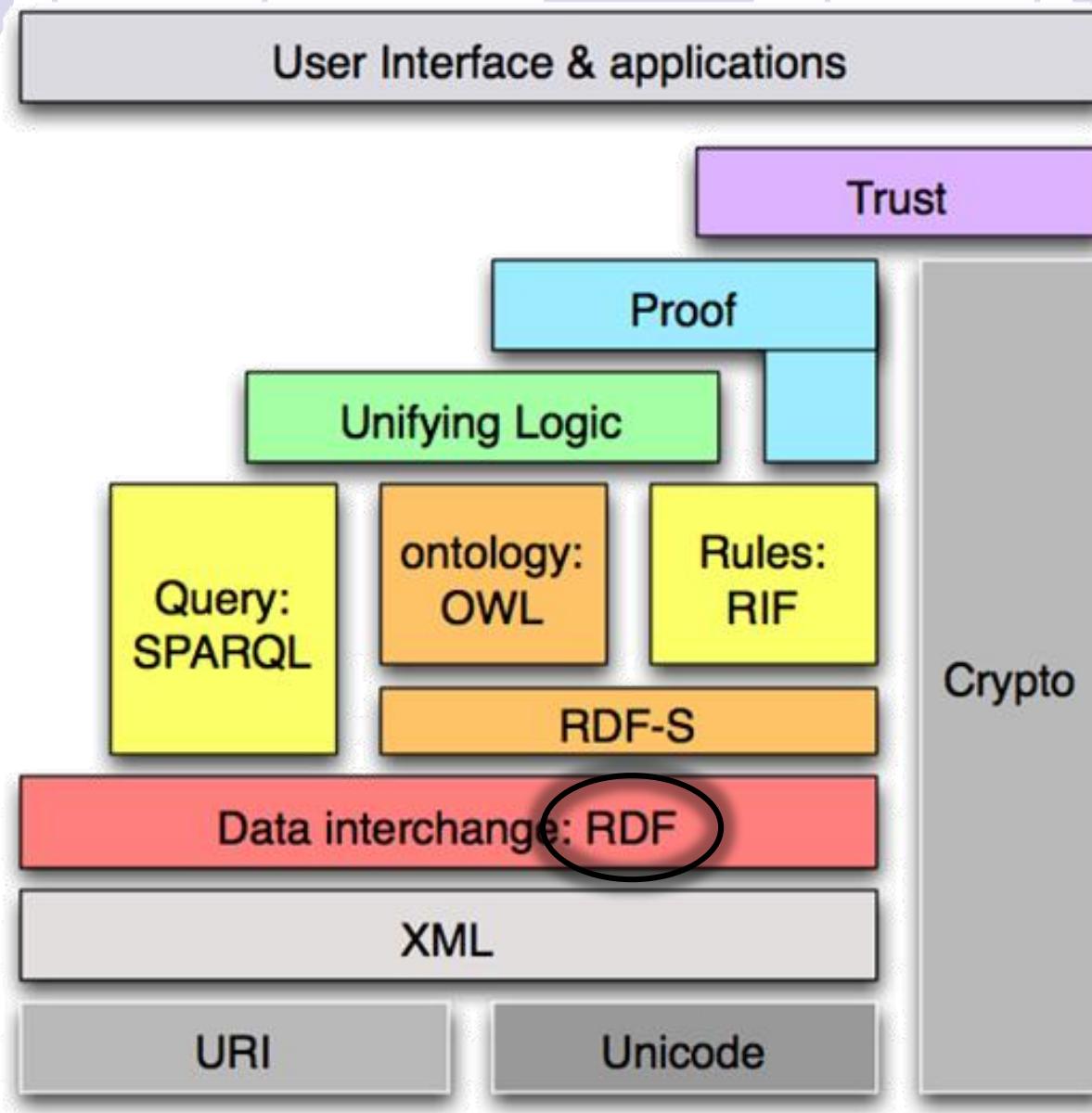
- Сложност на извличане на информация от XML документи:
 - Тъкмо N различни начина за представяне на XML съдържание чрез логическа структура
 - Изиска се нормализиране на всички възможни представления с цел ефективни заявки
 - Означават едно и също нещо на потребителя, но различни неща за машината
 - RDF е по-малко гъвкав:
 - По-малко гъвкав => по-голяма степен на оперативна съвместимост
 - Консистентен начин за представяне на твърдения

Resource Description Framework (RDF)

- Рамка за описване на ресурсите
- Модел за данни
- Представя синтаксис, който да позволи обмена и използването на информация, съхранявана на различни места
- Въпросът е да се улесни четенето и правилното използване на информация от компютри, а не непременно от хора

Стек на семантическия Уеб (W3C, 2006)

Adapted from http://en.wikipedia.org/wiki/Semantic_Web_Stack



Идея на RDF 1/2

- Resource Description Framework (RDF) – език за представяне на информация за метаданни за **ресурси в Уеб** (напр. име, автор, дата на създаване)
- Генерализация на концепцията **Web resource** - RDF може да представи информация на ресурси, които да се идентифицират в Уеб, дори и когато *те не могат да бъдат директно извлечени* в Уеб
- Предназначени за обработка от приложения, но не за представяне пред потребители

Идея на RDF 2/2

- RDF служи за:
 - идентифициране на неща, използващи Уеб идентификатори (URI адреси), и
 - описание на ресурси чрез прости свойства и стойности.
- Това позволява чрез RDF да се представят прости твърдения за ресурси като граф от възли и дъги, представляващи ресурси, както и техните свойства и стойности.

„Things with properties having values“

- RDF описва: неща (**things**) имащи свойства (**properties**) и техните стойности (**values**) – ресурси, описвани чрез **statements**:
 - **http://www.me-xml.edu/index.html has a creator whose value is Boyan Bontchev**
 - **http://www.me-xml.edu/index.html has a creation-date whose value is November 21, 2019**
 - **http://www.me-xml.edu/index.html has a language whose value is Bulgarian**

Субект-предикат-обект (Subject-Predicate-Object)

http://www.example.org/index.html has a
creator whose value is **John Smith**

- Субект (**subject**) е описаното нещо и се задава чрез URL
http://www.example.org/index.html
- Предикат (**predicate**) е свойството/характеристиката на субекта – в случая "**creator**"
- Обект (**object**) е стойността "**John Smith**"

RDF графи

http://www.example.org/index.html has a creator whose value is John Smith

- **subject –**
http://www.example.org/index.html
 - **predicate –**
http://purl.org/dc/elements/1.1/creator
 - **object –**
http://www.example.org/staffid/85740
-
- ```
graph TD; subgraph Triple [RDF Triple]; S1([http://www.example.org/index.html]) -- "http://purl.org/dc/elements/1.1/creator" --> O1([http://www.example.org/staffid/85740]); end;
```

# Triples нотация

- Използва наредена тройка от **subject**, **predicate**, и **object**:

```
<http://www.example.org/index.html>
 <http://purl.org/dc/elements/1.1/creator>
 <http://www.example.org/staffid/85740> .
```

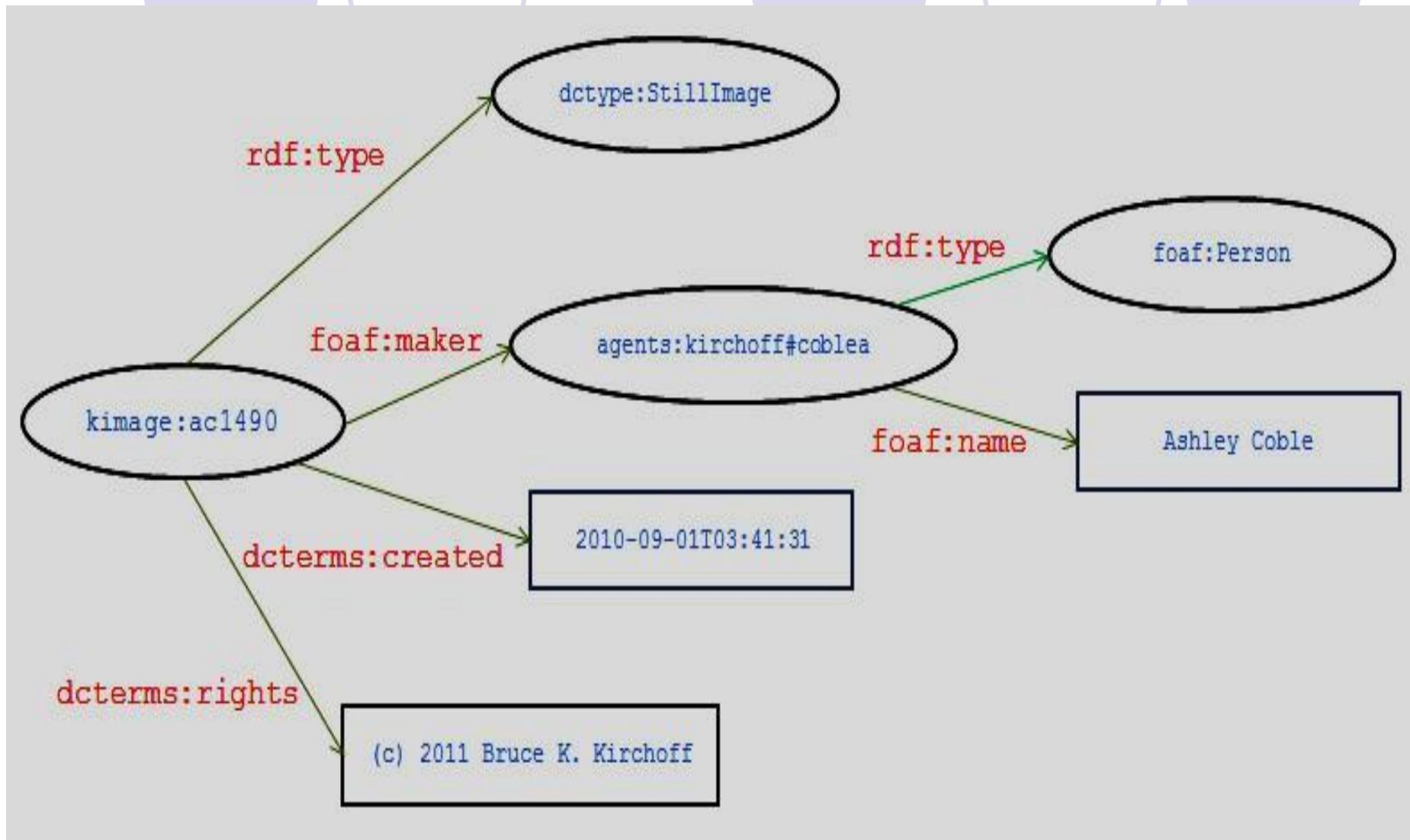
```
<http://www.example.org/index.html>
 <http://www.example.org/terms/creation-date>
 "August 31, 2016" .
```

```
<http://www.example.org/index.html>
 <http://purl.org/dc/elements/1.1/language>
 "en" .
```

# Популярни префикси на QName

- Префикс **rdf:**, пространство от имени с URI:  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- Префикс **rdfs:**, пространство от имени с URI:  
<http://www.w3.org/2000/01/rdf-schema#>
- Префикс **dc:**, пространство от имени с URI:  
<http://purl.org/dc/elements/1.1/>
- Префикс **owl:**, пространство от имени с URI:  
<http://www.w3.org/2002/07/owl#>
- Префикс **xsd:**, пространство от имени с URI:  
<http://www.w3.org/2001/XMLSchema#>

# RDF примерен граф



# RDF сериализация - вариант 1

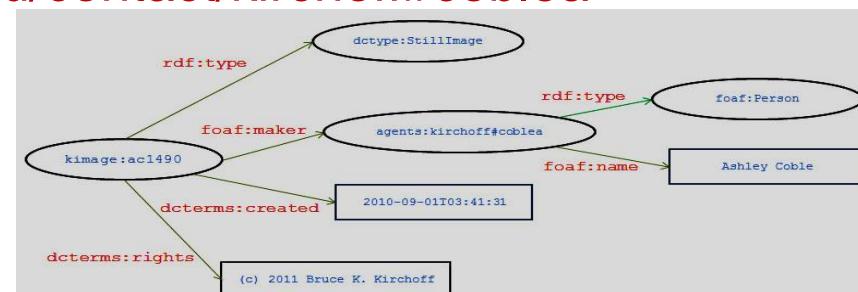
```

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:dcterms="http://purl.org/dc/terms/"
 xmlns:foaf="http://xmlns.com/foaf/0.1/">
 <rdf:Description rdf:about="http://bioimages.vanderbilt.edu/kirchoff/ac1490">
 <rdf:type rdf:resource="http://purl.org/dc/dcmitype/StillImage"/>
 <foaf:maker
 rdf:resource="http://bioimages.vanderbilt.edu/contact/kirchoff#coblea"/>
 <dcterms:created>2010-09-01T03:41:31</dcterms:created>
 <dc:rights>(c) 2011 Bruce K. Kirchoff</dc:rights>
 </rdf:Description>
 <foaf:Person
 rdf:about="http://bioimages.vanderbilt.edu/contact/kirchoff#coblea">
 <foaf:name>Ashley Coble</foaf:name>
 </foaf:Person>
 </rdf:RDF>

```

XML

RDFS



# RDF сериализация – вариант 1 в N3

Short  
hand

stands for

|    |                                                                                                                                         |
|----|-----------------------------------------------------------------------------------------------------------------------------------------|
| a  | < <a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a> >                       |
| =  | < <a href="http://www.w3.org/2002/07/owl#sameAs">http://www.w3.org/2002/07/owl#sameAs</a> >                                             |
| => | < <a href="http://www.w3.org/2000/10/swap/log#implies">http://www.w3.org/2000/10/swap/log#implies</a> >                                 |
| <= | < <a href="http://www.w3.org/2000/10/swap/log#implies">http://www.w3.org/2000/10/swap/log#implies</a> > but in<br>the inverse direction |

@prefix foaf: <<http://xmlns.com/foaf/0.1/>>.

@prefix rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>.

@prefix dc: <<http://purl.org/dc/elements/1.1/>>.

@prefix dcterms: <<http://purl.org/dc/terms/>>.

<<http://bioimages.vanderbilt.edu/contact/kirchoff#coble>>

**a** foaf:Person;

foaf:name "Ashley Coble".

<<http://bioimages.vanderbilt.edu/kirchoff/ac1490>>

dcterms:created "2010-09-01T03:41:31";

dc:rights "(c) 2011 Bruce K. Kirchoff";

**a** <<http://purl.org/dc/dcmitype/StillImage>>;

foaf:maker

<<http://bioimages.vanderbilt.edu/contact/kirchoff#coble>> .

# RDF сериализация - вариант 2

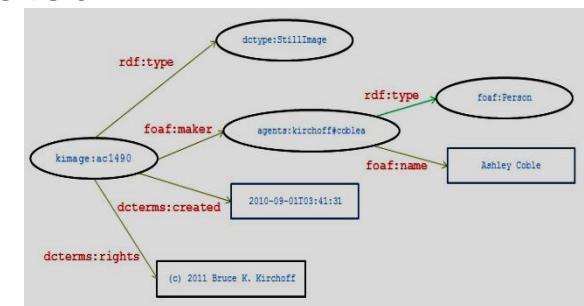
```

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:foaf="http://xmlns.com/foaf/0.1/">
<rdf:Description rdf:about="http://bioimages.vanderbilt.edu/kirchoff/ac1490">
 <rdf:type rdf:resource="http://purl.org/dc/dcmitype/StillImage"/>
 <foaf:maker>
 <foaf:Person
 rdf:about="http://bioimages.vanderbilt.edu/contact/kirchoff#coblea">
 <foaf:name>Ashley Coble</foaf:name>
 </foaf:Person>
 </foaf:maker>
 <dcterms:created>2010-09-01T03:41:31</dcterms:created>
 <dc:rights>(c) 2011 Bruce K. Kirchoff</dc:rights>
 </rdf:Description>
</rdf:RDF>

```

XML

RDFS



# Еквивалентни сериализации

- С неявен **rdf:type**:

```
<foaf:Person
```

```
 rdf:about="http://bioimages.vanderbilt.edu/contact/kirchoff#coblea">
 <foaf:name>Ashley Coble</foaf:name>
 </foaf:Person>
```

- С явно задаване на **rdf:type**:

```
<rdf:Description
```

```
 rdf:about="http://bioimages.vanderbilt.edu/contact/kirchoff#coblea">
 <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
 <foaf:name>Ashley Coble</foaf:name>
 </rdf:Description>
```

# RDF Online Validator

## (<https://www.w3.org/RDF/Validator/>)



## Validation Service

[Skip Navigation](#) [Home](#)  
[Documentation](#)  
[Feedback](#)

### Check and Visualize your RDF documents

[olde servlet](#)

Enter a URI or paste an RDF/XML document into the text field above. A 3-tuple (triple) representation of the corresponding data model as well as an optional graphical visualization of the data model will be displayed.

Check by Direct Input

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
 <rdf:Description rdf:about="http://www.w3.org/">
 <dc:title>World Wide Web Consortium</dc:title>
 </rdf:Description>
</rdf:RDF>
```

#### Display Result Options:

Triples and/or Graph:

Graph format:

Paste an RDF/XML document into the following text field to have it checked. More options are available in the [Extended interface](#).

Check by URI

# RDF компоненти

- Формален модел на данните
- Синтаксис за обмен на данни
- Вид схема (схема модел)
- Синтаксис за машинно-разбираеми схеми
- Протоколи за заявка и профилиране

# RDF схема

- Деклариране на речници
  - свойства, дефинирани от дадено общество
  - характеристики на свойства и/или ограничения върху съответните стойности
- Основни типове на RDF Schema
  - Property, Class, SubClassOf, Domain, Range
  - Минимално, но разширяемо
  - Без колизии с други системи от типове (за XML DTDs)
- Може да се изрази чрез модела и синтаксиса на RDF

# RDF дискриптивен език

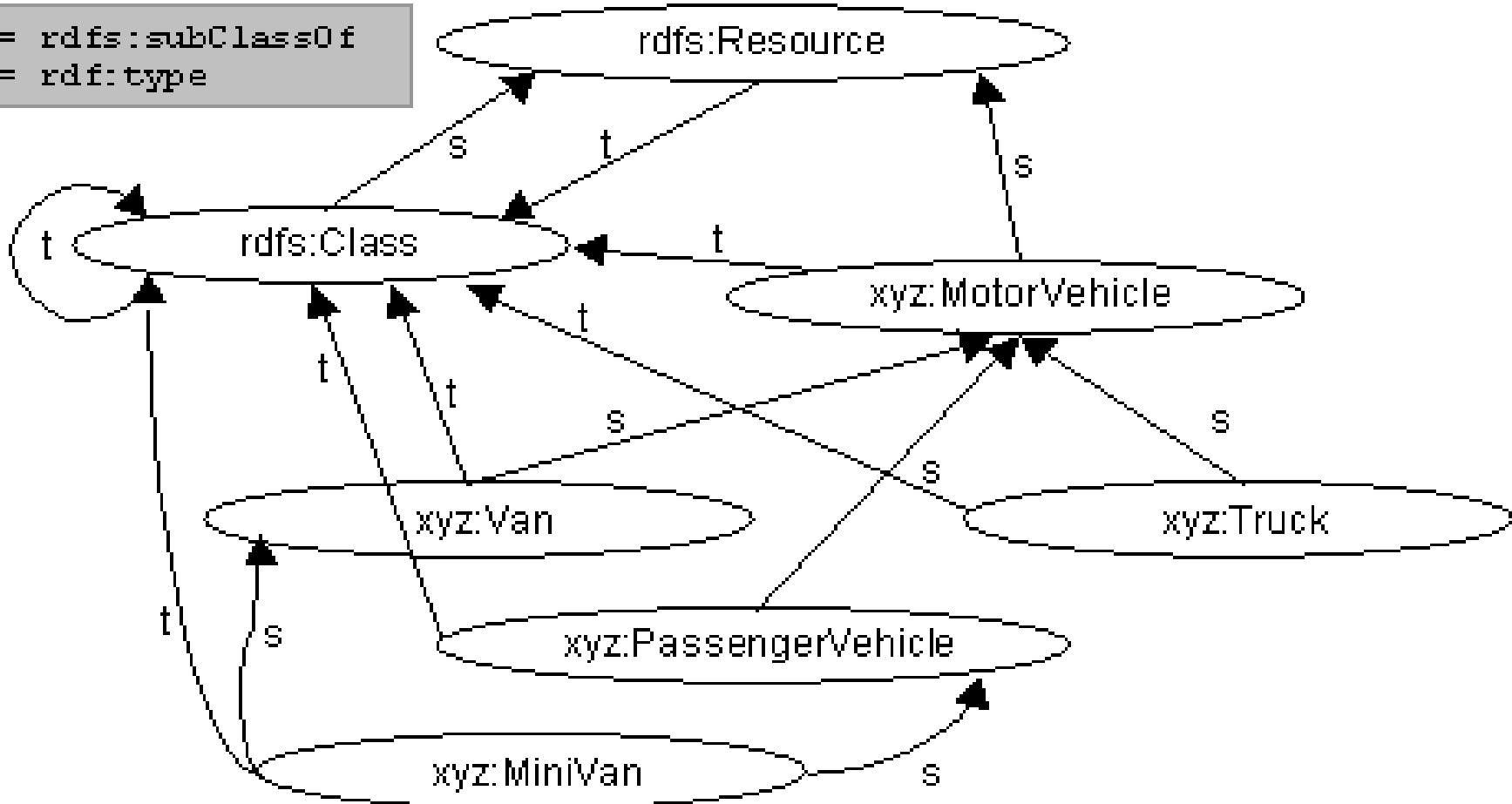
- Типове в RDF:

`<#john, rdf:type, #Student>`

- Какво е “#Student”?
- Нуждаем се от език за описание на RDF типове:
  - за дефиниране на класове:
    - “#Student **is a class**”
  - релации м/у класовете:
    - “#Student **is a sub-class of** #Person”
  - свойства на класове:
    - “#Person **has a property** hasName”
- RDF Schema е именно такъв език

# Примерна йерархия в RDF Schema

```
s = rdfs:subClassOf
t = rdf:type
```



# Представяне в RDF/XML 1/2

```
<rdf:RDF xml:lang="en" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <!-- Note: this RDF schema would typically be used in RDF instance
 data by referencing it with an XML namespace declaration, for example
 xmlns:xyz="http://www.w3.org/2000/03/example/vehicles#". This allows
 us to use abbreviations such as xyz:MotorVehicle to refer
 unambiguously to the RDF class 'MotorVehicle'. -->
 <rdf:Description ID="MotorVehicle">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>
 <rdf:Description ID="PassengerVehicle">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
 </rdf:Description>
```

# Представяне в RDF/XML 2/2

```
<rdf:Description ID="Truck">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description ID="Van">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description ID="MiniVan">
 <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 <rdfs:subClassOf rdf:resource="#Van"/>
 <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```

# Изразителна мощност на RDFS

- Класове:

**<#Student, rdf:type, #rdfs:Class>**

- Йерархии от класове:

**<#Student, rdfs:subClassOf, #Person>**

- Свойства (Properties):

**<#hasName, rdf:type, rdf:Property>**

- Йерархии от свойства:

**<#hasMother, rdfs:subPropertyOf, #hasParent>**

- Асоцииране на свойства с класове (1 от 2):

– “The property **#hasName only applies to #Person**”:

**<#hasName, rdfs:domain, #Person>**

- Асоцииране на свойства с класове (2 от 2):

– “The type of the property **#hasName is #xsd:string**”:

**<#hasName, rdfs:range, xsd:string>**

# RDF речник (Vocabulary)

- Класове:
  - **rdf:Property**, **rdf:Statement**, **rdf:XMLLiteral**
  - **rdf:Seq**, **rdf:Bag**, **rdf:Alt**, **rdf>List**
- Свойства:
  - **rdf:type**, **rdf:subject**, **rdf:predicate**, **rdf:object**
  - **rdf:first**, **rdf:rest**, **rdf:\_n**
  - **rdf:value**
- Ресурси:
  - **rdf:nil**

# RDFS речник

- RDFS разширява RDF речника
- RDFS речникът се дефинира чрез пространството:  
<http://www.w3.org/2000/01/rdf-schema#>

RDFS класове:

- **rdfs:Resource**
- **rdfs:Class**
- **rdfs:Literal**
- **rdfs:Datatype**
- **rdfs:Container**
- **rdfs:ContainerMembershipProperty**

RDFS свойства:

- **rdfs:domain**
- **rdfs:range**
- **rdfs:subPropertyOf**
- **rdfs:subClassOf**
- **rdfs:member**
- **rdfs:seeAlso**
- **rdfs:isDefinedBy**
- **rdfs:comment**
- **rdfs:label**

# RDFS принципи

- **Ресурс**
  - Всички ресурси са неявно екземпляри на **rdfs:Resource**
- **Клас**
  - Описва набор от ресурси
  - Класовете сами по себе си са ресурси – напр. хора, Уеб страници, документи...
- Йерархия от класове се дефинира чрез **rdfs:subClassOf**
- Всеки клас е член на **rdfs:Class**
- **Свойство (Property)**
  - свойствата са подмножество на RDFS Resources
    - **Домейн:** клас, асоцииран със свойство (свойството описва всички екземпляри на класа): **rdfs:domain**
    - **Обхват:** тип на стойностите на свойство: **rdfs:range**
    - Йерархия от свойства се дефинира чрез: **rdfs:subPropertyOf**

# Определяне на екземпляри на клас

1/2

- Ако в една тройка субектът има за свойство (`rdf:type`) обекта (стойността) С, тогава С е клас (`rdfs:Class`) и зададеният в тройката субект е негов екземпляр
- Ако свойството **depicts** е за субекта `img:cat123 foaf:depicts http://img.bilt.edu/12` и имаме `foaf:depicts rdfs:domain foaf:Image`, то следва, че `img:cat123 rdf:type foaf:Image`
- Ако свойството Т е дефинирано с обхват (range) С и ресурсът О е обект в тройка с предикат Т, то следва, че **O rdf:type C**

# Определяне на екземпляри на клас

## 2/2

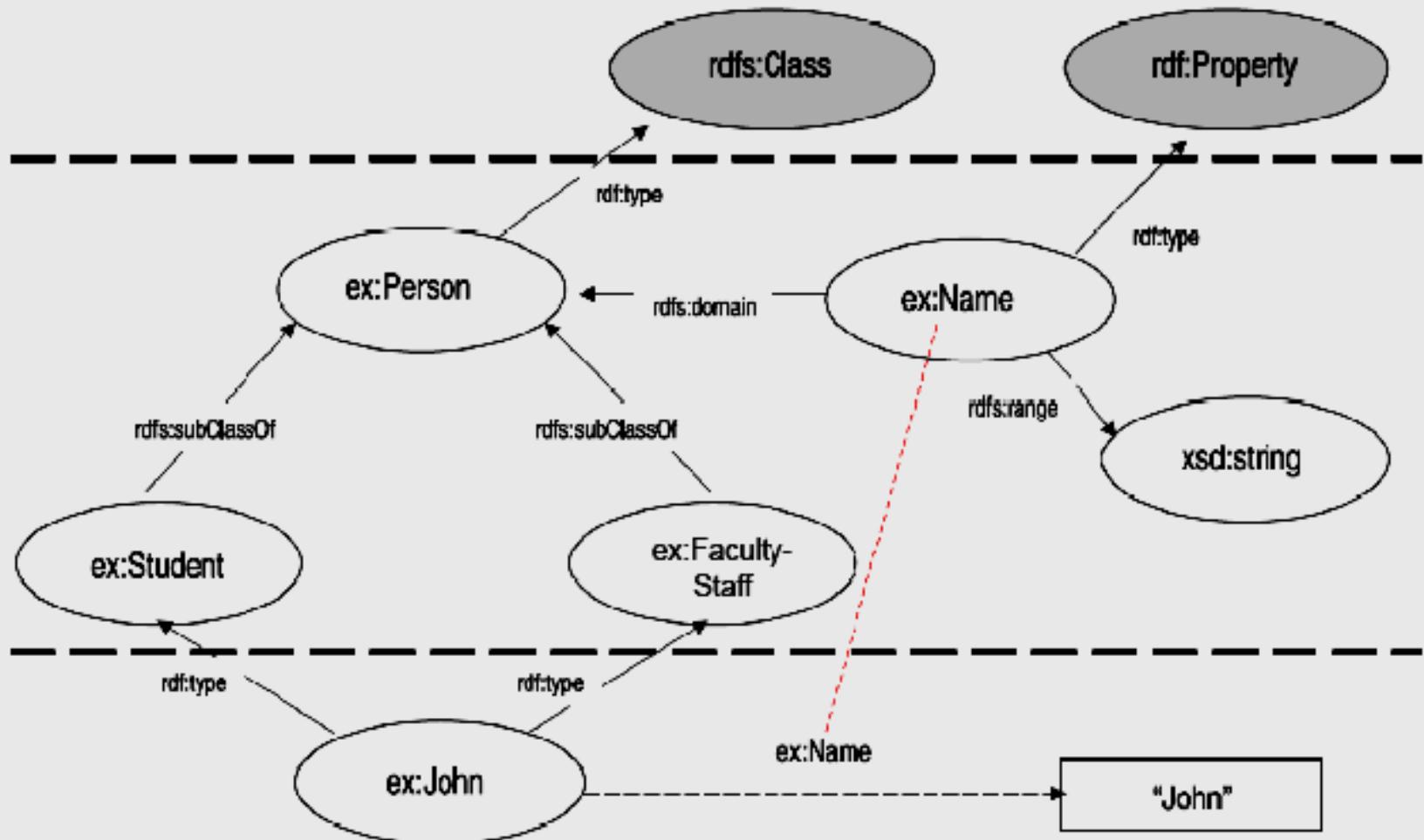
- Задаване на екземпляр на клас, указан чрез името на XML елемента на контейнера (виж слайд 17)
- Ако **foaf:Image rdfs:subClassOf foaf:Document** и още **kimage:ac1481 rdf:type foaf:Image**, тогава е вярно и **kimage:ac1481 rdf:type foaf:Document**
- Един ресурс може да е екземпляр на няколко класа

# Пример ([https://en.wikipedia.org/wiki/RDF\\_Schema](https://en.wikipedia.org/wiki/RDF_Schema))

| In English                                                                                                                                                             | The graph                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>Dog1 is an animal</li><li>Cat1 is a cat</li><li>Cats are animals</li><li>Zoos host animals</li><li>Zoo1 hosts the Cat2</li></ul> | <p>The graph illustrates the RDF triples corresponding to the English statements. Entities are represented as blue ovals, and predicates as arrows. <br/>ex:dog1 → rdf:type → ex:animal (green arrow)<br/>ex:cat1 → rdf:type → ex:cat (green arrow)<br/>ex:cat → rdfs:subClassOf → ex:animal (orange arrow)<br/>ex:zool → rdfs:range → ex:animal (orange arrow)<br/>ex:zool → zoo:host → ex:cat2 (orange arrow)<br/>ex:cat2 ← zoo:host → ex:zool (orange arrow)<br/>Legend: <b>RDF special terms</b> (green box), <b>RDFS special terms</b> (orange box)</p> |
| <b>RDF/turtle</b>                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

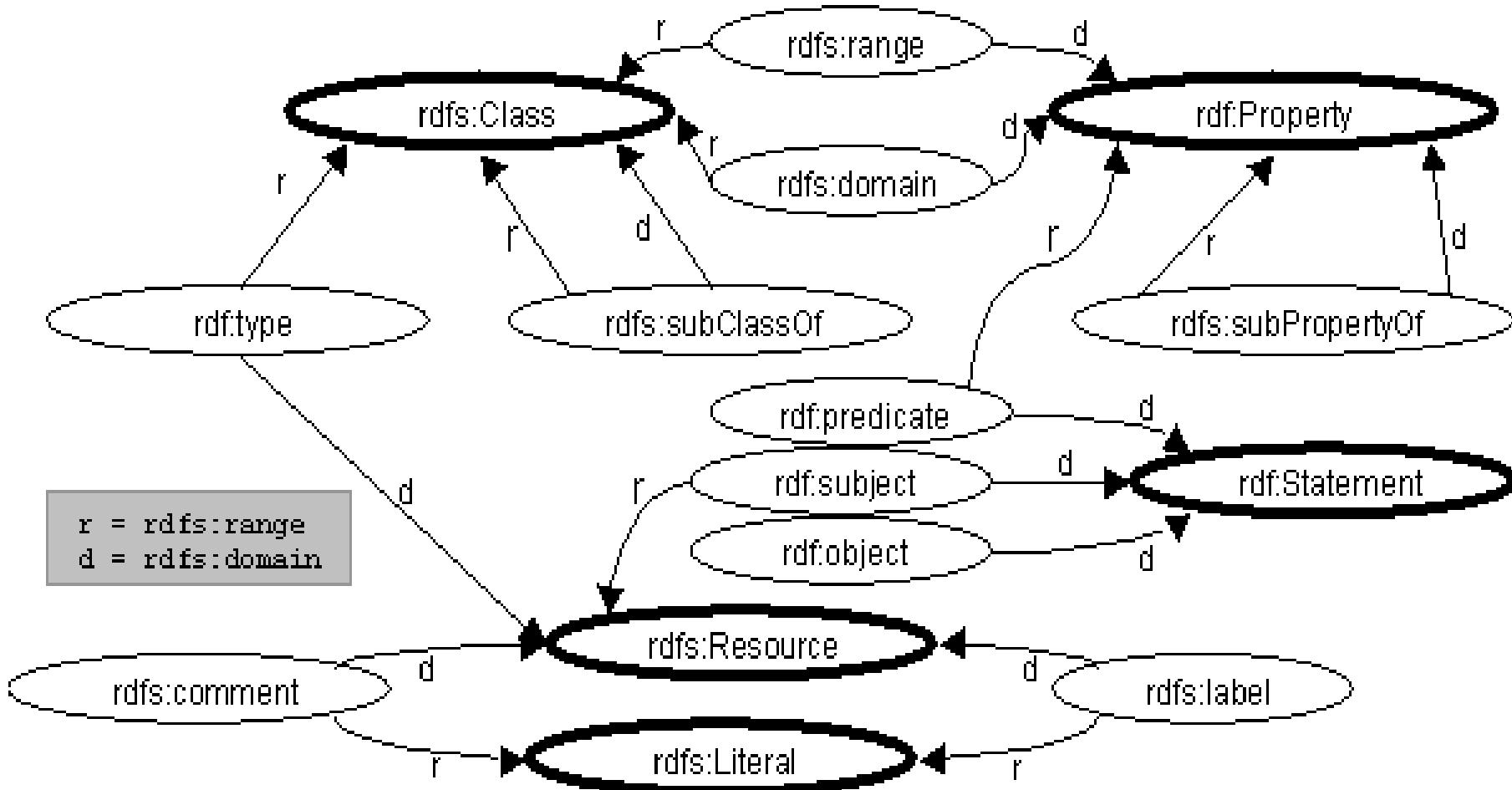
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
@prefix zoo: <http://example.org/zoo/> .
ex:dog1 rdf:type ex:animal .
ex:cat1 rdf:type ex:cat .
ex:cat rdfs:subClassOf ex:animal .
zoo:host rdfs:range ex:animal .
ex:zool zoo:host ex:cat2 .
```

# RDFS пример

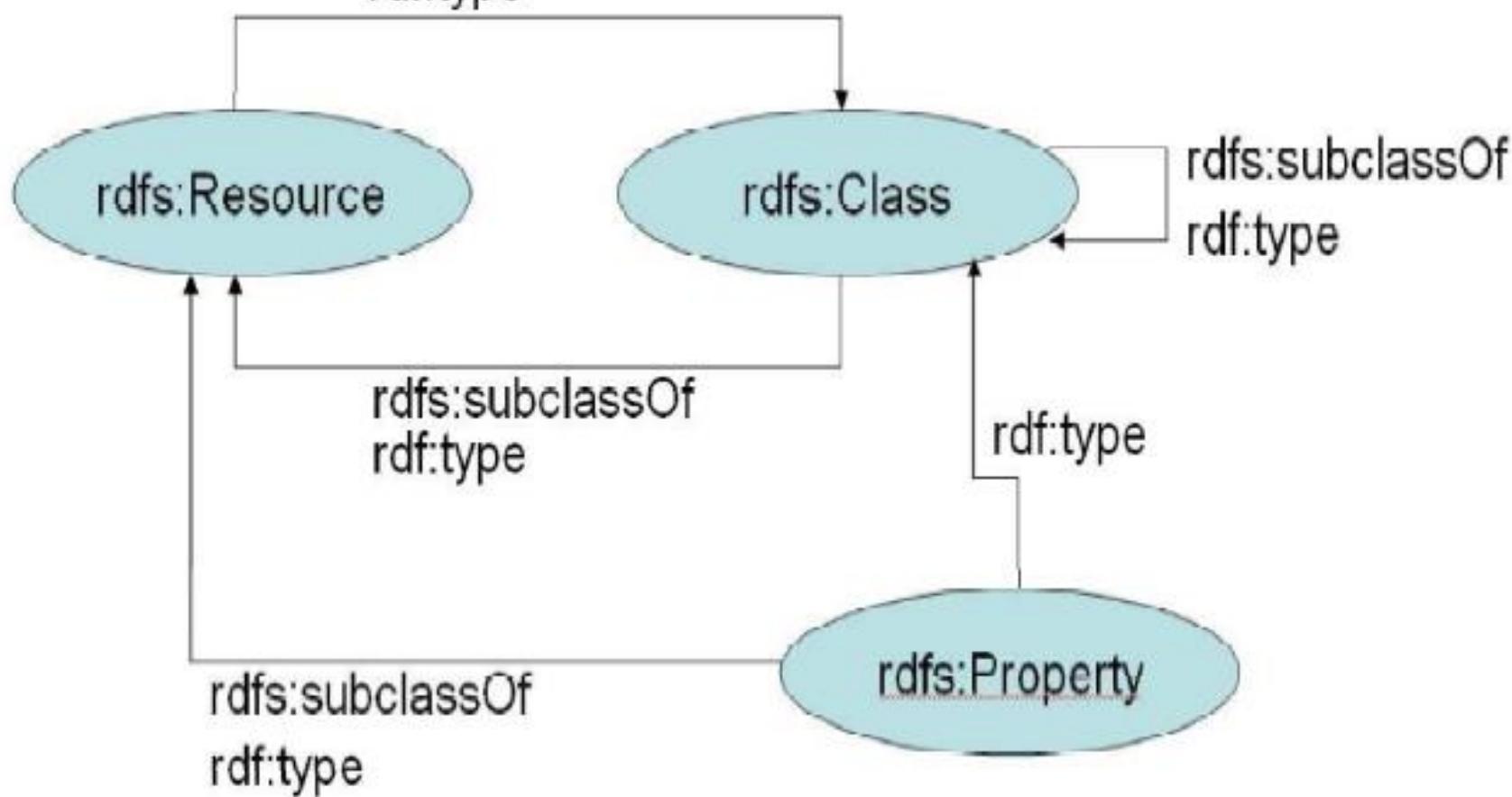


# Constraints in the RDF Schema

Nodes with **bold** outlines are instances of `rdfs:Class`.



# RDFS Vocabulary



# RDFS свойства на метаданни (Metadata Properties)

- Метаданните са “данни за данни”
- Към ресурс можем да прикрепим различни метаданни, чрез:
  - **rdfs:comment** → описание на ресурса в четим за хора вид, напр.

`<ex:Person>, rdfs:comment, "A person is any human being"`

- **rdfs:label** → име на ресурса в четим за хора вид, напр.

`<ex:Person>, rdfs:label, "Human being"`

- **rdfs:seeAlso** → посочва допълнителна информация за ресурса, напр. `<ex:Person>, rdfs:seeAlso, <http://xmlins.com/wordnet/1.6/Human>`
- **rdfs:isDefinedBy** → специален вид на **rdfs:seeAlso**, напр.  
`<ex:Person>, rdfs:isDefinedBy, <http://xmlins.com/wordnet/1.6/Human>`

# RDF литерали

- обикновени литерал
  - напр. "**any string**"
  - с optionalен маркер за език, напр. "**Hello, how are you?"@en-GB**
- типови литерали
  - напр. "**hello**"^^xsd:string, "1"^^xsd:integer
  - препоръчани типове данни:
    - типове данни на XML Schema
- могат да се явяват само като обект в тройка

# Литерали в RDFS

- Всеки литерал е rdfs:Literal
- Напр. в: <#john, #hasName, “John”>
- Значи ли това обаче, че:  
<“John”, rdf:type, rdfs:Literal>
  - **НЕ!** Литерал не може да бъде субект
- Обаче:
  - <#john, #hasName, \_:X>
  - <\_:X, rdf:type, rdfs:Literal>

# rdfs:Datatype ([https://www.w3.org/TR/rdf-schema/#ch\\_datatype](https://www.w3.org/TR/rdf-schema/#ch_datatype))

- rdfs:Datatype е клас от типове данни.
- Всички екземпляри на rdfs:Datatype отговарят на RDF модела на типове данни, описан в RDF Concepts specification [RDF11-CONCEPTS].
- rdfs:Datatype задава типове данни, съвместими с XML Schema.
- rdfs:Datatype е както екземпляр, така и под-клас на rdfs:Class.
- Всеки екземпляр на rdfs:Datatype е под-клас на rdfs:Literal.

## **rdfs:Container ([https://www.w3.org/TR/rdf-schema/#ch\\_datatype](https://www.w3.org/TR/rdf-schema/#ch_datatype))**

Класът **rdfs:Container** е супер-клас на RDF контейнер-класовете:

- [rdf:Bag](#),
- [rdf:Seq](#),
- [rdf:Alt](#)

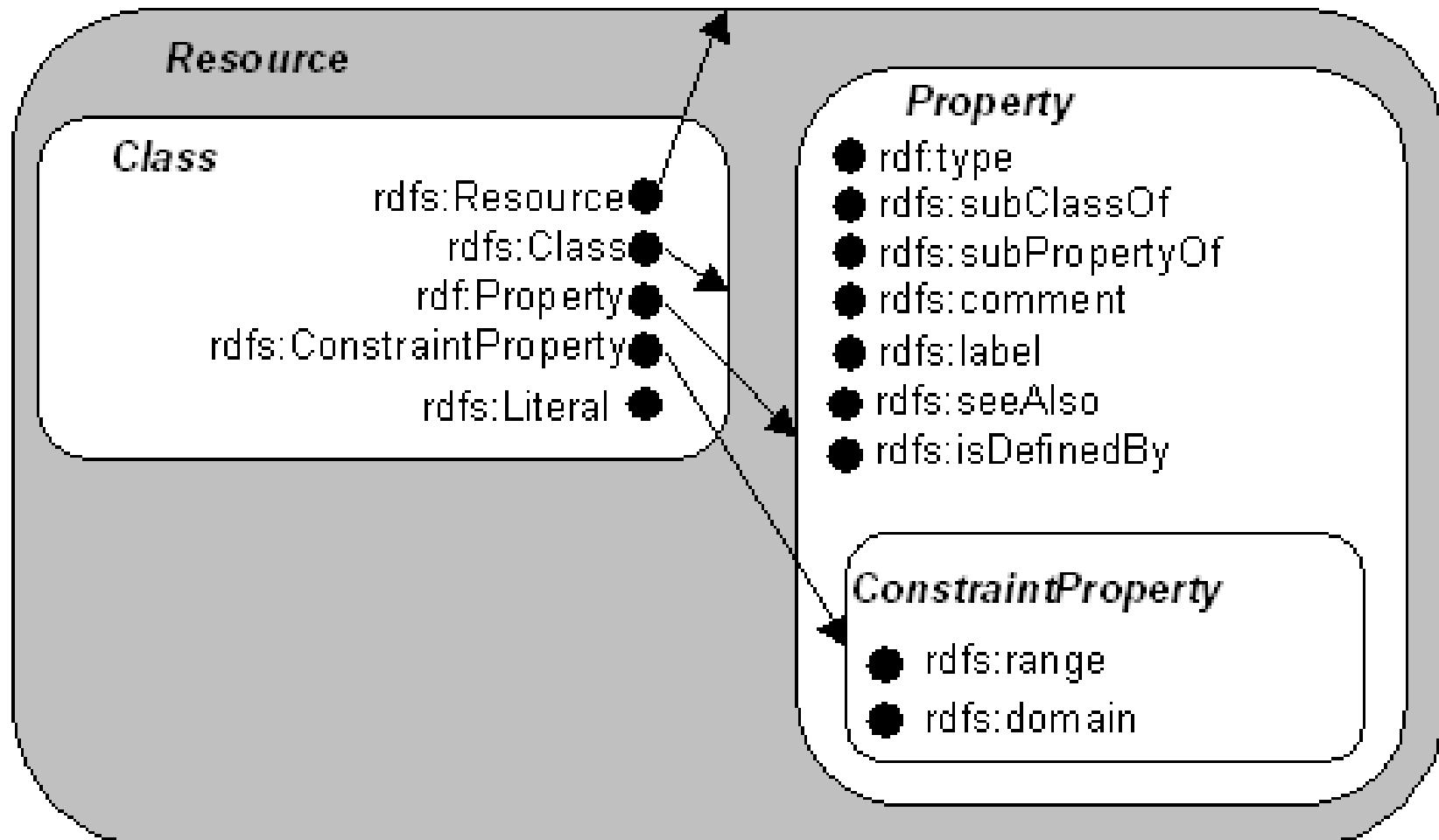
# rdfs:ContainerMembershipProperty ([https://www.w3.org/TR/rdf-schema/#ch\\_datatype](https://www.w3.org/TR/rdf-schema/#ch_datatype))

Ако е даден контейнерът С, с тройки във формата:

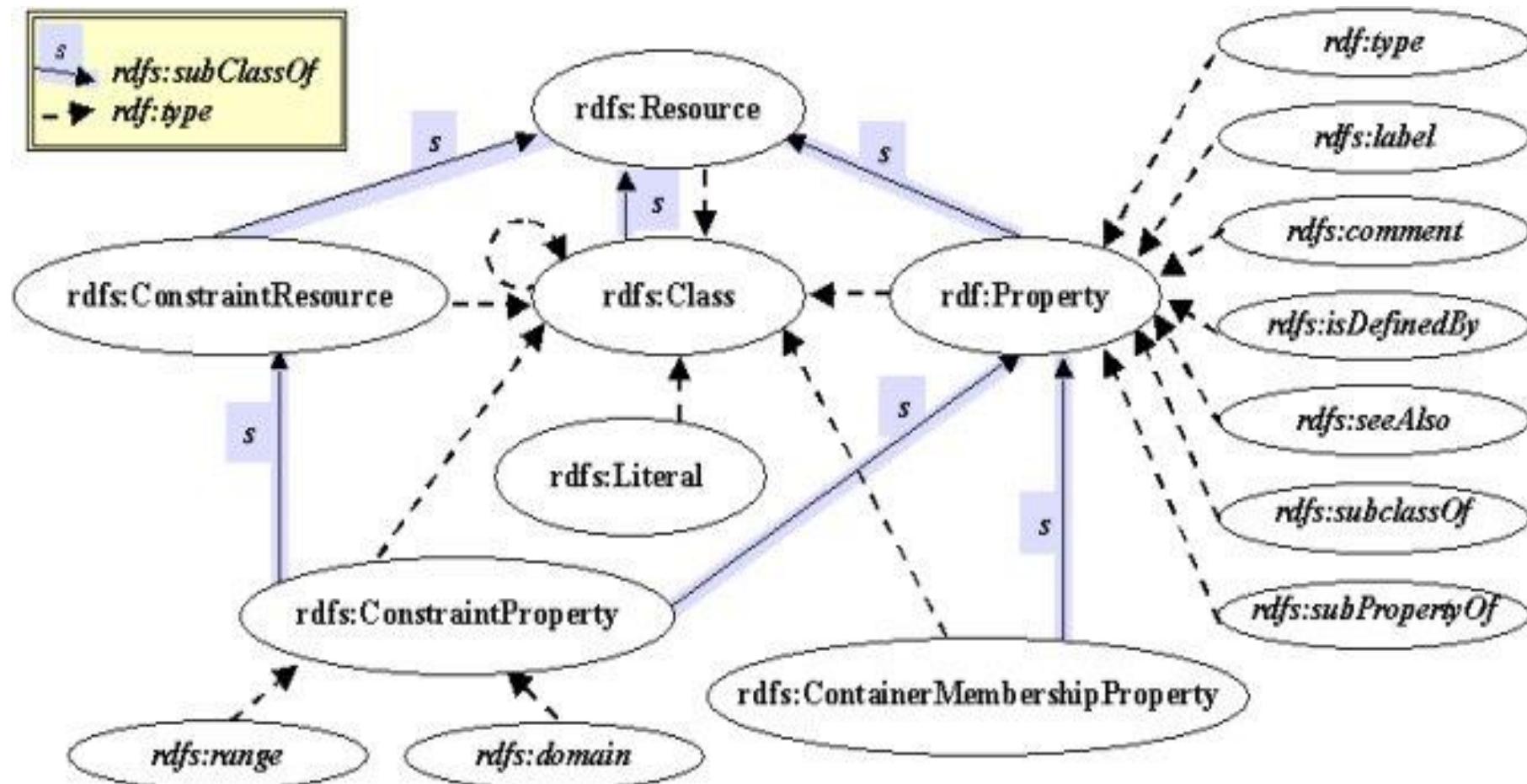
C rdf:\_nnn O (\_nnn = \_1, \_2, ...):

- Класът rdfs:ContainerMembershipProperty има за екземпляри свойствата rdf:\_1, rdf:\_2, rdf:\_3 ..., които указват, че ресурс O е член на контейнера C.
- rdfs:ContainerMembershipProperty е под клас на [rdf:Property](#).
- Всеки екземпляр на rdfs:ContainerMembershipProperty е [rdfs:subPropertyOf](#) на свойството [rdfs:member](#).

# Classes and Resources as Sets and Elements



# Class Hierarchy for the RDF Schema



# RDFS семантика

- RDF(S) речника има вградено значение
- RDF(S) сематиката:
  - прави значението експлицитно
  - дефинира какво следва от RDF графа,  
т.е. как се интерпретира графа
- За домашна работа: **RDF Semantics - a W3C Recommendation 10 February 2004**, <http://www.w3.org/TR/rdf-mt/>

# Модели на метаданни

Dublin Core

Qualified Dublin Core

Разработка на речници

FOAF речник

RSS

SPARQL

Примери

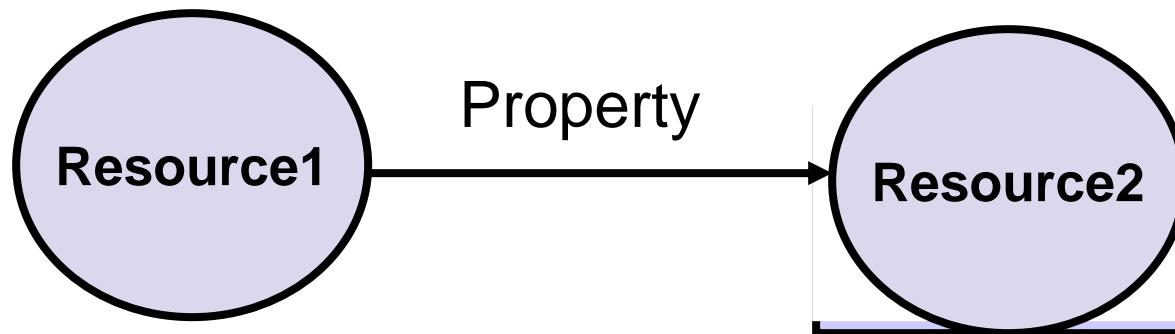


XML

RDFS

# RDF модел на данни

- Проектиран да наложи структурно ограничение върху синтаксиса за консистентно кодиране, обмен и обработка на метаданни
- Позволява общностите в дадена предметна област да определят свои собствени семантики
- Осигурява структурна оперативна съвместимост



# Модел на метаданни Dublin Core (DC)

- DC (от 1995г. насам) е приложение на RDF модела на данни
- RDF е достатъчно богат, за да поддържа целите на модела на Dublin Core
- Задава разширяема имплементационна рамка за приложения
- Развива се от Dublin Core Metadata Initiative - <http://www.dublincore.org/>

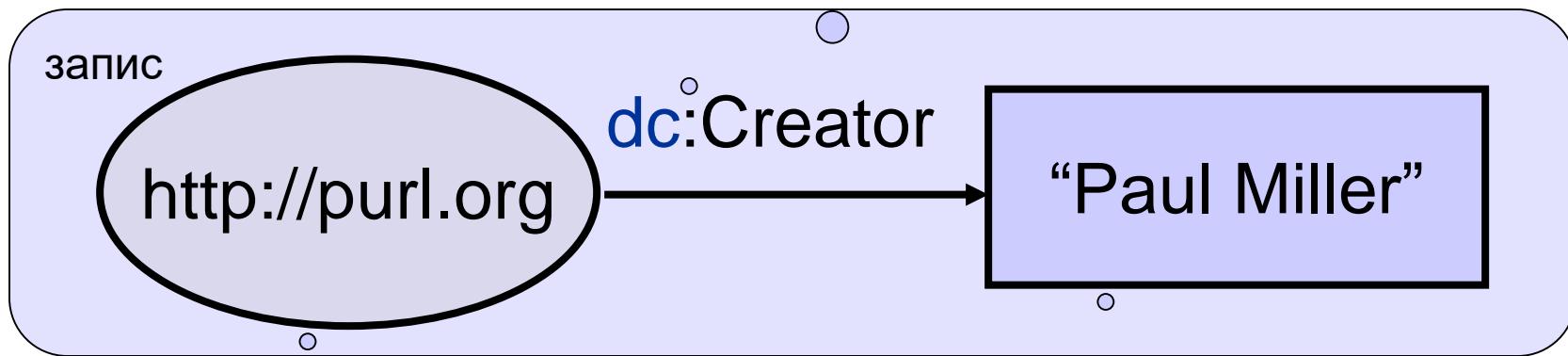
# Пространства от имена

- **DC** пространството дефинира елементите на ядрото (1998)
  - <http://purl.org/dc/elements/1.0/>
- **DCQ** пространството дефинира Dublin Core квалифициатори (<http://dublincore.org/documents/2000/07/11/dcterms-qualifiers/>) и механизми за разширяемост (2000)
  - <http://purl.org/dc/qualifiers/1.0/> - заместени от:
- **DCT** пространството дефинира Dublin Core термини (<http://dublincore.org/documents/dcmi-terms/>) и механизми за разширяемост (2012)
  - <http://purl.org/dc/terms/1.0/>

# Терминология в Dublin Core

- *Resource* – ресурс е всичко, което има идентичност, напр. документ, изображение, както и абстракции, които нямат представяне в Интернет
- *Property* – свойство е специфичен аспект, х-ка, атрибут или релация, описваща ресурса
- *Record* – запис е набор от структурирани метаданни за ресурса, обхващащи едно или повече свойства и техните стойности (*values*)

# Dublin Core модел на данни



ресурс

стойност

# Разлика м/ "Simple" (неквалифициран) и "Qualified" Dublin Core

- "**Simple Dublin Core**" не използва квалифicatorи, а само основните 15 елементи на DC Metadata, зададени като прости двойки атрибут - стойност, без каквото и да било схеми за кодиране, изброени списъци или друга инфо за обработката на даден ресурс
- "**Qualified Dublin Core**" използва допълнителни квалифicatorи за рафиниране на смисъла на ресурса, което увеличава спецификата и точността на метаданните. Напр. "date" е пример за DC елемент, за който има възможност да бъде допълнително уточнен като особен вид дата (като дата на последна промяна, дата на публикуване и др.)
- За съжаление, често квалифicatorите въвеждат допълнителна сложност, която може да направи метаданните по-малко оперативно съвместими. Решение: *контролирани речници*

# Контролирани речници

- Осигуряват начин да се организират знания за последващо извлечане
- Контролираните схеми речник използват само предварително определени термини, които са били предварително избрани/одобрени от дизайнера на речника
- Използват дефинирани и прилагани процедури за актуализирането им
- Използват се в схеми за индексиране, предметни рубрики (subject headings), тезауруси (thesauri), класификации (taxonomies) и други системи за организация на знанието

# Simple Dublin Core

(<http://dublincore.org/documents/2003/04/02/dc-xml-guidelines/>)

- A *simple DC record* is made up of one or more *properties* and their associated *values*.
- Each *property* is an attribute of the *resource* being described.
- Each *property* must be one of the 15 DCMES (*Dublin Core Metadata Element Set, Ver. 1.1: Reference Description*) elements.
- *Properties* may be repeated.
- Each *value* is a literal string.
- Each literal string *value* may have an associated language (e.g. en-GB).

# Simple Dublin Core

(<http://dublincore.org/documents/2003/04/02/dc-xml-guidelines/>)

```
<?xml version="1.0"?>
<metadata xmlns="http://example.org/myapp/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://example.org/myapp/
 http://example.org/myapp/schema.xsd"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
 <dc:title> UKOLN </dc:title>
 <dc:description> UKOLN is a national focus of expertise in
digital information management. </dc:description>
 <dc:publisher> UKOLN, University of Bath </dc:publisher>
 <dc:identifier> http://www.ukoln.ac.uk/ </dc:identifier>
</metadata>
```

# Qualified Dublin Core

(<http://dublincore.org/documents/2003/04/02/dc-xml-guidelines/>)

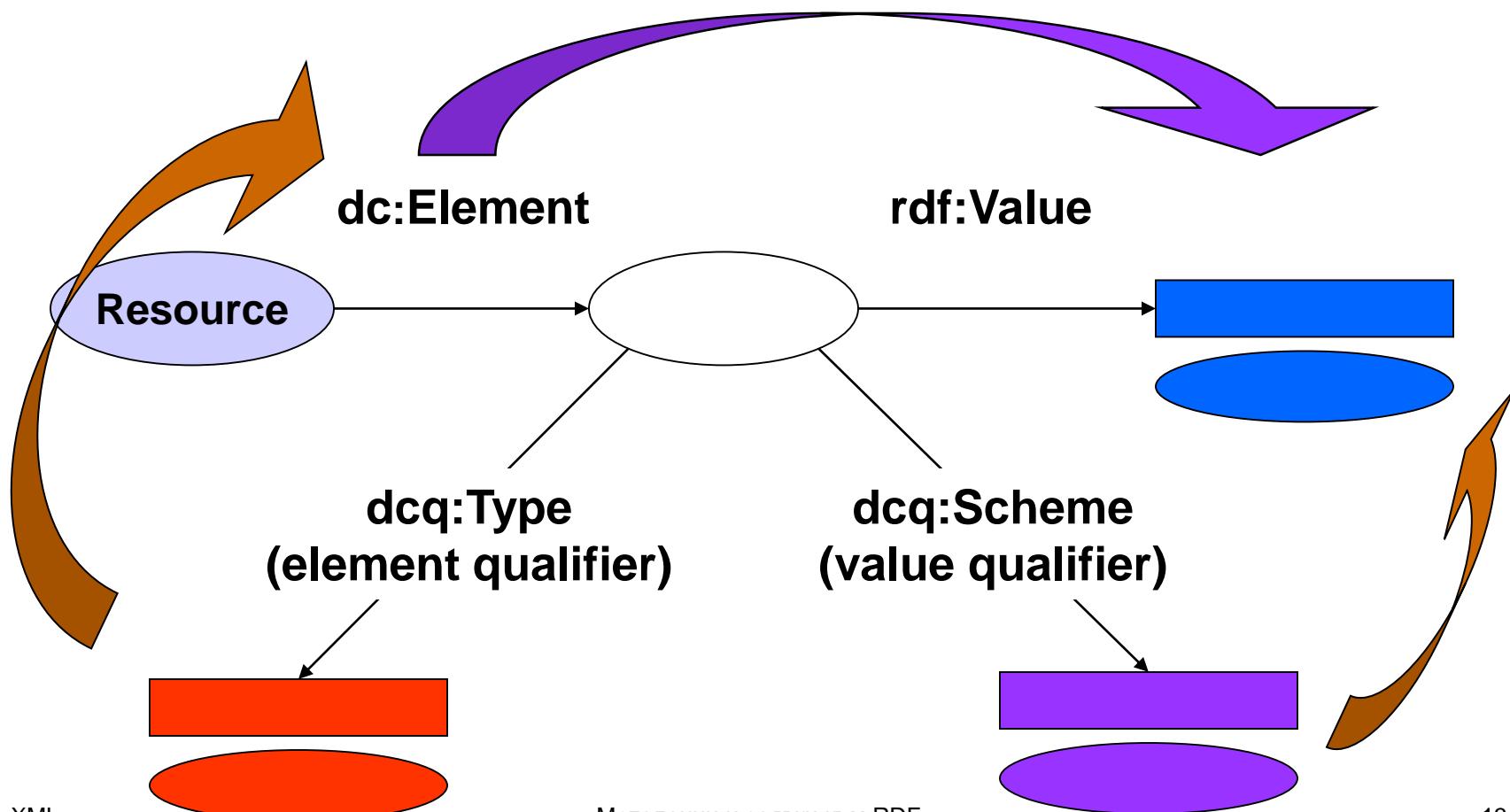
- A *qualified DC record* is made up of one or more *properties* and their associated *values*.
- Each *property* is an attribute of the *resource* being described.
- Each *property* must be either:
  - one of the 15 DC elements,
  - one of the other elements recommended by the DCMI (e.g. audience) [\[DCTERMS\]](#),
  - one of the *element refinements* listed in the DCMI Metadata Terms recommendation [\[DCTERMS\]](#).
- *Properties* may be repeated.
- Each *value* is a literal string.
- Each *value* may have an associated *encoding scheme*.
- Each *encoding scheme* has a *name*.
- Each literal string *value* may have an associated language (e.g. en-GB).

# Qualified Dublin Core

(<http://dublincore.org/documents/2003/04/02/dc-xml-guidelines/>)

```
<?xml version="1.0"?>
<metadata xmlns="http://example.org/myapp/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://example.org/myapp/
 http://example.org/myapp/schema.xsd"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:dcterms="http://purl.org/dc/terms/">
 <dc:title> UKOLN </dc:title>
 <dcterms:alternative> UK Office for ... </dcterms:alternative>
 <dc:subject> national centre, network inf. support... </dc:subject>
 <dc:subject xsi:type="dcterms:DDC"> 062 </dc:subject>
 <dc:subject xsi:type="dcterms:UDC"> 061(410) </dc:subject>
 <dc:description> UKOLN is ... </dc:description>
 <dc:description xml:lang="fr"> UKOLN est un centre </dc:description>
 </metadata>
```

# Dublin Core модел като RDF



# Най-популярни DC елементи за метаданни

- title (the name given the resource)
- creator (the person or organization responsible for the content)
- subject (the topic covered)
- description (a textual outline of the content)
- publisher (those responsible for making the resource available)
- contributor (those who added to the content)
- date (when the resource was made available)
- type (a category for the content)
- format (how the resource is presented)
- identifier (numerical identifier for the content such as a URL)
- source (where the content originally derived from)
- language (in what language the content is written)
- relation (how the content relates to other resources)
- coverage (where the resource is physically located)
- rights (a link to a copyright notice)

## *All the 15 elements of Simple Dublin Core*

| <b><u>Instantiation:</u></b>         |             |            |          |
|--------------------------------------|-------------|------------|----------|
| Date                                 | Format      | Identifier | Language |
| <b><u>Content:</u></b>               |             |            |          |
| Title                                | Description | Coverage   |          |
| Relation                             | Source      | Subject    | Type     |
| <b><u>Intellectual Property:</u></b> |             |            |          |
| Contributor                          | Creator     | Publisher  | Rights   |

*Any element may be used as many or as few times as needed; there is no order to their use. You must refer to the schema in the namespace at the top of the file. (The schema details what elements may be used and how.)*

### **Example schema reference:**

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dc="http://purl.org/dc/elements/1.1/"
```

# Квалифицирани DC елементи

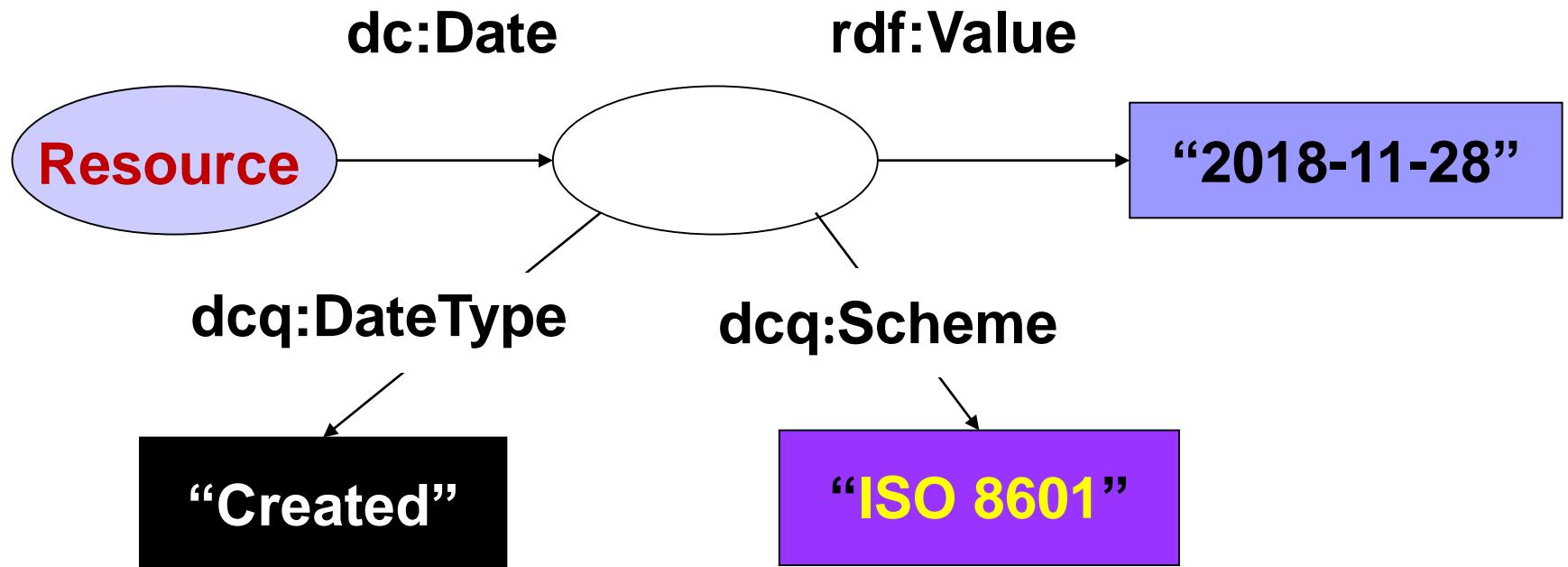
abstract, accessRights, accrualMethod, accrualPeriodicity, accrualPolicy, alternative, audience, available, bibliographicCitation, conformsTo, **contributor**, **coverage**, created, **creator**, **date**, dateAccepted, dateCopyrighted, dateSubmitted, **description**, educationLevel, extent, **format**, hasFormat, hasPart, hasVersion, **identifier**, instructionalMethod, isFormatOf, isPartOf, isReferencedBy, isReplacedBy, isRequiredBy, issued, isVersionOf, **language**, license, mediator, medium, modified, provenance, **publisher**, references, **relation**, replaces, requires, **rights**, rightsHolder, **source**, spatial, **subject**, tableOfContents, temporal, **title**, **type**, valid

# DC квалифicatorи

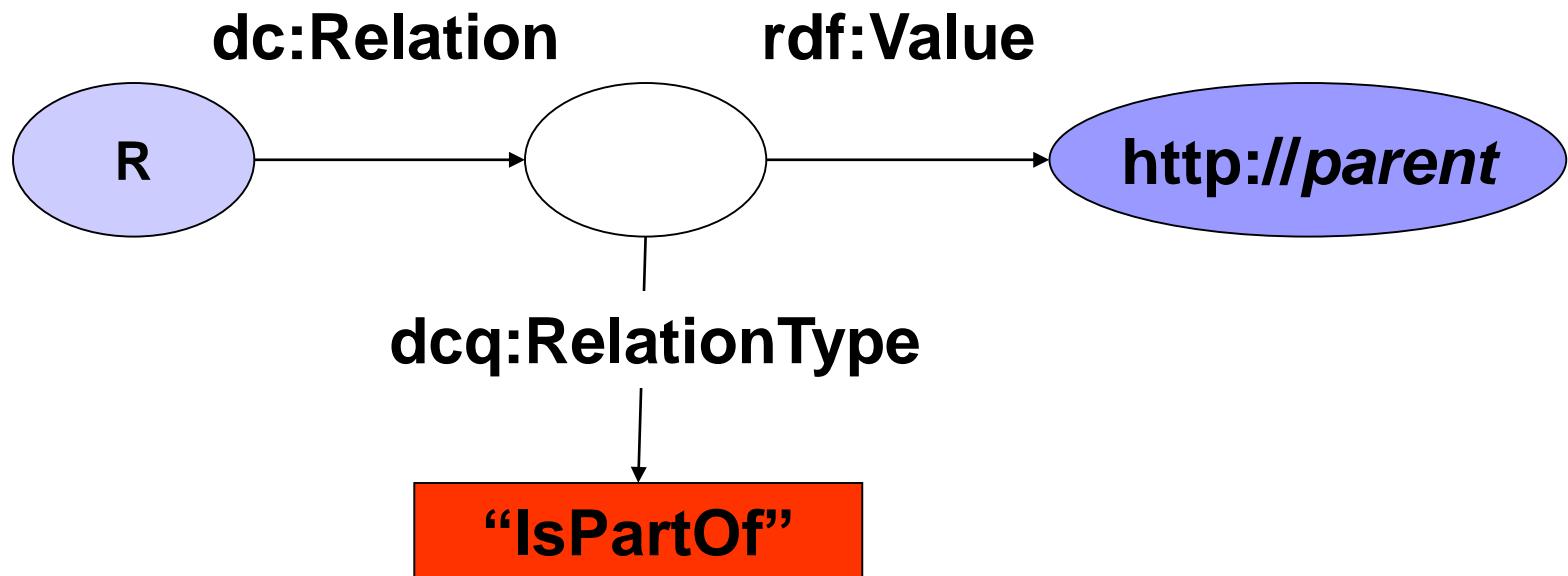
Квалифicatorи на:

- Елементи и термове
- Стойност
- Тип
- Език

# Пример за метаданни ‘Date’



# Пример за метаданни ‘Relation’



# Квалификатор на елемент и термини

- Базови механизми в модела на данните, които поддържат квалификацията на елемент, свързващ ресурса и стойността
  - Например терминът "Illustrator" може да бъде използван, за да се квалифицира елемента "Creator", който свързва някакви ресурси и дадена стойност
- Термини (Terms)
  - Ресурсът, идентифициращ "Illustrator"

# Квалификатори на стойност и термини

- Идентифицират кодирането, парсването и правилата за обработка на стойността:
  - LNF: “Lastname,[sp]Firstname”
  - ISO8601: 1998-10-01
  - DDC: 325.251
  - AAT: legal files
- Термини
  - Ресурси, дефиниращи LNF, ISO8601, DDC, AAT (Art & Architecture Thesaurus) и т.н.

# Dewey Decimal Classification (DDC)

- The Dewey Decimal Classification organizes library materials by discipline or field of study.
- Main divisions include philosophy, social sciences, science, technology, and history.
- The scheme comprises ten classes, each divided into ten divisions, each having ten sections.

500 Natural sciences and mathematics

    510 Mathematics

        516 Geometry

            516.3 Analytic geometries

                516.37 Metric differential geometries

                    516.375 Finsler geometry

# Квалификатори на език

- Дефинира език за стойността
  - lang=fr “chat” спрямо lang=en “chat”
- XML предоставя начин за третиране на езика (чрез xml:lang)
  - RDF и DC го използват

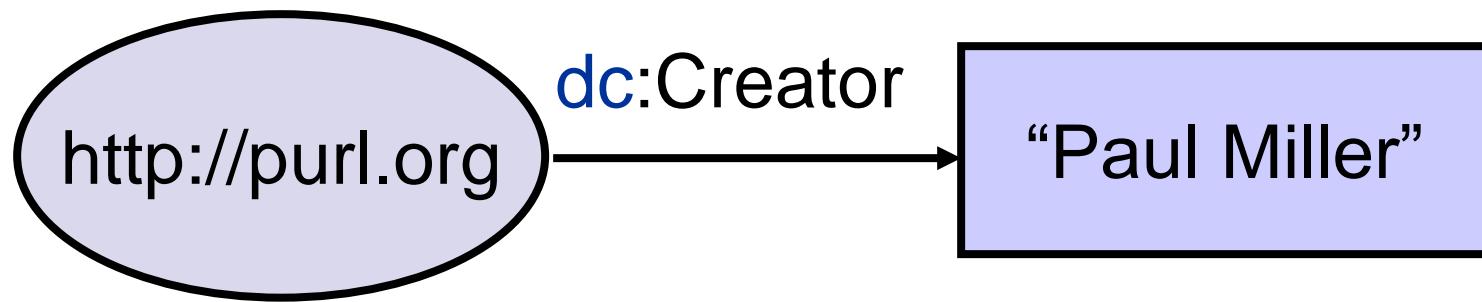
# Резултат

- Разделянето на тези конструкции е важно за разширяемостта им от други общини, описващи ресурси
- Dublin Core инициативата за метаданни не определя всички възможни термини
- Тя определя канонично множество от тях с механизми за разширение

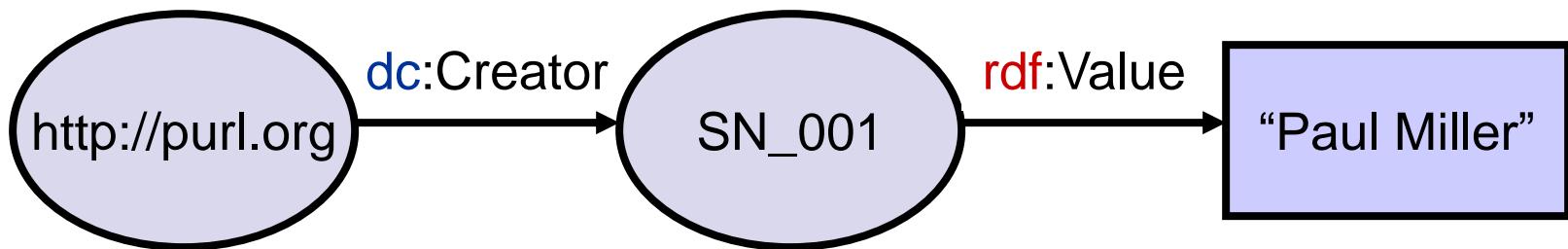
# DC в HTML (RFC 2731 Encoding Dublin Core Metadata in HTML, <http://www.ietf.org/rfc/rfc2731.txt>)

```
<html> <head> <title> Nutritional Allocation Increase </title>
<meta name = "DC.Creator" content = "Simpson, Homer">
<meta name = "DC.Title" content = "Nutritional Allocation Increase">
<meta name = "DC.Date.Created" content = "1999-03-08">
<meta name = "DC.Identifier" content =
"http://moes.bar.com/doh/homer.html">
<meta name = "DC.Format" content = "text/html; 1320 bytes">
<meta name = "DC.Language" content = "en-BUREAUCRATESE">
<meta name = "RC.MetadataAuthority" content = "Springfield Nuclear">
<meta name = "DC.Type" content = "Memorandum">
<link rel = "schema.DC" href = "http://purl.org/DC/elements/1.0/">
</head> <body> <p> From: Acting Shift Supervisor To: Plant Control Personnel
RE: Nutritional Allocation Increase Date: 1999-03-08 <p> Pursuant to directive
DOH:10.2001/405aec of article B-2022, subsection 48.2.4.4.1c regarding staff
morale and employee productivity standards, the current allocation of doughnut
acquisition funds shall be increased effective immediately. </body> </html>
```

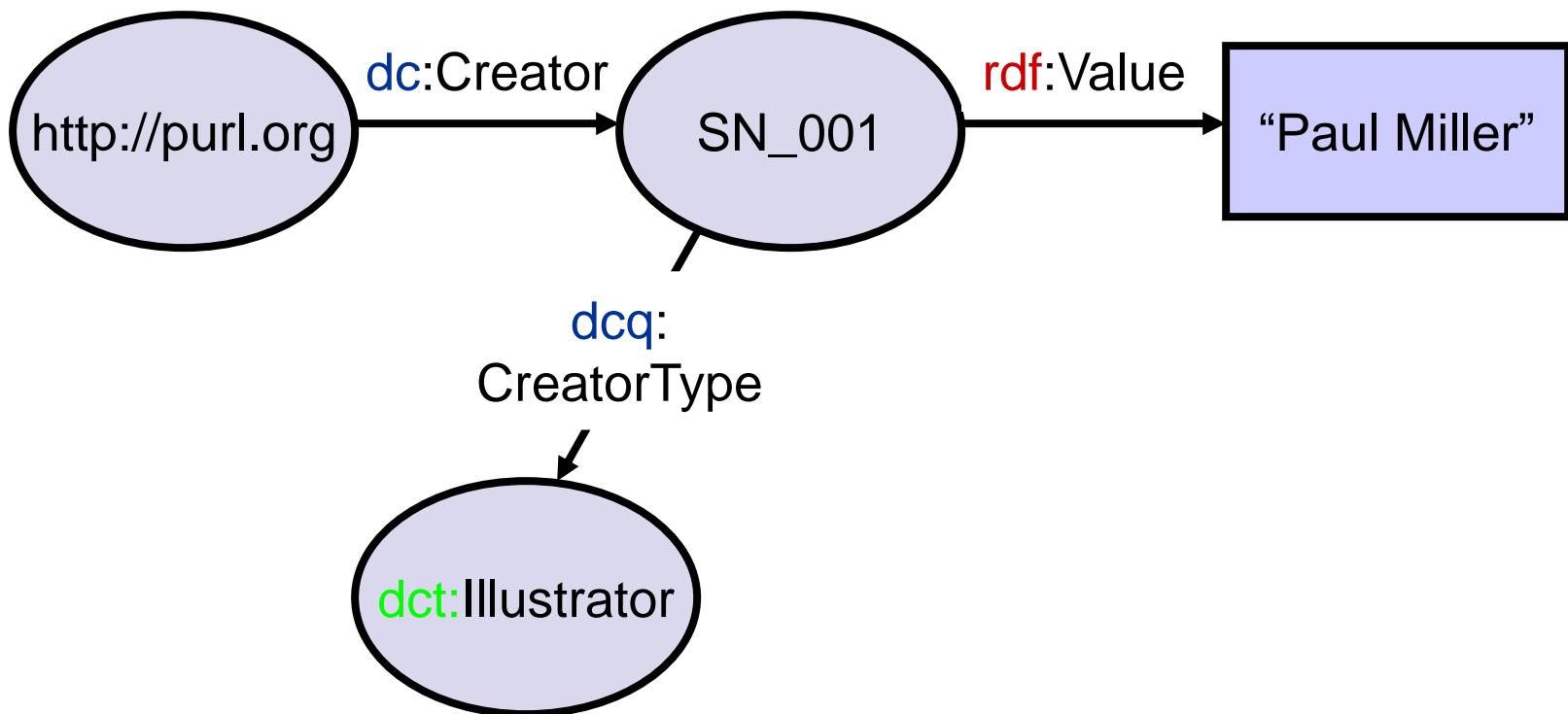
# Dublin Core модел на данни



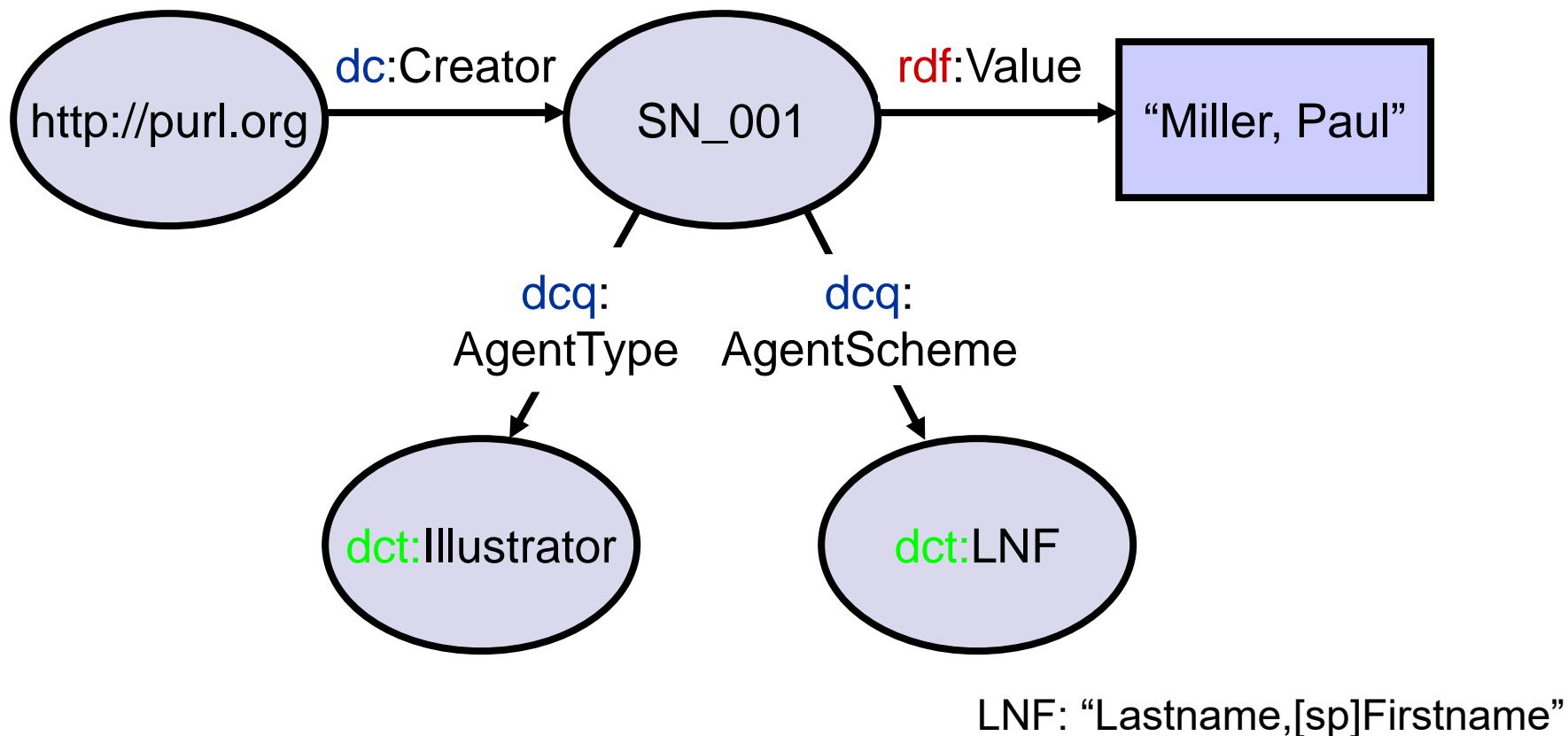
# Dublin Core модел на данни



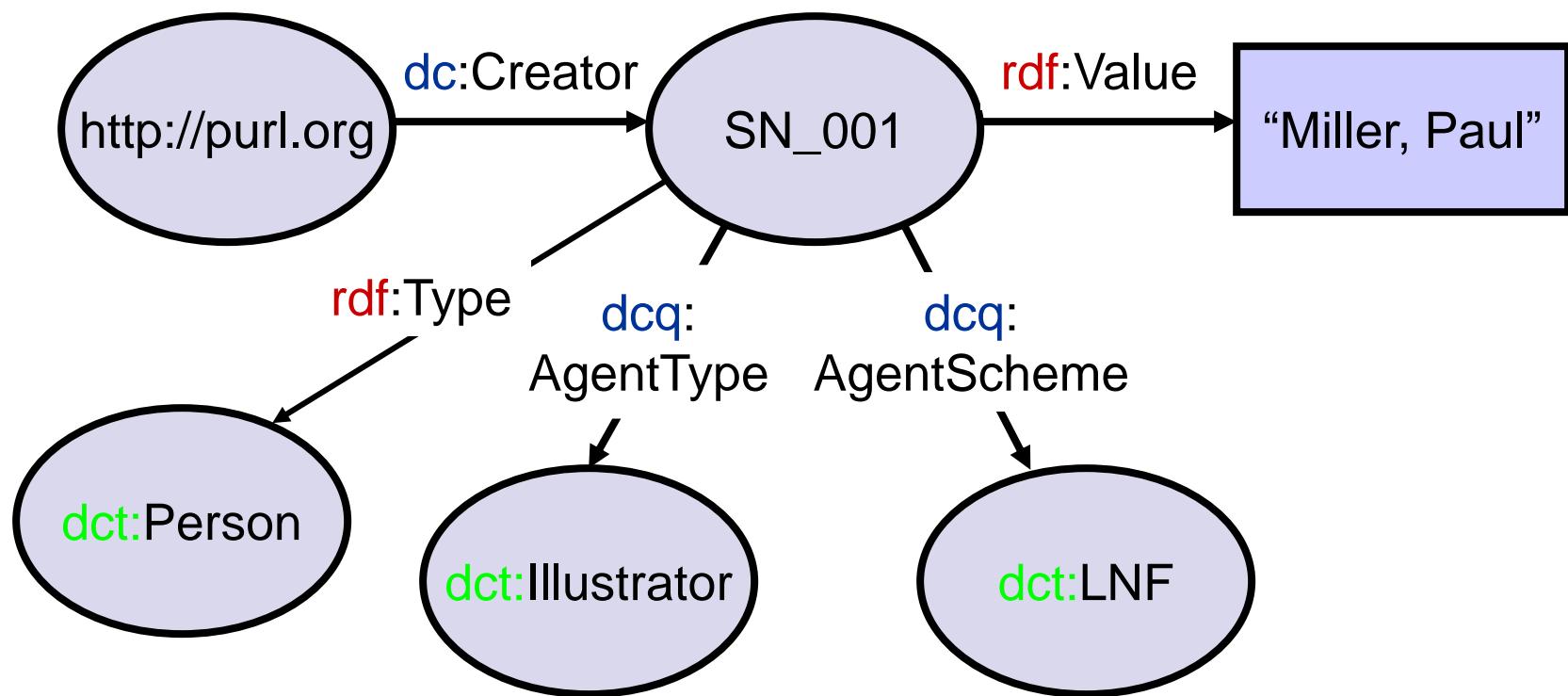
# Dublin Core модел на данни



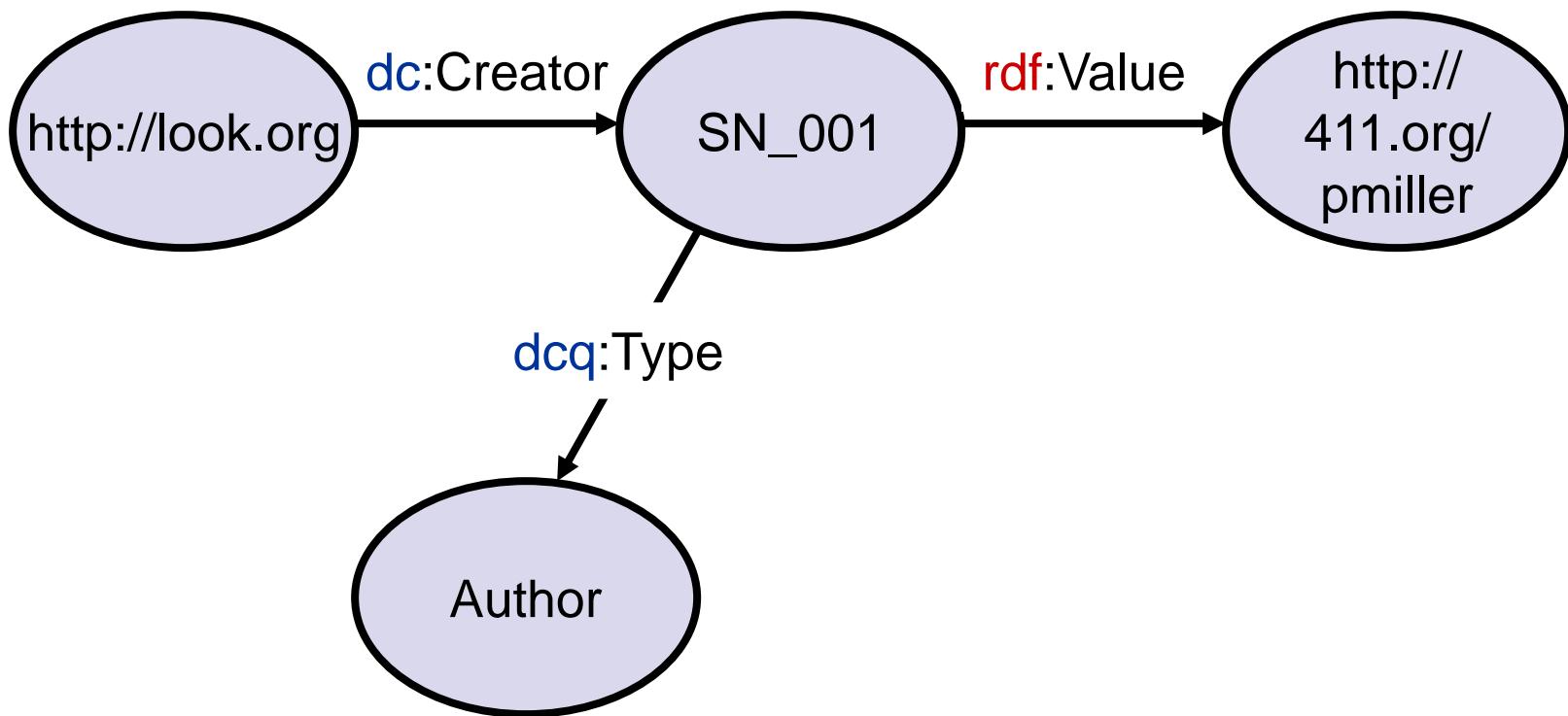
# Dublin Core модел на данни



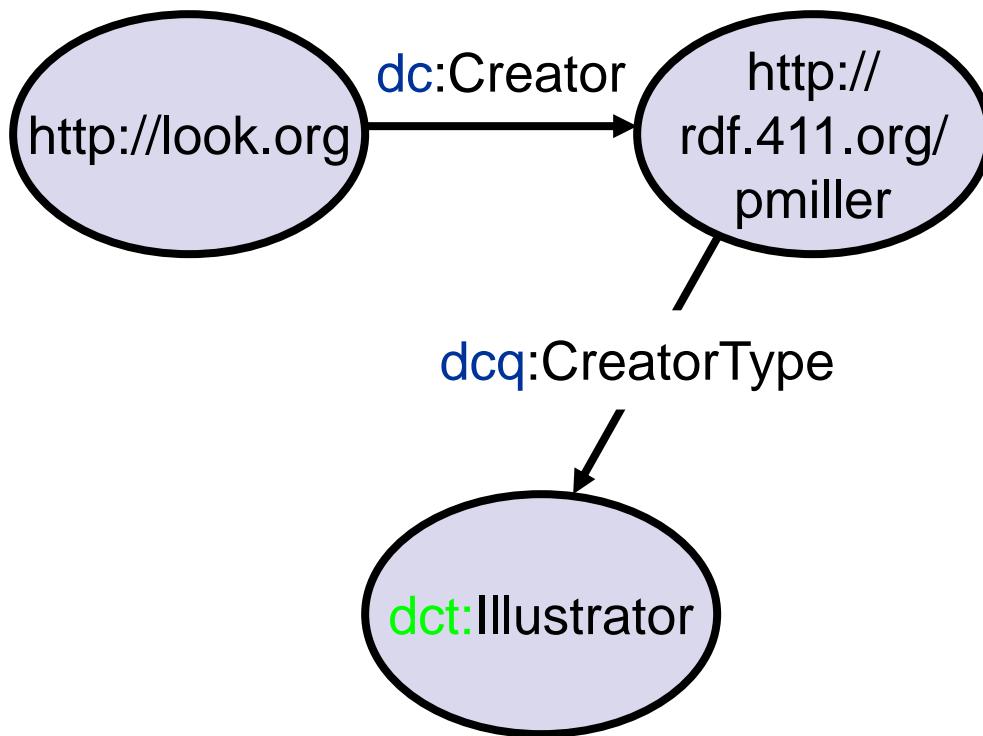
# Dublin Core модел на данни



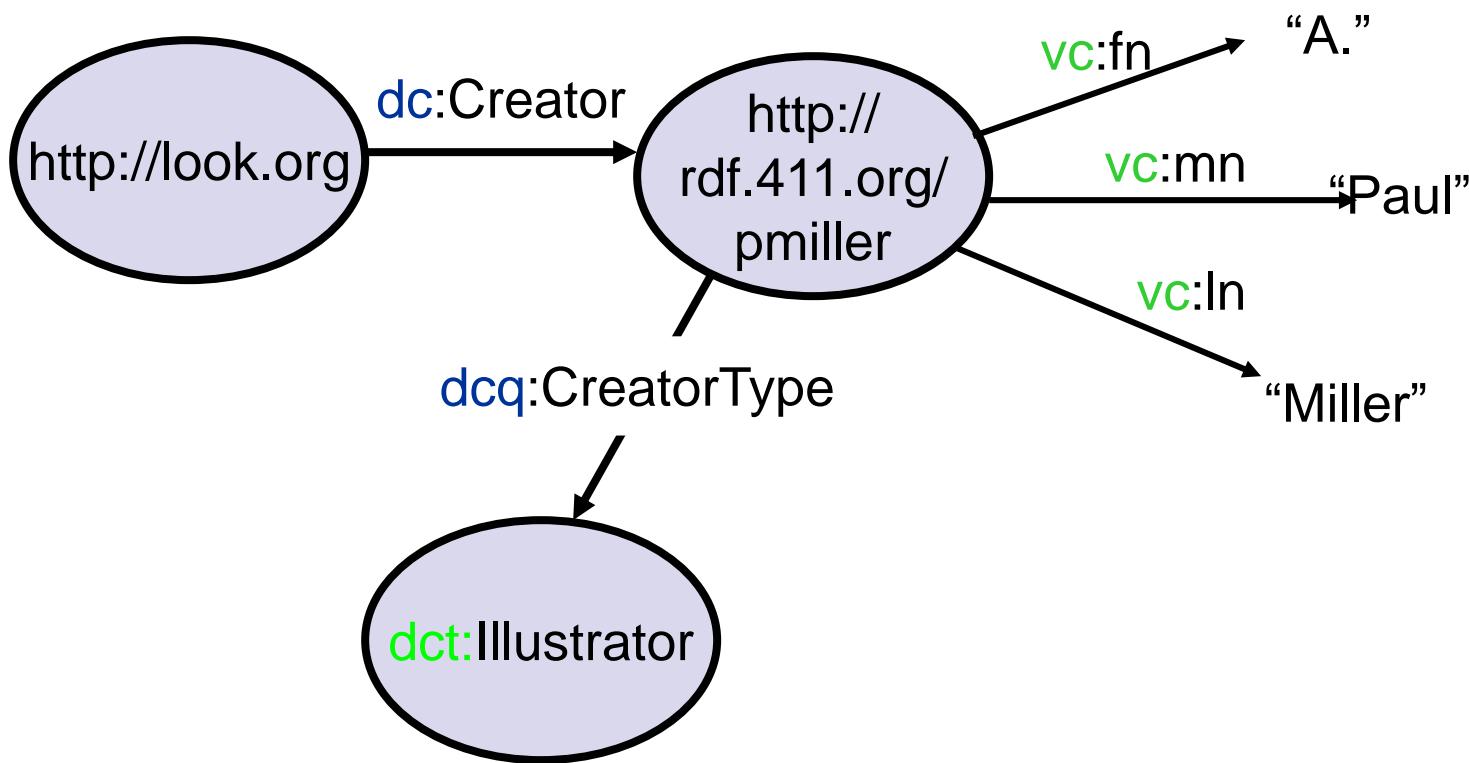
# Dublin Core модел на данни



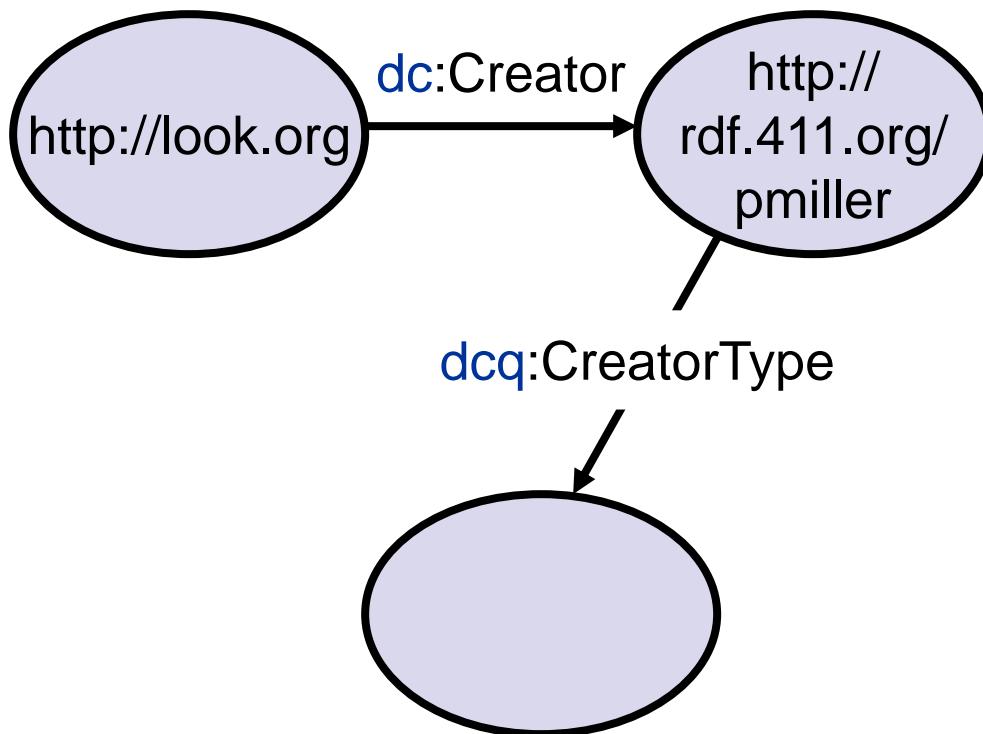
# Dublin Core модел на данни



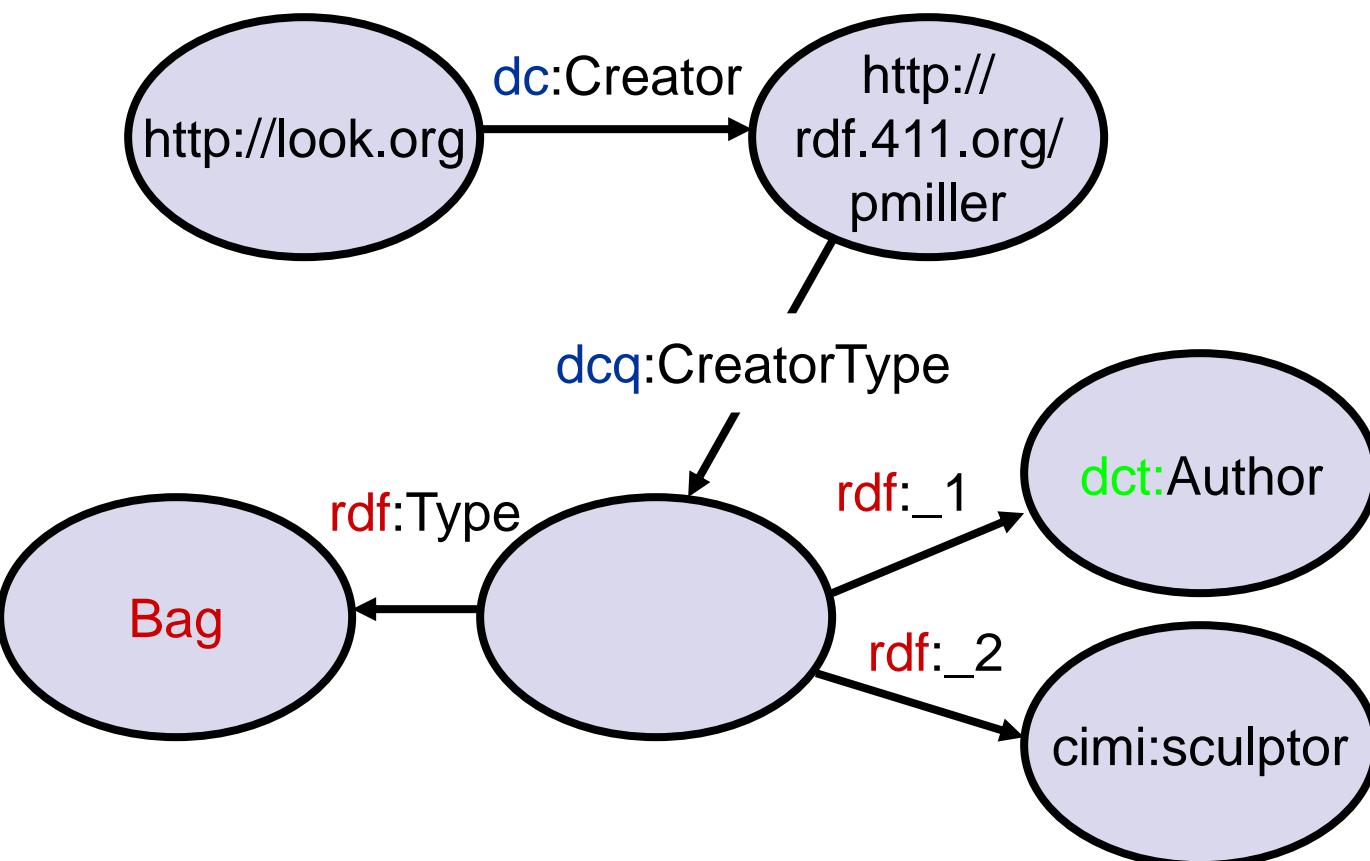
# Dublin Core модел на данни



# Dublin Core модел на данни



# Dublin Core модел на данни



# Dublin Core metadata editor - <http://www.ukoln.ac.uk/cgi-bin/dcdot.pl>

Dublin Core metadata editor generator – a service retrieving a Web page and automatically generating Dublin Core metadata:

- either HTML <meta> tags suitable for embedding in the <head>...</head> section of the page
- or RDF.

The generated metadata can be edited using the form provided. Optional, context sensitive, help is available while editing.

Stopped in 2013...

### За <http://sinoptik.bg/sofia-bulgaria-100727011?location> :

```
<?xml version="1.0"?> <!DOCTYPE rdf:RDF SYSTEM
"http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="http://sinoptik.bg/sofia-bulgaria-100727011/10-
days">
 <dc:title> 10-дневна прогноза за времето в София, България - Sinoptik.bg
 </dc:title>
 <dc:subject> 16:54; 07:51; mm; 07:49; Vesti.bg; 16:55; 07:52; 16:56; 07:54;
9:03; 07:50; Edna.bg; 9:02; 07:53; DOX; 16:53; 9:04; Sinoptik.bg; English; m/s;
Pariteni.bg; Sportni.bg; Adwise; 07:48 </dc:subject>
 <dc:description> 10-дневна подробна прогноза за времето в София.
 </dc:description>
 <dc:type> Text </dc:type> <dc:format> text/html; charset=UTF-8 </dc:format>
 <dc:format> 66233 bytes </dc:format>
</rdf:Description>
</rdf:RDF>
```

# Оптайте:

[https://nsteffel.github.io/dublin\\_core\\_generator/generator\\_nq.html](https://nsteffel.github.io/dublin_core_generator/generator_nq.html)

The screenshot shows a web browser displaying the [dublincoregenerator.com - a better dublin core generator](https://www.dublincoregenerator.com/generator_nq.html) website. The page has a blue header bar with links to Main Page, Simple Generator, Advanced Generator, xZINECOREx Generator, About, and Contribute. Below the header is a yellow "Directions" section containing a bulleted list of instructions for using the generator. At the bottom is an "Input" section with three fields: Title?, Creator?, and Subject?, each with a [+][-] button.

dublincoregenerator.com - a better dublin core generator

Main Page Simple Generator Advanced Generator xZINECOREx Generator About Contribute

**Directions**

- Fill in the fields below and click on "Generate Code!" to convert your input into fully formed Dublin Core metadata code. Additional options for the format of the output code are available below.
- If you need additional copies of a given field, click the plus sign to the upper-right of the tag's name to add an additional copy of it.
- Click the minus sign to delete any unneeded additional copies -- don't worry about removing tags you don't intend to use, the system will ignore any empty tags (and you can't delete the first row anyway).
- If you are unsure how a specific tag works, you can click the question mark next to the tag's name to see the tag's entry in Diane Hilmann's wonderful guide "Using Dublin Core -- The Elements."
- If you would like to use encoding schemes and the more advanced qualified elements of Dublin Core metadata, use the Advanced Generator located [here](#).

**Input**

Title?	[+][-]
Creator?	[+][-]
Subject?	[+][-]

DC Template (Based on DCMES)

Use a pipe (as a vertical bar) character " | " to separate multiple creators, contributors, subjects, formats, identifiers, etc.  
(Find it above the "\" on the right side of a keyboard.)

Submit options are: "Preview as HTML"; "Preview as XML"; "Preview as RDF/XML"; "Preview as RDF-Turtle"; and "Clear all".

### Title

Creator (Last, First M. or from a name authority) (Link to VIAF)

© 2013 Pearson Education, Inc.

### Description

Map 2001

Publisher

### Contributor

ANSWER

Date (yyyy-mm-dd, yyyy-mm, yyyy)

Type (Recommend to use [DCMI Type Vocabulary](#))

#### Identifier (URI, URL, DOI, ISBN, ISSN, ...)

## Source

Language (Select a value from the three letter language tags of ISO 639)

© 2010 Pearson Education, Inc., publishing as Pearson Addison Wesley. All rights reserved.

**Relation** (A URI of another resource)

**Coverage** (Spacial and temporal coverage of the content)

Rights (A statement or a link to a statement)

[View Details](#) | [Edit](#) | [Delete](#)

# Дискусия

- Dublin Core определя редица елементи на метаданни, но как да третираме стойностите за тези елементи?
  - Те могат да бъдат неограничени текстови стойности или...
  - Да се задават от предварително определени речници?
- Отговор: зависи от целта и начина на използване.

# Разработка и поддръжка на речници

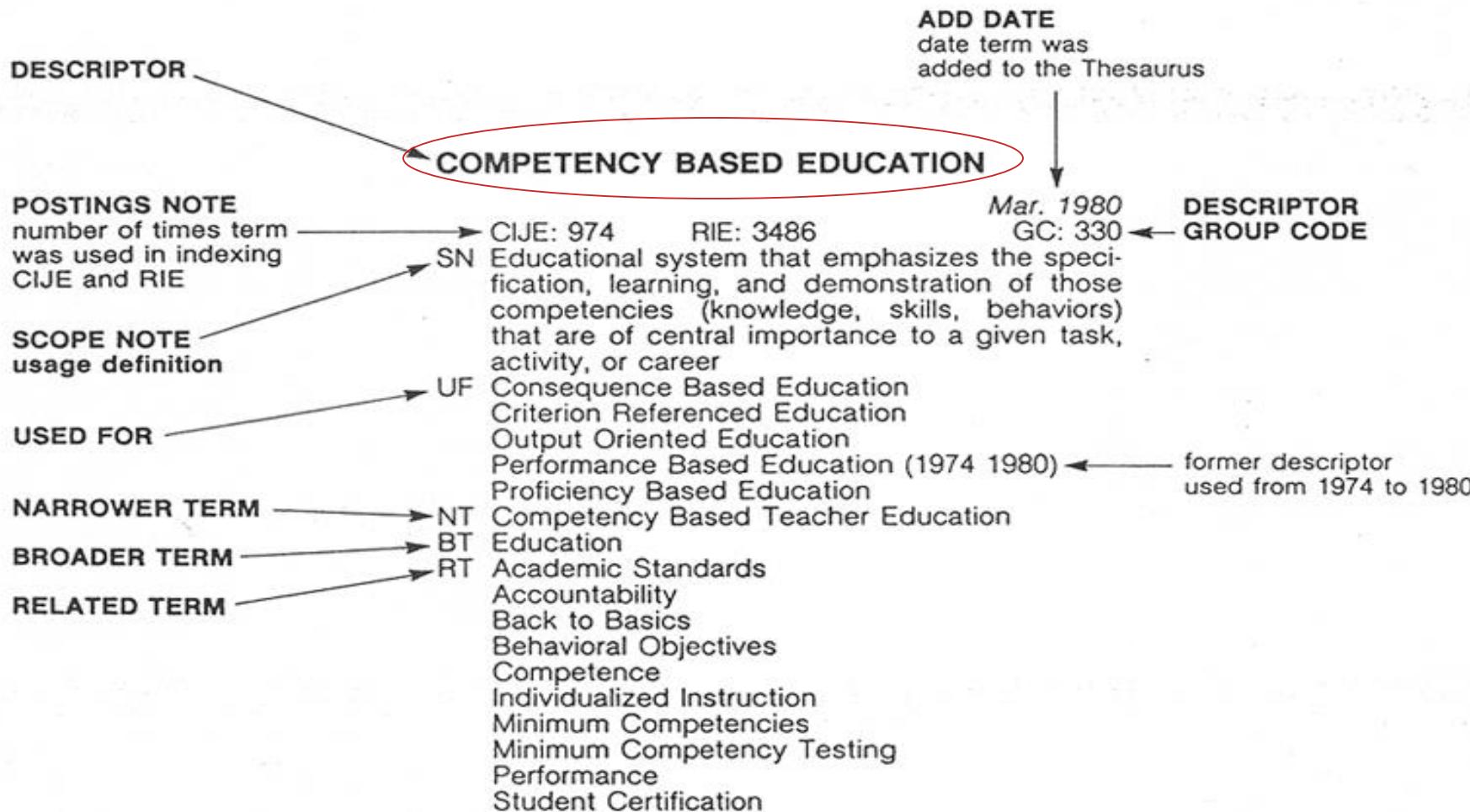
- **Vocabulary Problem:** Как ще се изграждат и поддържат списъците на предварително зададени стойности, които могат да се ползват за някои от елементите на метаданни?
- **Tagging Problem:** Как ще задаваме пълни и консистентни стойности за елементите за метаданни?
  - Полза от автоматичните класификатори?
  - Процедури за детектиране на грешки и възстановяване от тях?
- **ROI Problem:** Как да използваме в приложениета съдържание, метаданни и речници така, че да получим добавена стойност в бизнеса?

Термин	Определение
<b>Metadata елемент</b>	Поле за съхранение на информация за част от съдържанието, напр. Title, Creator, Subject, Date, ...
<b>Metadata стойност</b>	Съдържанието на даден елемент за описание на метаданни (текст или избор от предефиниран речник).
<b>Metadata схема</b>	Дефинирано множество от елементи за метаданни (напр. Dublin Core).
<b>Свободен лiteral</b>	Текстова стойност за метаданни без ограничения (евентуално със зададен формат)
<b>Речник (Vocabulary)</b>	Списък на предварително зададени стойности за елемент на метаданни.
<b>Контролиран речник</b>	Речник с дефинирани и прилагани процедури за актуализирането му.

Тип речник	Ниво на сложност	Описание	Тип на релация
<b>Term List</b>	1	Прост списък от термини, без вътрешна структура или отношения.	Няма
<b>Synonym Rings</b>	2	Списък на набор от термини, които се считат за равностойни. Широко използвани в софтуера за търсене.	Еквивалентност
<b>Authority Files</b>	3	Списък от имена на известни лица или артефакти - хора, организации, книги и др.	Референция
<b>Classification Schemes</b>	4	Йерархична подредба на понятия (концепции); таксономия.	Йерархия
<b>Thesauri</b>	5	Йерархична подредба на понятия плюс информация за допълнителни, не-йерархични отношения (напр. по-широко/потъсно понятие, синоними, еквивалентност...).	“Is-a” йерархия + слаби релации
<b>Ontologies</b>	6	Организиране на понятия и отношения, базирани на модел на реалността – напр. симптоми, болести и лечение в медицината.	Типови релации, базирани на модели

# SAMPLE THESAURUS ENTRY

## ALPHABETICAL DESCRIPTOR DISPLAY





# ERIC Processing and Reference Facility

Educational Resources Information Center

[contact us](#)   [site map](#)   [links](#)

[Home](#)

[Ready References](#)

[Submitting Documents](#)

[Reproduction Release Form](#)

[Products](#)

[Resources](#)

[Back to Thesaurus Search](#)

Term:	<b>Competency Based Education</b>
Record Type:	Main
Scope Note:	Educational system that emphasizes the specification, learning, and demonstration of those competencies (knowledge, skills, behaviors) that are of central importance to a given task, activity, or career
Category:	330
Broader Terms:	<a href="#">Education</a> ;
Narrower Terms:	<a href="#">Competency Based Teacher Education</a> ;
Related Terms:	<a href="#">Academic Standards</a> ; <a href="#">Accountability</a> ; <a href="#">Back to Basics</a> ; <a href="#">Behavioral Objectives</a> ; <a href="#">Competence</a> ; <a href="#">Individualized Instruction</a> ; <a href="#">Minimum Competencies</a> ; <a href="#">Minimum Competency Testing</a> ; <a href="#">Outcome Based Education</a> ; <a href="#">Performance</a> ; <a href="#">Performance Based Assessment</a> ; <a href="#">Student Certification</a> ;
Used For:	Consequence Based Education; Criterion Referenced Education; Output Oriented Education; Performance Based Education (1974 1980); Proficiency Based Education
Use Term:	
Use And:	
Add Date:	03/10/1980

# Степен на контрол

- **Неконтролиран (Uncontrolled)** – Всеки може да добавя нещо по всяко време и то без усилия да запази нещата консistentни. Възможни са множество списъци и варианти.
- **Управляван (Managed)** – Софтуерът поддържа списък, който е в гарантирана консистентност (без повторения, няма ‘висящи’ възли) във всеки един момент. Почти всеки може да добави нещо, при спазване правилата на съвместимостта.
- **Контролиран (Controlled)** – Поддържа документиран процес за актуализация на речника. Малко хора имат правомощия да променят списъка от термини.

+/- от контрола

- Контролът повишава цената, но може да осигури значителни ползи относно качеството на данните и от намаляване на вариациите

# За допълнителен прочит

- RDF Home Page
  - <http://www.w3.org/RDF>
- Dublin Core Metadata Initiative
  - <http://purl.org/dc/>
- “Using Dublin Core”
  - <http://dublincore.org//documents/usageguide/>

# FOAF речник (<http://www.foaf-project.org/>)

- FOAF=*Friend of a Friend*
- Namespace URI: <http://xmlns.com/foaf/0.1>
- Технология за лесно споделяне и използване на информация за хора и техните дейности (напр. снимки, календари, блогове), за предаване на информация между уеб сайтове, както и автоматично разширяване, сливатне и използване на страници
- FOAF страниците се обработват от програми и описват хора, връзки между тях и техните артефакти.

# Основни категории във FOAF

- FOAF Basics
- Personal Info
- Online Accounts
- Projects and Groups
- Documents and Images

„Un amigo me dijo que un amigo le dijo...“ ☺

# FOAF речник: основни елементи

- Agent
- Person
- name
- nick
- title
- homepage
- mbox
- mbox\_sha1sum
- img
- depiction (depicts)
- surname
- family\_name
- givenname
- firstName

# Пример

```
<foaf:Person>
 <foaf:name>Behrad Zari</foaf:name>
 <foaf:mbox>zari@ce.sharif.edu</foaf:mbox>
 <foaf:homepage
 rdf:resource="http://www.xyz.ir/jrad"/>
 <foaf:img
 rdf:resource="http://www.xyz.ir/jrad/pic-
 small.jpeg"/>
</foaf:Person>
```

# FOAF речник: Personal Info

- weblog
- knows
- interest
- currentProject
- pastProject
- plan
- based\_near
- workplaceHomepage
- workInfoHomepage
- schoolHomepage
- topic\_interest
- publications
- myersBriggs
- dnaChecksum

# FOAF речник: Online Accounts

- OnlineAccount
- OnlineChatAccount
- OnlineEcommerceAccount
- OnlineGamingAccount
- holdsAccount
- accountServiceHomepage
- accountName
- icqChatID
- msnChatID
- aimChatID
- jabberID
- yahooChatID

# FOAF речник: Projects and Groups

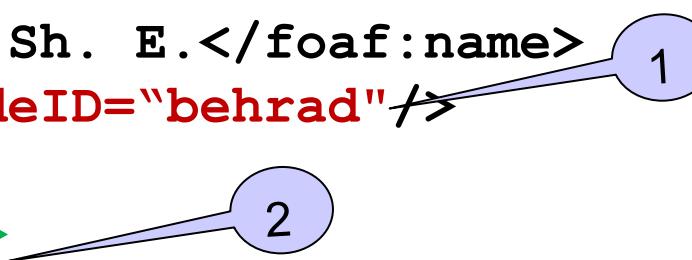
- Project
- Organization
- Group
- member
- membershipClass
- fundedBy
- theme

# FOAF речник: Documents and Images

- Document
- Image
- PersonalProfileDocument
- topic (page)
- primaryTopic
- tipjar
- sha1
- made (maker)
- thumbnail
- logo

# Два начина за изразяване на връзки

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<foaf:Person rdf:nodeID="behrad">
 <foaf:name>Behrad Zari</foaf:name>
</foaf:Person>
<foaf:Person>
 <foaf:name>Kiumars Sh. E.</foaf:name>
 <foaf:knows rdf:nodeID="behrad" />
 <foaf:knows>
 <foaf:Person>
 <foaf:name>Y. Ganji</foaf:name>
 </foaf:Person>
 </foaf:knows>
</foaf:Person>
</rdf:RDF>
```

## За повече информация

- FOAF Vocabulary Specification -  
<http://xmlns.com/foaf/spec/>
- The Friend of a Friend (FOAF) project -  
<http://www.foaf-project.org/>

# RSS



- RSS (Rich Site Summary или Really Simple Syndication, оригинално RDF Site Summary) използва семейство от стандартни web feed формати за публикуване на често опреснявана информация от новинарски сайтове, блогове, и др.
- Стандартният формат на XML файл позволява информацията да бъде публикувана веднъж и да се разглежда от много различни програми
- Ползата от RSS е обобщаването на цялото съдържание от няколко уеб източника на едно място. Вече не трябва да посещаваме различни уеб сайтове, за да получим най-новата информация по темите, от които се интересуваме.

# RSS пример

(<http://en.wikipedia.org/wiki/RSS>)

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel> <title>RSS Title</title>
<description>This is an example of an RSS feed</description>
<link>http://www.someexamplerssdomain.com/main.html</link>
<lastBuildDate>Mon, 06 Sep 2013 </lastBuildDate>
<pubDate>Mon, 06 Sep 2013 </pubDate>
<ttl>1800</ttl>
<item> <title>Example entry</title>
<description>Here is some text containing</description>
<link>http://www.wikipedia.org/</link>
<guid>unique string per item</guid>
<pubDate>Mon, 06 Sep 2013 </pubDate>
</item> </channel> </rss>
```

# How to Add an RSS Feed to a Web Page?

- <https://www.lifewire.com/how-to-add-rss-feed-3469294>

```

```



# SPARQL 1/2

- SPARQL (произнася се "sparkle,, и идва от SPARQL Protocol and RDF Query Language)
- RDF език за заявки за извлечане и манипулиране на данни, съхранявани в Resource Description Framework формат
- От 2008г., SPARQL 1.0 е препоръка на W3C
- SPARQL заявката може да включва triple patterns, conjunctions, disjunctions + optional patterns.

# SPARQL 2/2

- SPARQL осигурява средства за:
  - Извличане на информация под формата на URI адреси, празни възли, обикновени и типизирани литерали.
  - Извличане на RDF подграфи.
  - Изграждане на нови RDF графи на базата на информация в претърсваните графи
- Сравняване на шаблони в графи
- Променливи - глобален обхват; обозначени с "\$" или "?"
- Условия на заявки - въз основа на Turtle синтаксис
- Описание на данни – в Turtle формат

# Пример ([https://en.wikipedia.org/wiki/RDF\\_Schema](https://en.wikipedia.org/wiki/RDF_Schema))

In English	The graph
<ul style="list-style-type: none"> <li>Dog1 is an animal</li> <li>Cat1 is a cat</li> <li>Cats are animals</li> <li>Zoos host animals</li> <li>Zoo1 hosts the Cat2</li> </ul>	<pre> graph LR     dog1((ex:dog1)) -- "rdf:type" --&gt; animal((ex:animal))     cat1((ex:cat1)) -- "rdf:type" --&gt; cat((ex:cat))     cat -- "rdfs:subClassOf" --&gt; animal     zoo1((ex:zoo1)) -- "zoo:host" --&gt; cat2((ex:cat2))     </pre> <p>RDF special terms      RDFS special terms</p>

## RDF/turtle

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
@prefix zoo: <http://example.org/zoo/> .

ex:dog1 rdf:type ex:animal .
ex:cat1 rdf:type ex:cat .
ex:cat rdfs:subClassOf ex:animal .
zoo:host rdfs:range ex:animal .
ex:zoo1 zoo:host ex:cat2 .

```

```

PREFIX ex: <http://example.org/>
SELECT ?animal
WHERE {
 ?animal a ex:animal .
}

```

# Пример (<https://en.wikipedia.org/wiki/SPARQL>)

**PREFIX foaf:**

<<http://xmlns.com/foaf/0.1/>>

**SELECT** ?name

?email

**WHERE** {

?person **a foaf:Person** .

?person **foaf:name** ?name .

?person **foaf:mbox** ?email .

}

# За домашна работа

Запознайте се със:

- SPARQL Query Language for RDF  
*от Cristina Feier,*

<http://www.gise.cse.iitb.ac.in/wiki/images/2/25/SPARQL.pdf>

- SPARQL Tutorial,

<https://jena.apache.org/tutorials/sparql.html>

# Въведение в онтологии и OWL



Онтологии  
OWL  
Типове  
Конструкции  
Примери

# Използвани ресурси

- Илюстрации и коментари: Vagan Terziyan
- W3C документи
  - Guide: <http://www.w3.org/TR/owl-guide/>
  - Reference: <http://www.w3.org/TR/owl-ref/>
  - Semantics and Abstract Syntax:  
<http://www.w3.org/TR/owl-semantics/>
- OWL справочници
  - Ian Horrocks, Sean Bechhofer:  
<http://www.cs.man.ac.uk/~horrocks/Slides/Innsbruck-tutorial/>
  - Roger L. Costello, David B. Jacobs:  
<http://www.xfront.com/owl/>
- Примерни онтологии:  
<http://www.daml.org/ontologies/>

# Онтологична визия за Семантичния Уеб

## Необходимост от онтологии

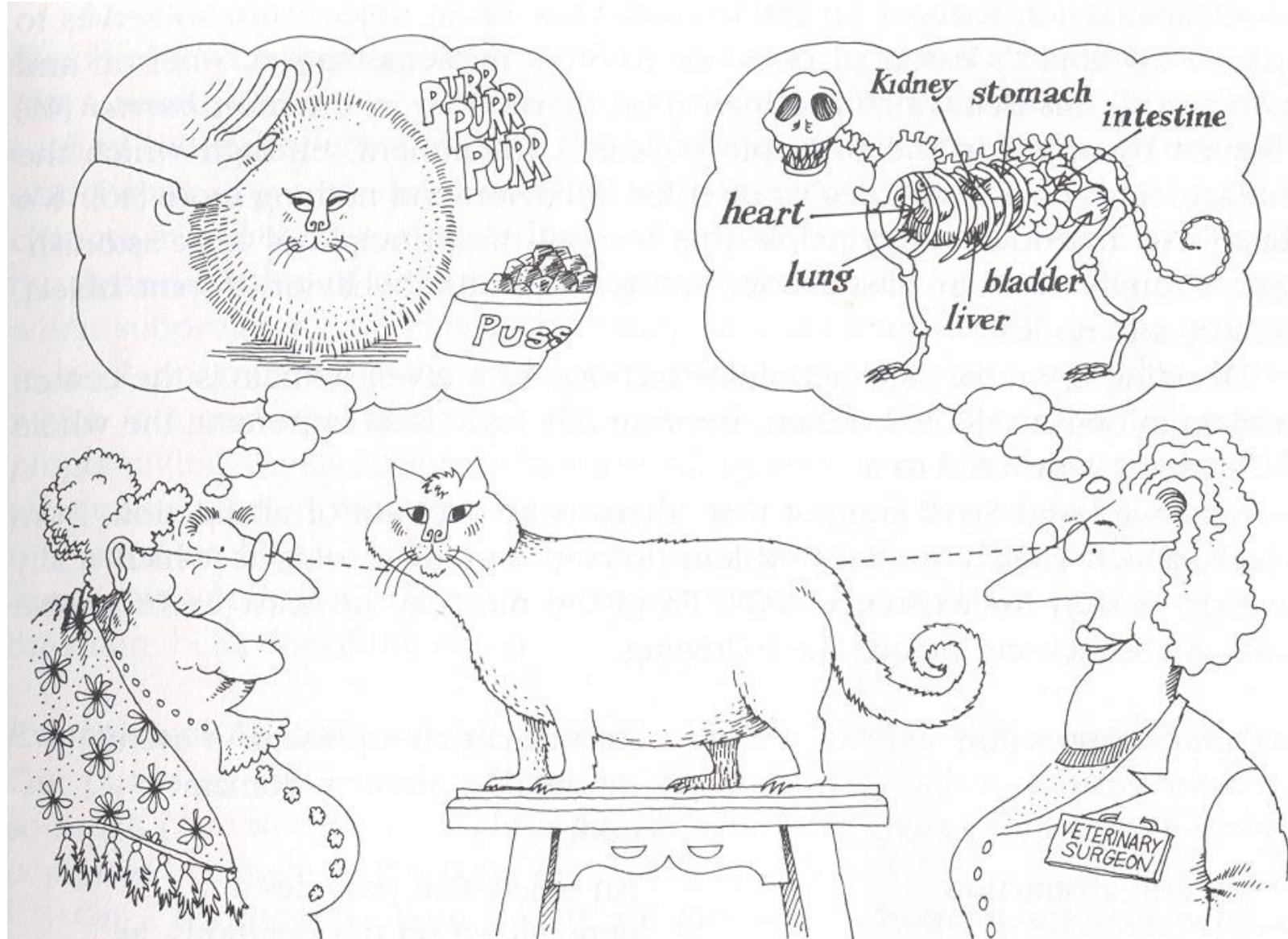
### Онтологията е:

- документ, който официално и по стандартизиран начин определя юрархията на класовете в рамките на домейн, семантичните отношения между термовете (понятията) и правилата за извод

### Използване на онтологии:

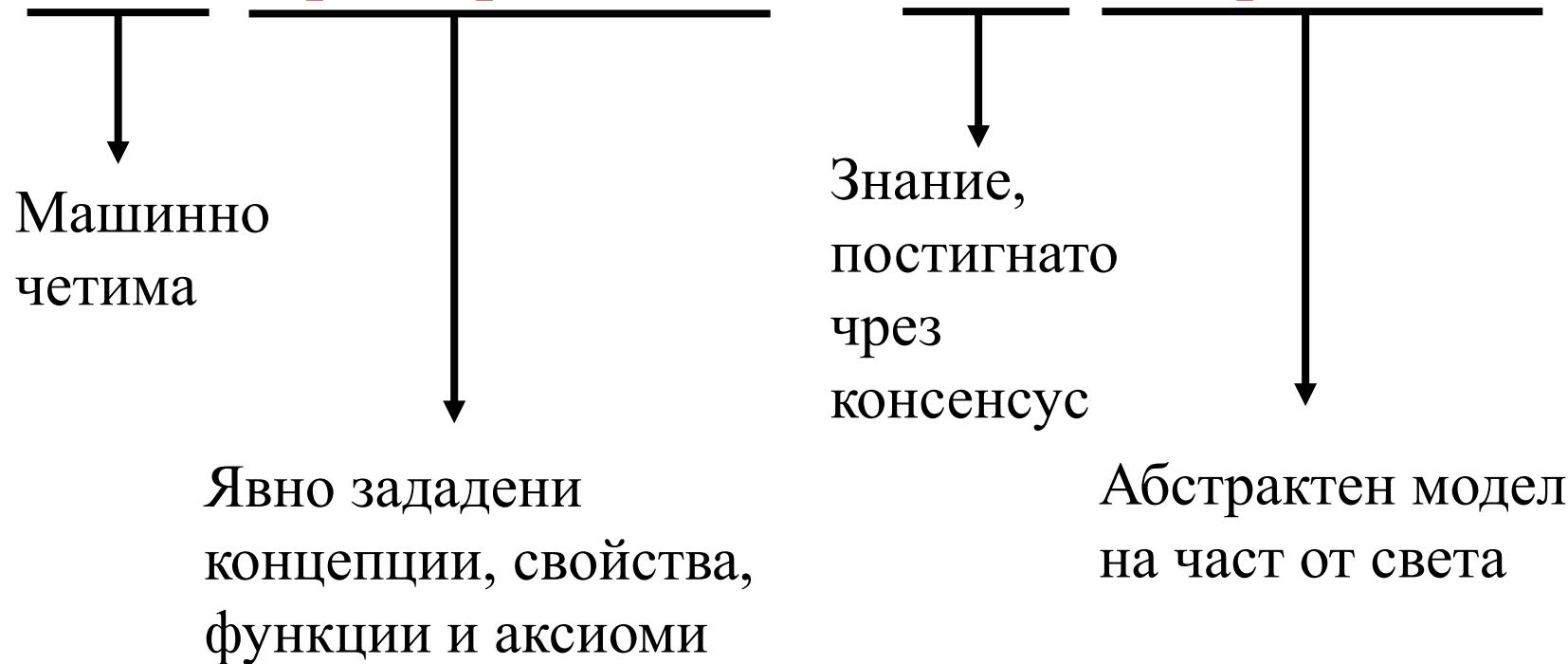
- споделянето на семантиката на данните между разпределени приложения

# Комуникации между хора



# Какво е онтология?

**Formal, explicit specification of a shared conceptualization**

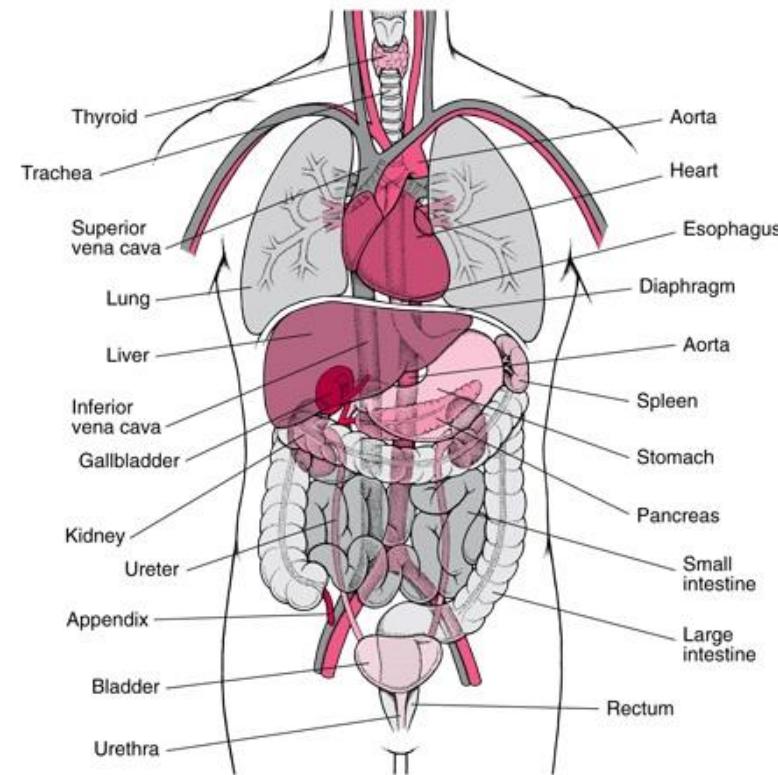


**Tom Gruber, Knowledge Systems Laboratory at Stanford University**

# Какво е онтология?

Модел на (някои от) аспектите на част от света

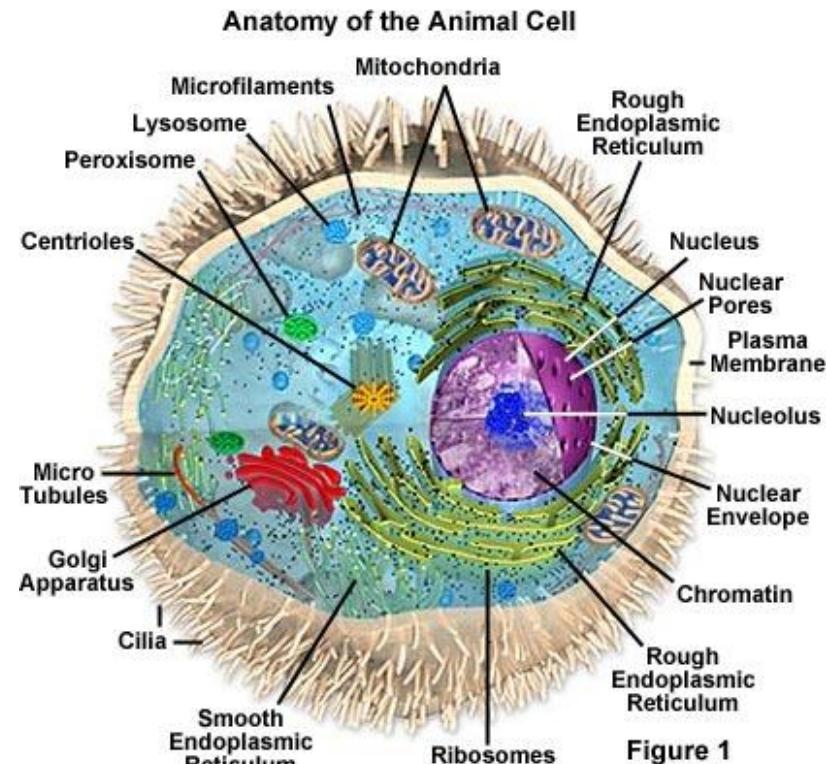
- Въвежда **речник** свързан с предметната област, напр.:
  - Анатомия



# Какво е онтология?

Модел на (някои от) аспектите на част от света

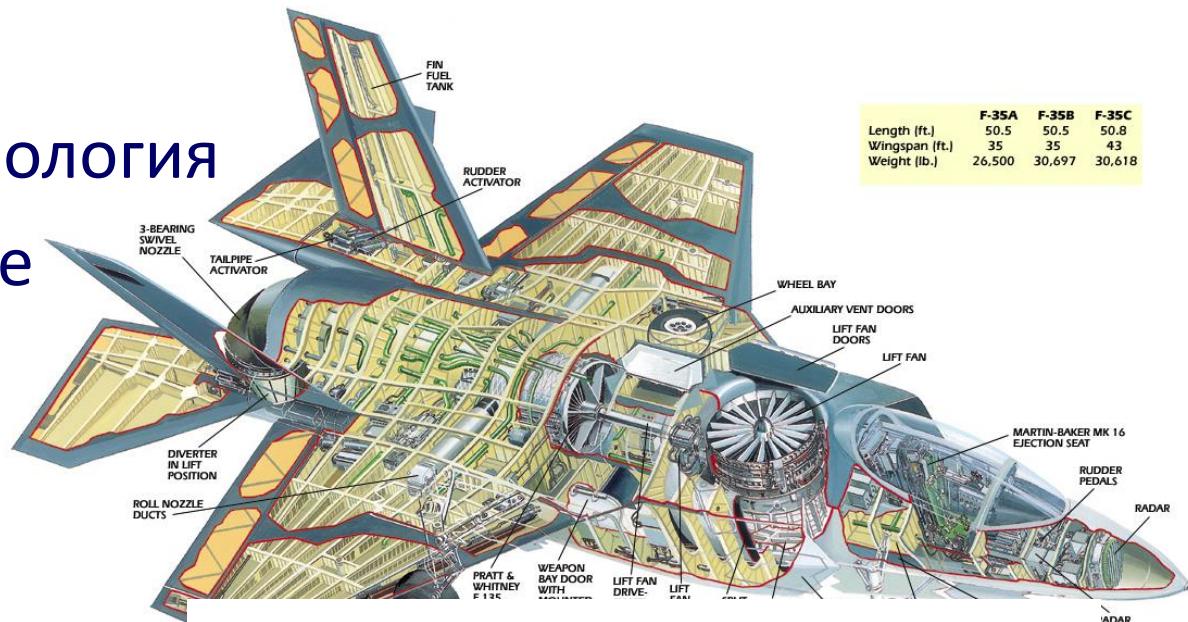
- Въвежда **речник** свързан с предметната област, напр.:
  - Анатомия
  - Молекулярна биология



# Какво е онтология?

Модел на (някои от) аспектите на част от света

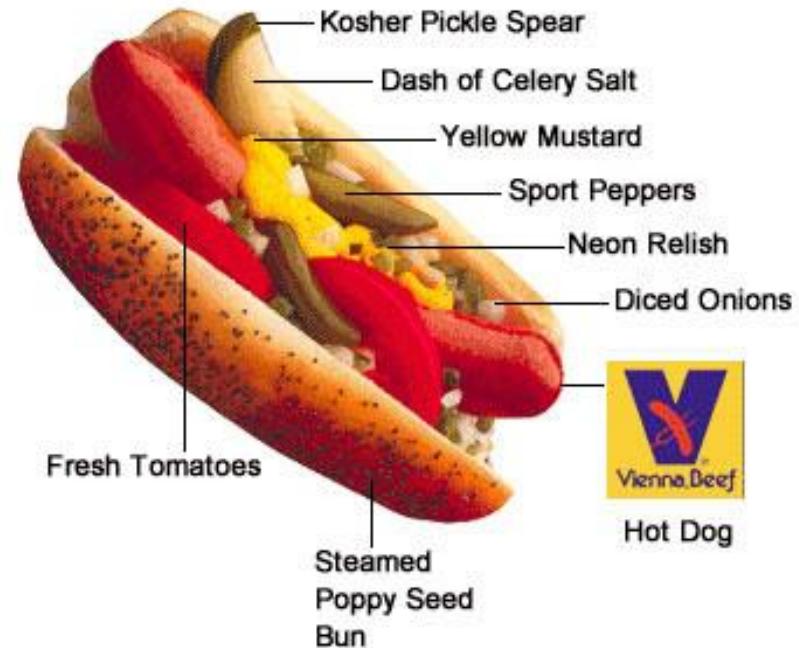
- Въвежда **речник** свързан с предметната област, напр.:
  - Анатомия
  - Молекулярна биология
  - Самолетостроене



# Какво е онтология?

Модел на (някои от) аспектите на част от света

- Въвежда **речник** свързан с предметната област, напр.:
  - Анатомия
  - Молекулярна биология
  - Самолетостроене
  - Сандвичи

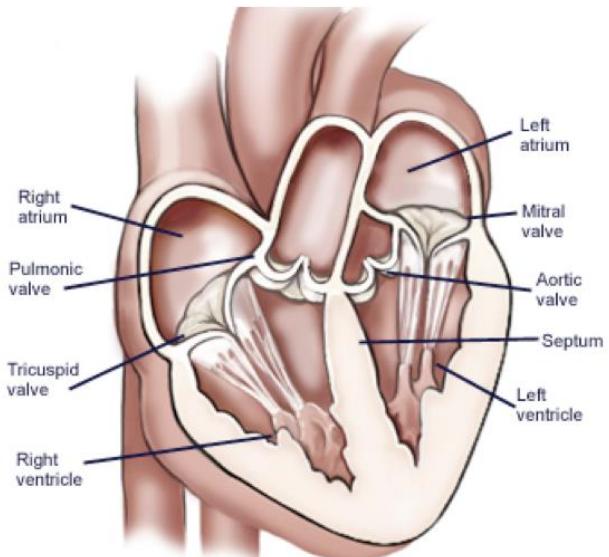


# Какво е онтология?

Модел на (някои от) аспектите на част от света

- Въвежда **речник** свързан с предметната област
- Определя **значението** на термините:

**Heart is a muscular organ that  
is part of the circulatory system**



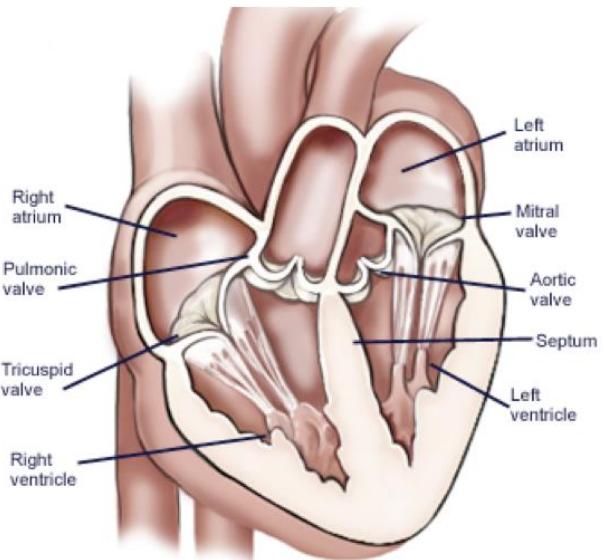
# Какво е онтология?

Модел на (някои от) аспектите на част от света

- Въвежда **речник** свързан с предметната област
- Определя **значението** на термините:

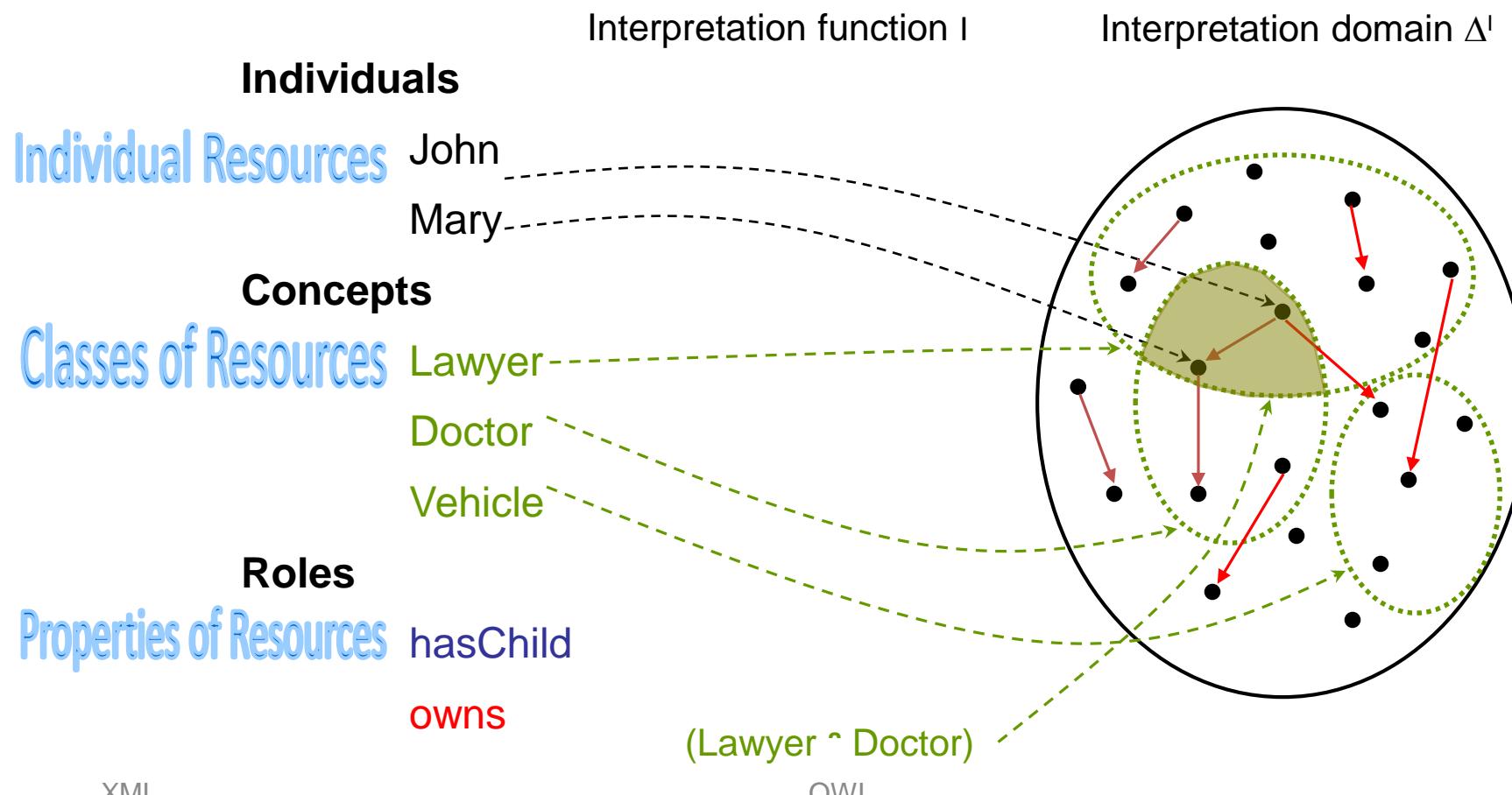
Heart **is a muscular organ that is part of the circulatory system**

- **Формализация** чрез логика

$$\begin{aligned} \forall x. [\text{Heart}(x) \rightarrow \text{MuscularOrgan}(x) \wedge \\ \exists y. [\text{isPartOf}(x, y) \wedge \\ \text{CirculatorySystem}(y)]] \end{aligned}$$


# DL семантика

Задаване на семантика:



# Елементи на онтологиите

- Концепции (класове) + тяхната йерархия
- Свойства на класовете (слотове/атрибути)
- Ограничения над с-ва (type, cardinality, domain)
- Релации м/у концепции (disjoint, equality)
- Екземпляри (Instances)

# Как изграждаме онтология?

## Стъпки:

- определяме домейн и обхват
- изброяваме важните термини
- дефинираме класове и клас-йерархии
- дефинираме слотове (свойства)
- дефинираме ограничения над с-ва  
(кардиналност, тип на стойност, ...)

# Стъпка1: Определете областта и обхвата

**Домейн:** география

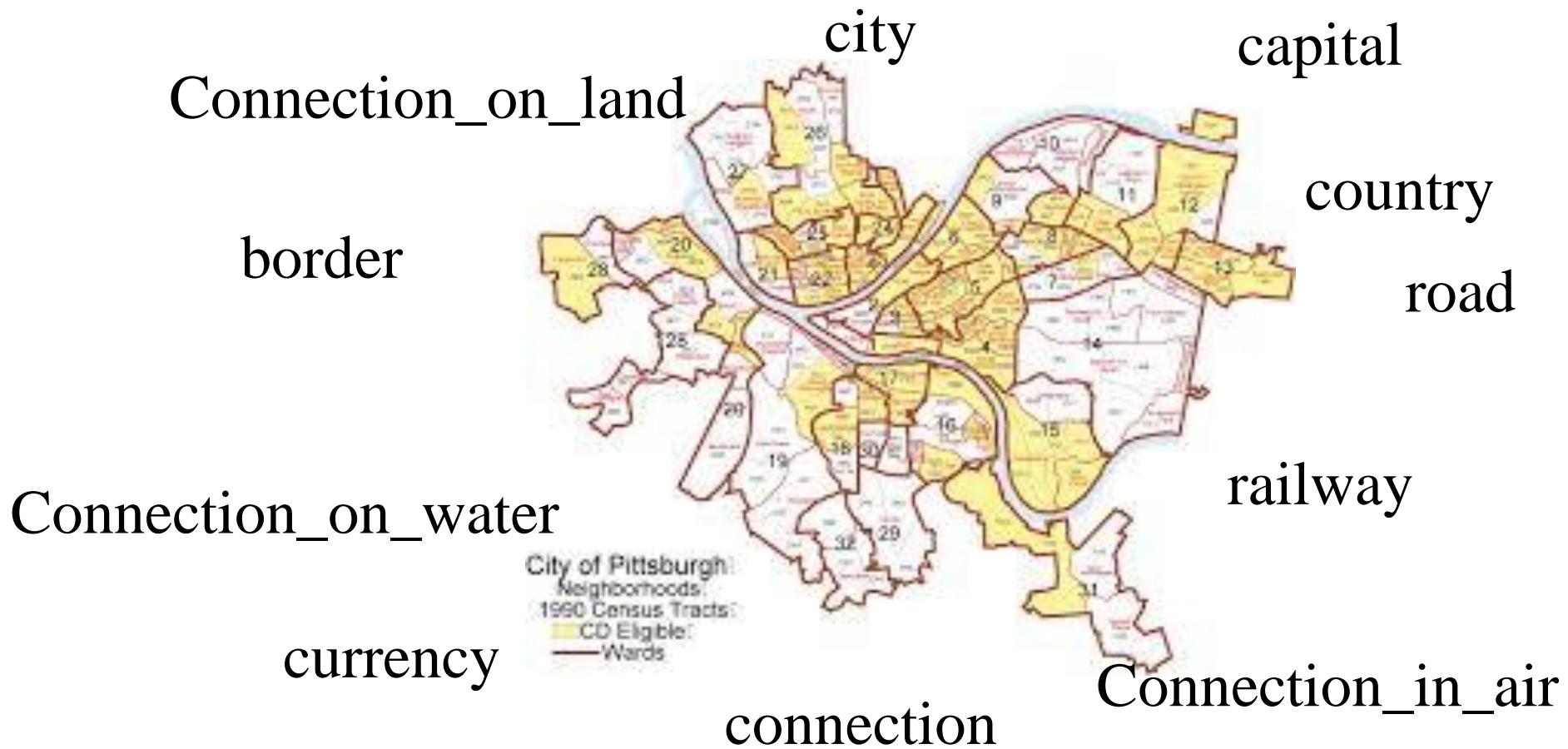
**Приложение:** агент-маршрутизатор

**Възможни въпроси:**

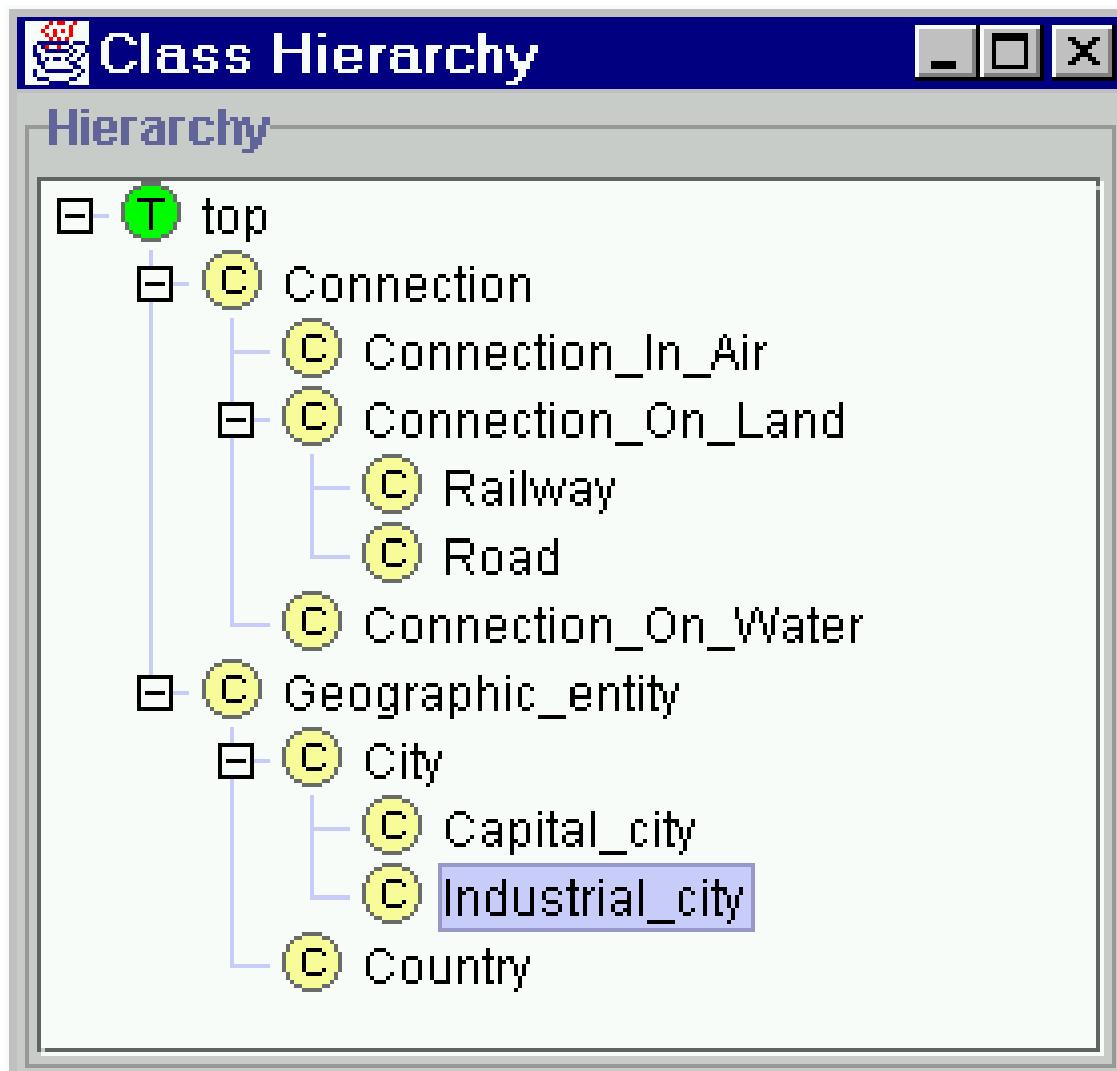
- Разстояние между два града?
- Какви са връзките между два града?
- В коя държава е даден град?
- Колко граници се пресичат?



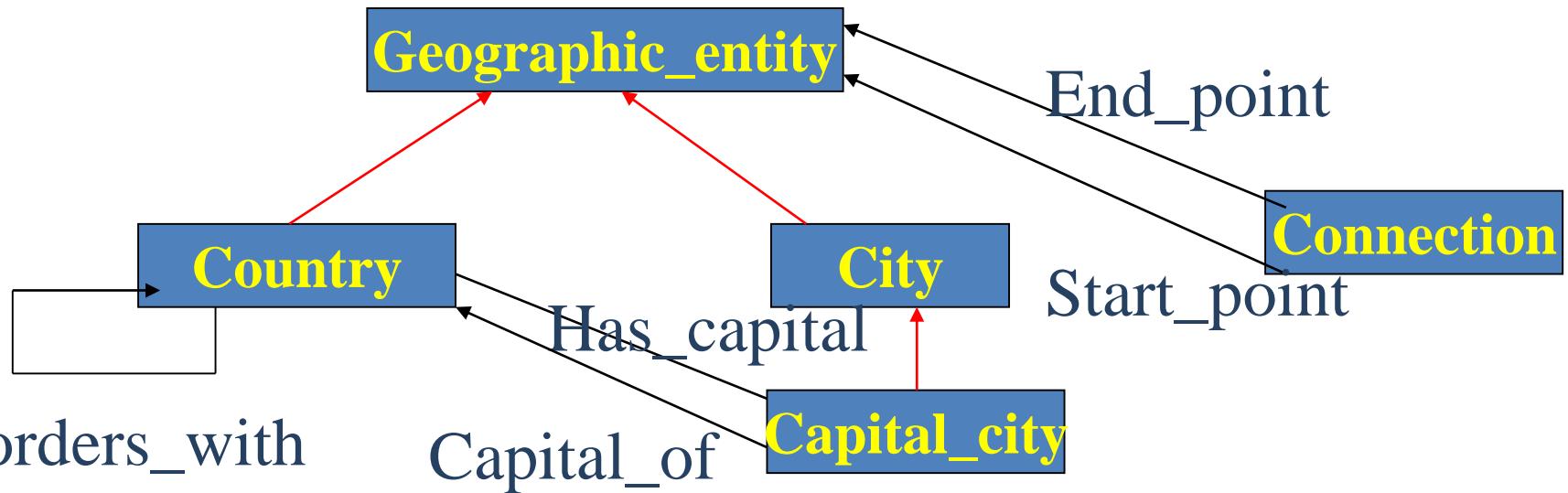
## Стъпка 2: Избройте важните термини



# Стъпка 3: Определяне на класове и йерархия



## Стъпка 4: Определяне на релации между класовете



## Стъпка 5/6: Слотове и ограничения

- Slot-cardinality

*Ex: Borders\_with multiple, Start\_point single*

- Slot-value type

*Ex: Borders\_with- Country*

# Стъпка 7: Правила, аксиоми и събития

**Правила:** съждения (statements) във формата на *if-then* (*antecedent-consequent*) изречения, които описват логическите изводи, които могат да се извлекат от едно твърдение в определена форма

**Аксиоми:** твърдения (включително правила) в логическа форма, които заедно съставляват цялостната теория, описана от онтологията в своята област на приложение. За разлика от генеративните граматики и формалната логика, където аксиомите включват само изявления, утвърдени като априорни знания, тук аксиомите също така включват различни деривации.

**Събития:** промяна на атрибутите или отношения

# Web Ontology Language (OWL)

10 февруари 2004 г. World Wide Web Consortium съобщи окончателното одобрение на два ключови Semantic Web технологии, ревизираният Resource Description Framework (RDF) и **Web Ontology Language (OWL)**.

<http://www.w3.org/2004/01/sws-pressrelease.html.en>

# Въведение в OWL

- Какво е OWL?

- OWL е език за дефиниране на уеб онтологии и свързаните с тях бази от знания (Knowledge Bases)
- OWL е преразглеждане на DAML+OIL (Web-based representation and inference layer for ontologies) езици за уеб онтологии, относно използването им в практиката



# Пример

- Има два типа животни, Male и Female.

```
<rdfs:Class rdf:id="Male">
 <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
```

- Елементът `subClassOf` element заявява, че субектът му - `Male` – е под клас на обекта му - ресурсът `#Animal`.

```
<rdfs:Class rdf:id="Female">
 <rdfs:subClassOf rdf:resource="#Animal"/>
 <owl:disjointWith rdf:resource="#Male"/>
</rdfs:Class>
```

- Някои животни са `Female`, но никое не е `Male` и `Female` (в тази онтология), понеже двета класа са disjoint (зададено чрез елемента `disjointWith`).

# OWL пример в Protégé (1)

- Клас (Class)
  - Person superclass
  - Man, Woman subclasses
- Свойства (Properties)
  - isWifeOf, isHusbandOf
- Х-ки на свойства, ограничения (restrictions)
  - inverseOf
  - domain
  - range
  - cardinality
- Изрази с класове
  - disjointWith

# OWL Example in Protégé (2)

MyOntology Protégé 2.0 beta (file:/C:/ellisr/ontology/MyOntology.pprj, OWL files) [ ] [ ] X

Project Edit Window OWL Help

OWLClasses Properties Forms Individuals Ontology

Class Hierarchy V C < > X

- :THING
- Person
  - Woman
  - Man

Annotations V C + -

Property	Value

Properties at Class V C X + -

Name	Type	Cardinality	Other Facets
isWifeOf	Instance	single	classes={(Man)}

Restrictions V C Υc Ξc ≈c =c ≥c ≤c Filler

Property	Restriction	Filler

Definition V C + - X

Disjoint classes V C + E + - X

- Man

XML

OWL

31

# OWL пример в Protégé (3)

MyOntology Protégé 2.0 beta (file:/C:/ellisr/ontology/MyOntology.pprj, OWL files)

Project Edit Window OWL Help

OWLClasses Properties Forms Individuals Ontology

Properties V C X

isHusbandOf (type=owl:ObjectProperty)

Name Labels SameAs DifferentFrom Annotations

isHusbandOf

Documentation

Cardinality HasValue V C + -

required  at least  1

multiple  at most  1

Equivalent V C + -

Range Some Values From

Instance

Classes V C + - X

Woman

Domain defined

Domain V C + -

Man

Inverse Property V C + -

isWifeOf

OWL Symmetric Transitive AnnotationProperty InverseFunctional

XML 32

# OWL в един слайд

- **Symmetric:** if  $P(x, y)$  then  $P(y, x)$
- **Transitive:** if  $P(x, y)$  and  $P(y, z)$  then  $P(x, z)$
- **Functional:** if  $P(x, y)$  and  $P(x, z)$  then  $y=z$
- **InverseOf:** if  $P_1(x, y)$  then  $P_2(y, x)$
- **InverseFunctional:** if  $P(y, x)$  and  $P(z, x)$  then  $y=z$
- **allValuesFrom:**  $P(x, y)$  and  $y=\text{allValuesFrom}(C)$
- **someValuesFrom:**  $P(x, y)$  and  $y=\text{someValuesFrom}(C)$
- **hasValue:**  $P(x, y)$  and  $y=\text{hasValue}(v)$
- **cardinality:**  $\text{cardinality}(P) = N$
- **minCardinality:**  $\text{minCardinality}(P) = N$
- **maxCardinality:**  $\text{maxCardinality}(P) = N$
- **equivalentProperty:**  $P_1 = P_2$
- **intersectionOf:**  $C = \text{intersectionOf}(C_1, C_2, \dots)$
- **unionOf:**  $C = \text{unionOf}(C_1, C_2, \dots)$
- **complementOf:**  $C = \text{complementOf}(C_1)$
- **oneOf:**  $C = \text{one of}(v_1, v_2, \dots)$
- **equivalentClass:**  $C_1 = C_2$
- **disjointWith:**  $C_1 \neq C_2$
- **sameIndividualAs:**  $I_1 = I_2$
- **differentFrom:**  $I_1 \neq I_2$
- **AllDifferent:**  $I_1 \neq I_2, I_1 \neq I_3, I_2 \neq I_3, \dots$
- **Thing:**  $I_1, I_2, \dots$

Properties limited to a single value

## Legend:

Properties are indicated by:  $P, P_1, P_2$ , etc

Specific classes are indicated by:  $x, y, z$

Generic classes are indicated by:  $C, C_1, C_2$

Values are indicated by:  $v, v_1, v_2$

Instance documents are indicated by:  $I_1, I_2, I_3$ , etc.

A number is indicated by:  $N$

$P(x, y)$  is read as: “property  $P$  relates  $x$  to  $y$ ”

# Други примери

- $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
- $\text{Man} \equiv \text{Person} \sqcap \neg \text{Woman}$
- $\text{Mother} \equiv \text{Woman} \sqcap \exists \text{hasChild}.\text{Person}$
- $\text{Father} \equiv \text{Man} \sqcap \exists \text{hasChild}.\text{Person}$
- $\text{Parent} \equiv \text{Father} \sqcup \text{Mother}$
- $\text{Grandmother} \equiv \text{Mother} \sqcap \exists \text{hasChild}.\text{Parent}$

Можем да заключим (макар да не е изрично дефинирано), че:

- $\text{Grandmother} \sqsubseteq \text{Person}$
- $\text{Grandmother} \sqsubseteq \text{Woman}$

# Наръчник: проектиране на онтологии с Protégé

- Protégé е редактор за онтологии и бази знания (<http://protege.stanford.edu> ).
- Protégé е Java инструмент с отворен код, който предоставя разширяема архитектура за създаването на персонализирани приложения.
- Наличен е също така и онлайн на <http://webprotege.stanford.edu/>

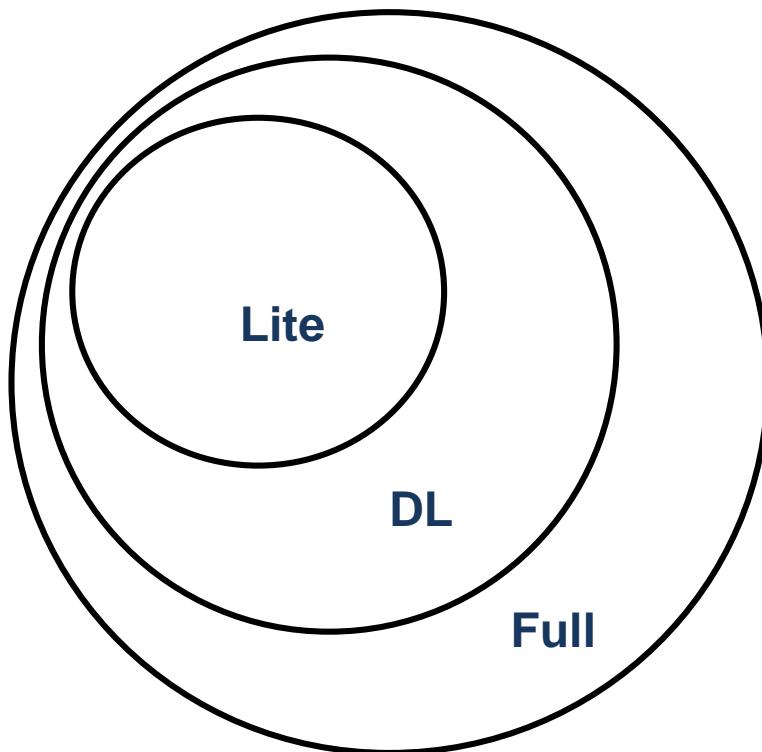
# OWL ни помага...

...да опишем нещо, вместо само да го именоваме.

Класът (BlueThing) няма смисъл само по себе си

Но класът (BlueThing complete  
owl:Thing  
restriction (hasColour someValuesFrom (Blue)))

# OWL се предлага в три варианта



## **Lite**

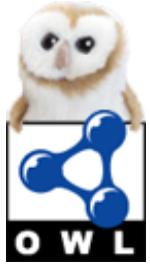
частично ограничен, за да се подпомогне кривата на обучение

## **DL = Description Logic**

Дескриптивната логика е вид First Order Logic (FOL) и служи за разсъждения

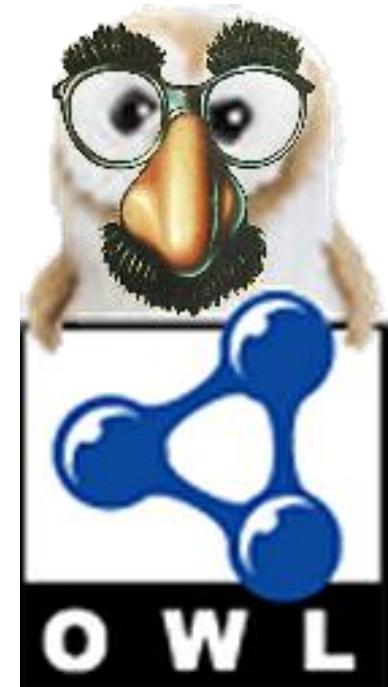
## **Full**

неограничено ползване на OWL конструкции, но не предполага разсъждения



# OWL

- OWL често се смята за разширение на RDF, което не е съвсем вярно
- OWL е синтактично независим език, който има няколко общи представления
  - Abstract Syntax
  - N3
  - RDF / XML



# OWL синтаксис: abstract syntax

- Най-ясен и четим за човека  
синтаксис

```
Class(SpicyPizza complete
 annotation(rdfs:label "PizzaTemperada"@pt)
 annotation(rdfs:comment "Any pizza that
has a spicy topping is a SpicyPizza"@en)
 Pizza
 restriction(hasTopping
 someValuesFrom(SpicyTopping)))
```

Axioms are used to associate class and property identifiers with either **partial** or **complete** specifications of their characteristics

at least one of the **hasTopping** properties of a **Pizza** must point to an individual that is a **SpicyTopping**

# OWL синтаксис: N3

- Препоръчва се за четими от човек фрагменти

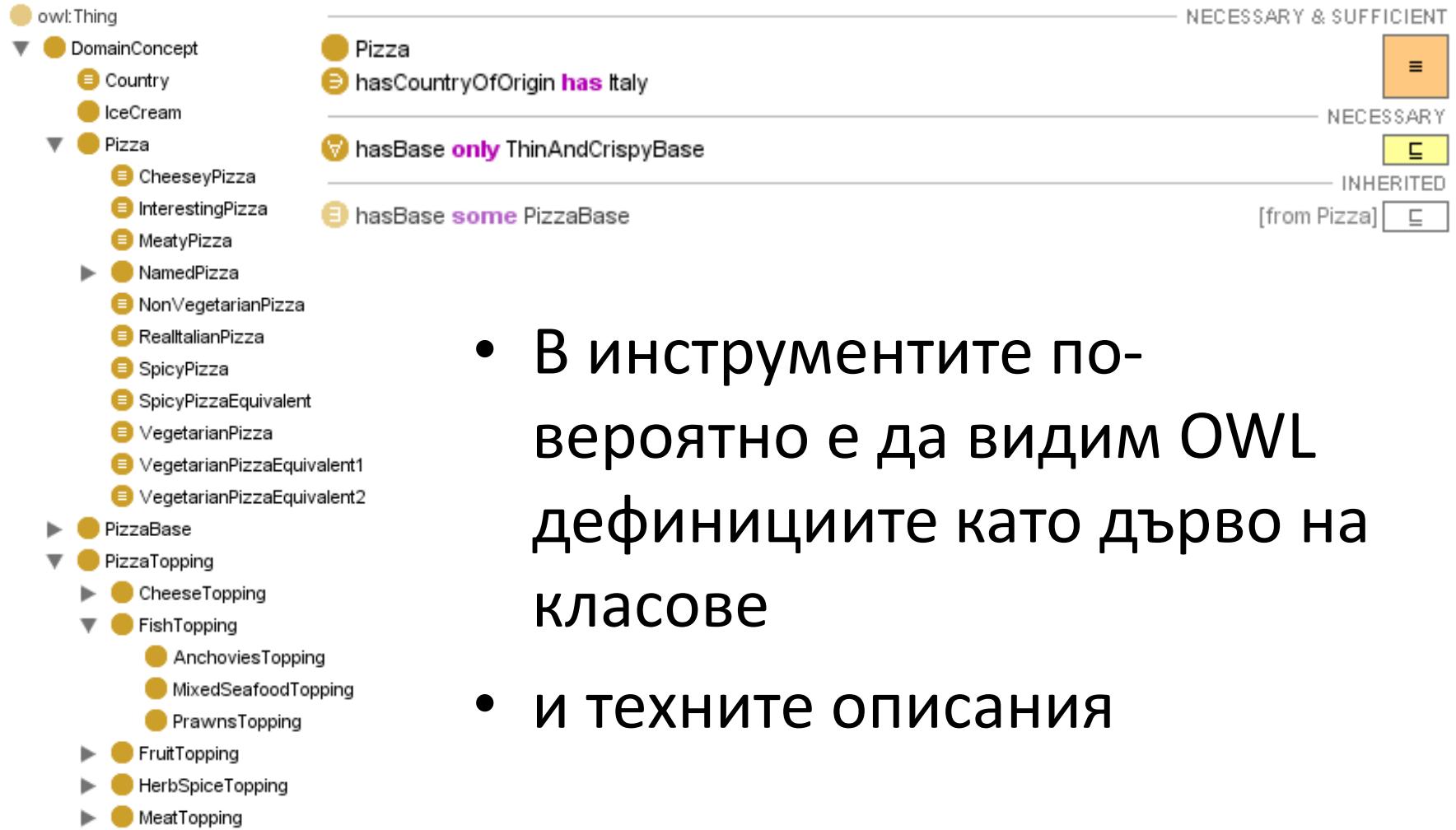
```
Default:SpicyPizza
 a owl:Class ;
 rdfs:comment "Any pizza that has a spicy topping is a
 SpicyPizza"@en ;
 rdfs:label "PizzaTemperada"@pt ;
 owl:equivalentClass
 [a owl:Class ;
 owl:intersectionOf (default:Pizza
 [a owl:Restriction ;
 owl:onProperty default:hasTopping ;
 owl:someValuesFrom default:SpicyTopping
])
]
 .
```

# OWL синтаксис : RDF/XML

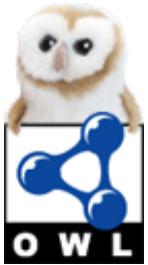
- Препоръчва се за сериализация

```
<owl:Class rdf:ID="SpicyPizza">
 <rdfs:label xml:lang="pt">PizzaTemperada</rdfs:label>
 <rdfs:comment xml:lang="en">Any pizza that has a spicy
topping is a SpicyPizza</rdfs:comment>
 <owl:equivalentClass>
 <owl:Class>
 <owl:intersectionOf rdf:parseType="Collection">
 <owl:Class rdf:about="#Pizza"/>
 <owl:Restriction>
 <owl:onProperty>
 <owl:ObjectProperty rdf:about="#hasTopping"/>
 </owl:onProperty>
 <owl:someValuesFrom rdf:resource="#SpicyTopping"/>
 </owl:Restriction>
 </owl:intersectionOf>
 </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

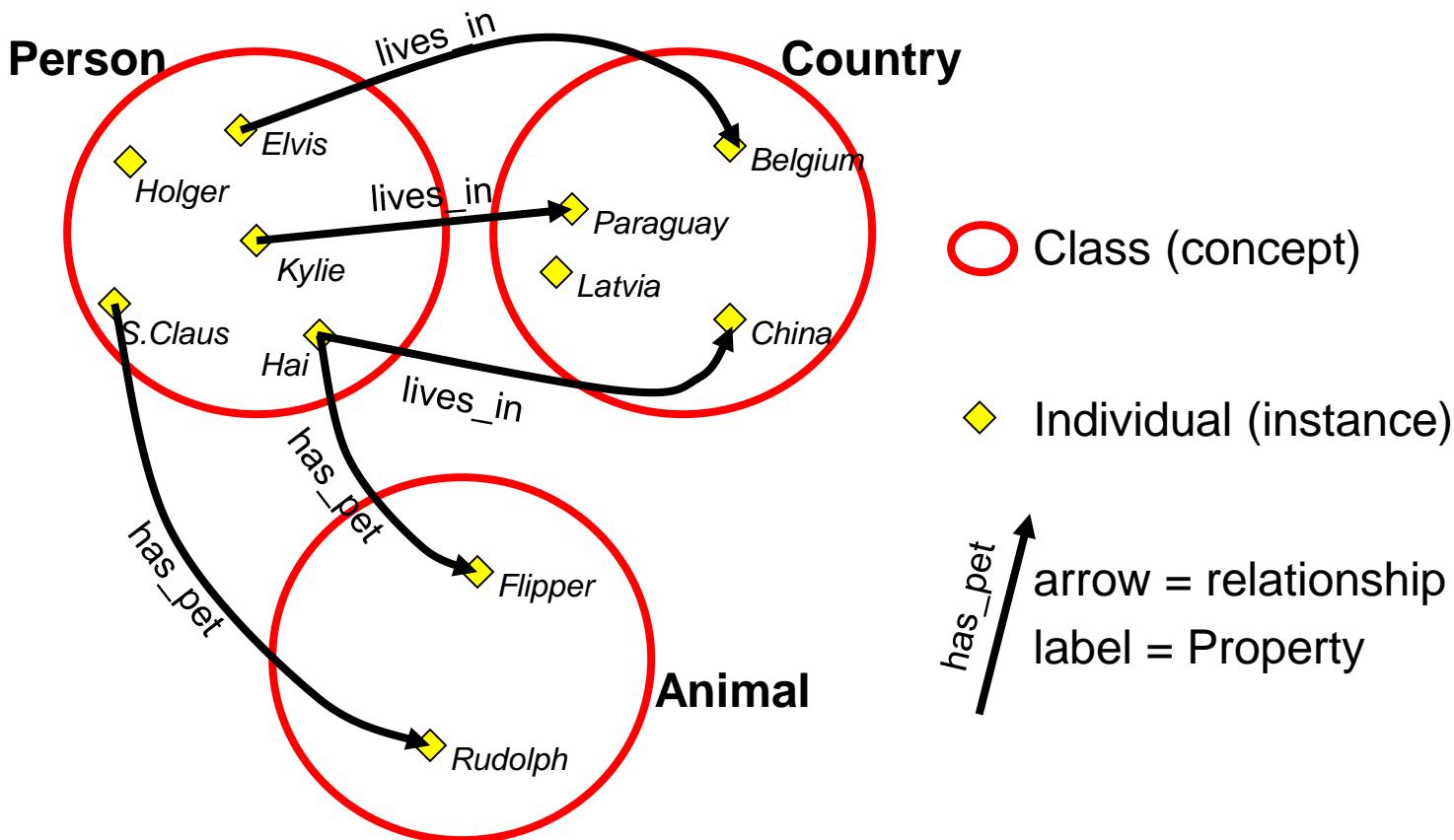
# Програмните инструменти „скриват“ синтаксиса



- В инструментите по-вероятно е да видим OWL дефинициите като дърво на класове
- и техните описания



# OWL конструкции



# OWL конструкции: Classes

**Примери: Mammal, Tree, Person, Building, Fluid, Company**

- Класовете са множества от екземпляри
- като “Type”, “Concept”, “Category”, “Kind”
- Членството в класа зависи от логическото описание, а не от името
- Класът има описание, тип и релации
- Класове не трябва да бъде непременно назовани - те могат да бъдат и логически изрази – например **things that have colour Blue**

# OWL конструкции: Properties

Примери: **hasPart**, **isInhabitedBy**, **isNextTo**, **occursBefore**

- Свойствата (Properties) свързват екземплярите или индивидите (Individuals)
- Индивидите са свързани чрез дадено свойство
- Релациите в OWL са **бинарни**:

Subject → predicate → Object

Individual a → hasProperty → Individual b

nick\_drummond → givesTalk → owl\_overview\_talk\_Dec\_2005

# OWL конструкции: Individuals

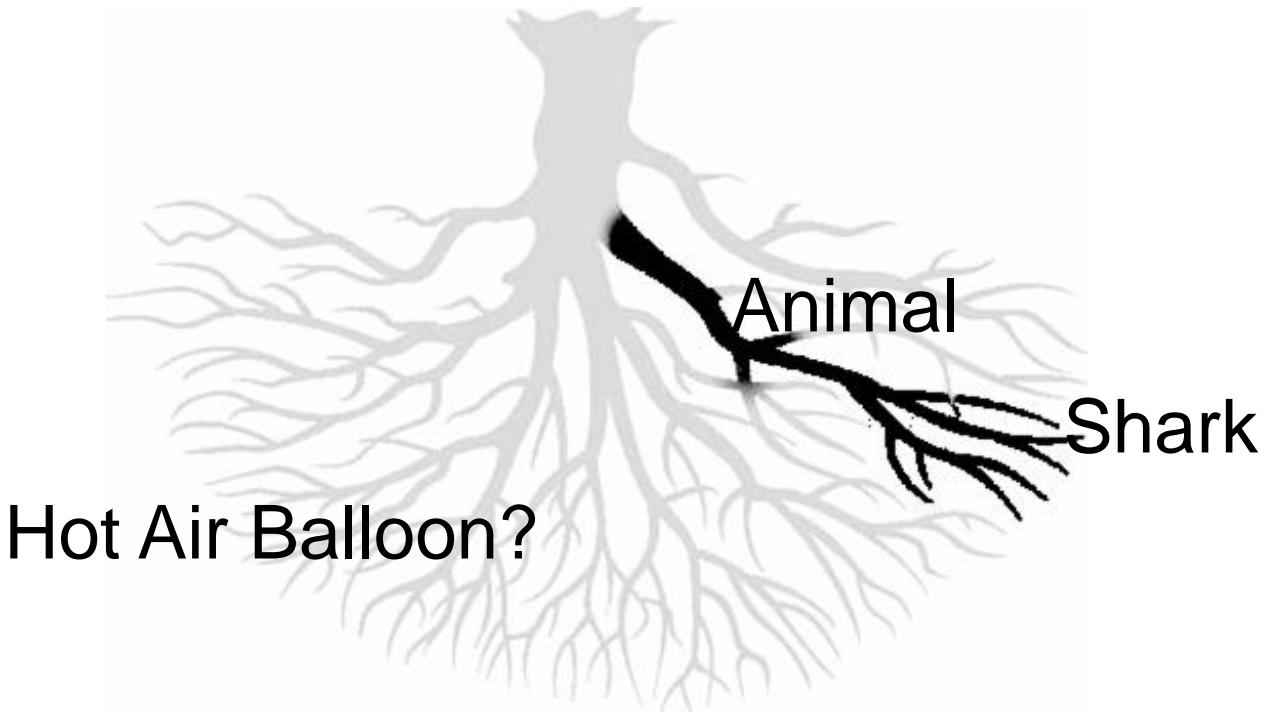
**Примери: me, you, this talk, this room**

- Екземплярите (Individuals) са **обектите** в предметната област
- като “Instance”, “Object”
- Един екземпляр може да бъде (често) член на множество класове

# Описание на йерархия от класове

Две важни неща за класа:

- Къде можем да ги сложим?
- Къде не можем да ги сложим?



owl:Thing

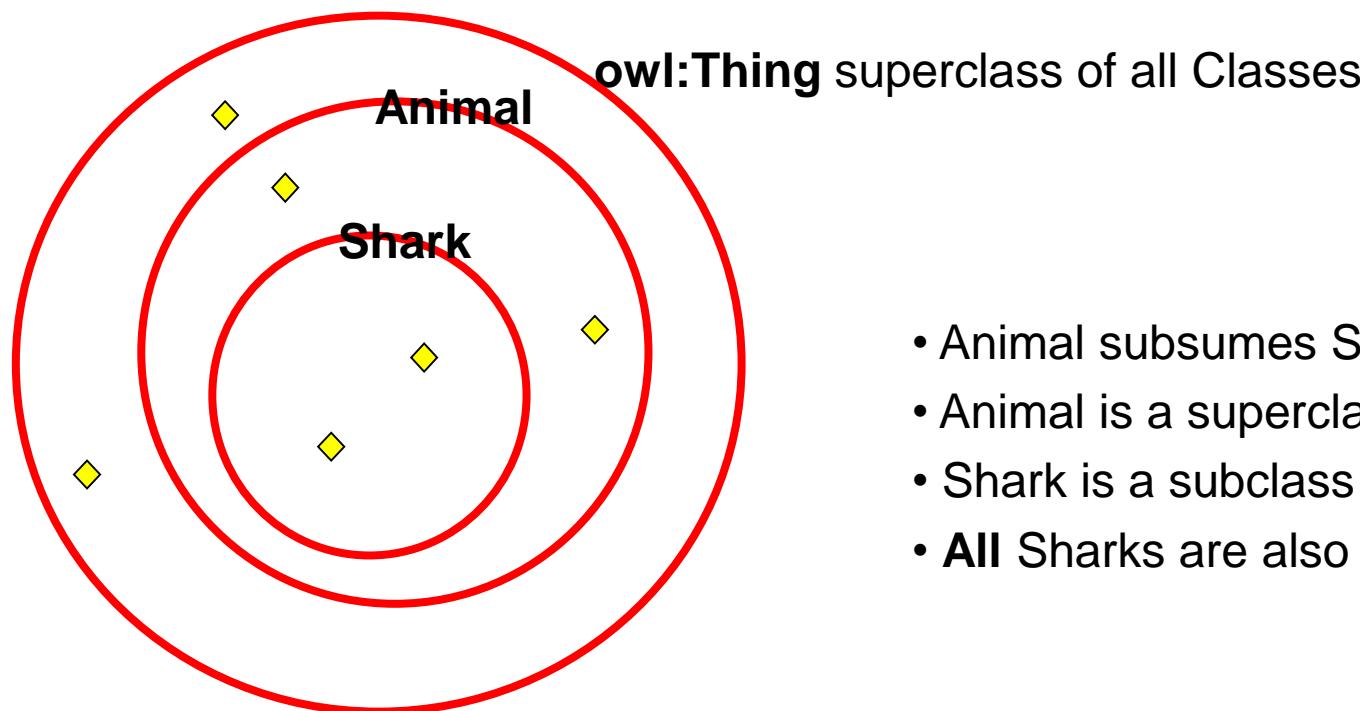
▼ Animal

Shark

# Къде можем да сложим класа?

## Включване (Subsumption) в OWL

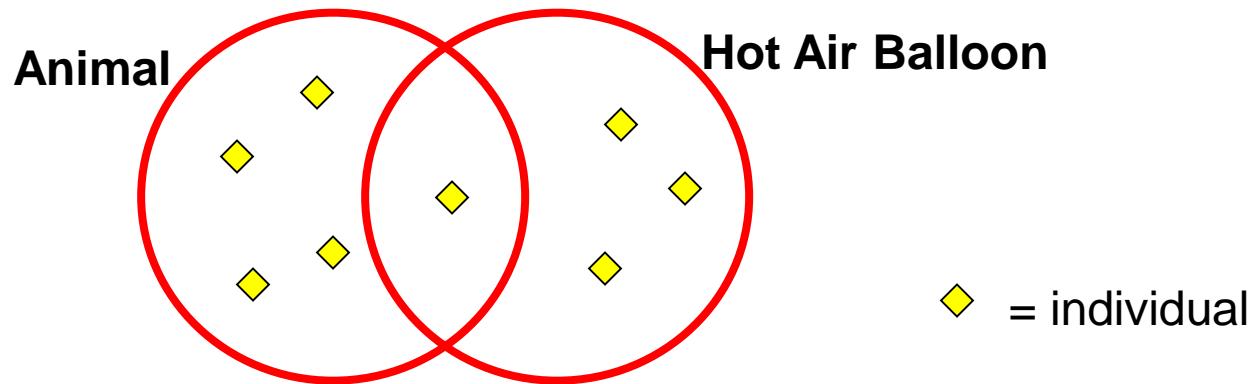
- Включването е основната ос (отношения) в OWL
- Superclass/subclass релация, от тип “is-a”
- **Всички** членове на подклас трябва да бъдат членове на неговите супер-класове



# Къде не можем да сложим класа?

## Disjointness in OWL

Независимо от това къде съществуват в юерархията, OWL предполага, че класовете могат да се препокриват

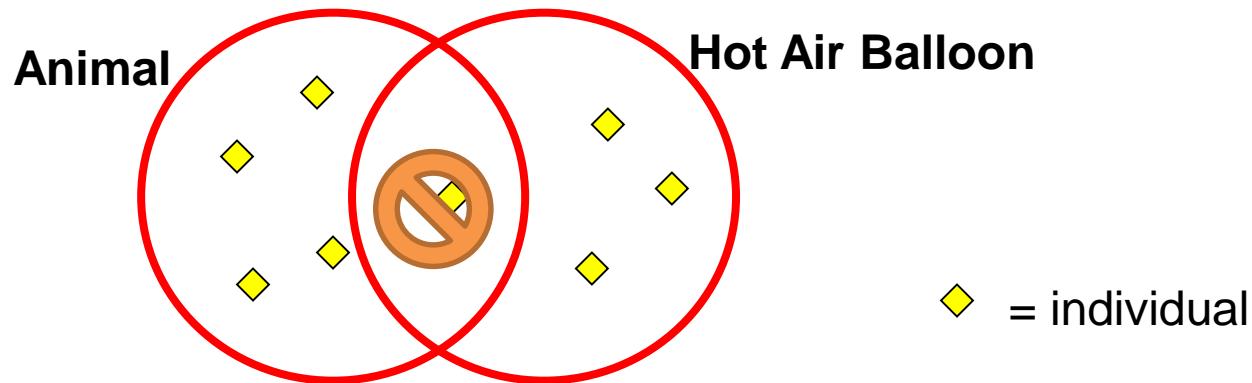


По подразбиране, един индивид може да е **Animal** и **Hot Air Balloon** в едно и също време

# Къде не можем да сложим класа?

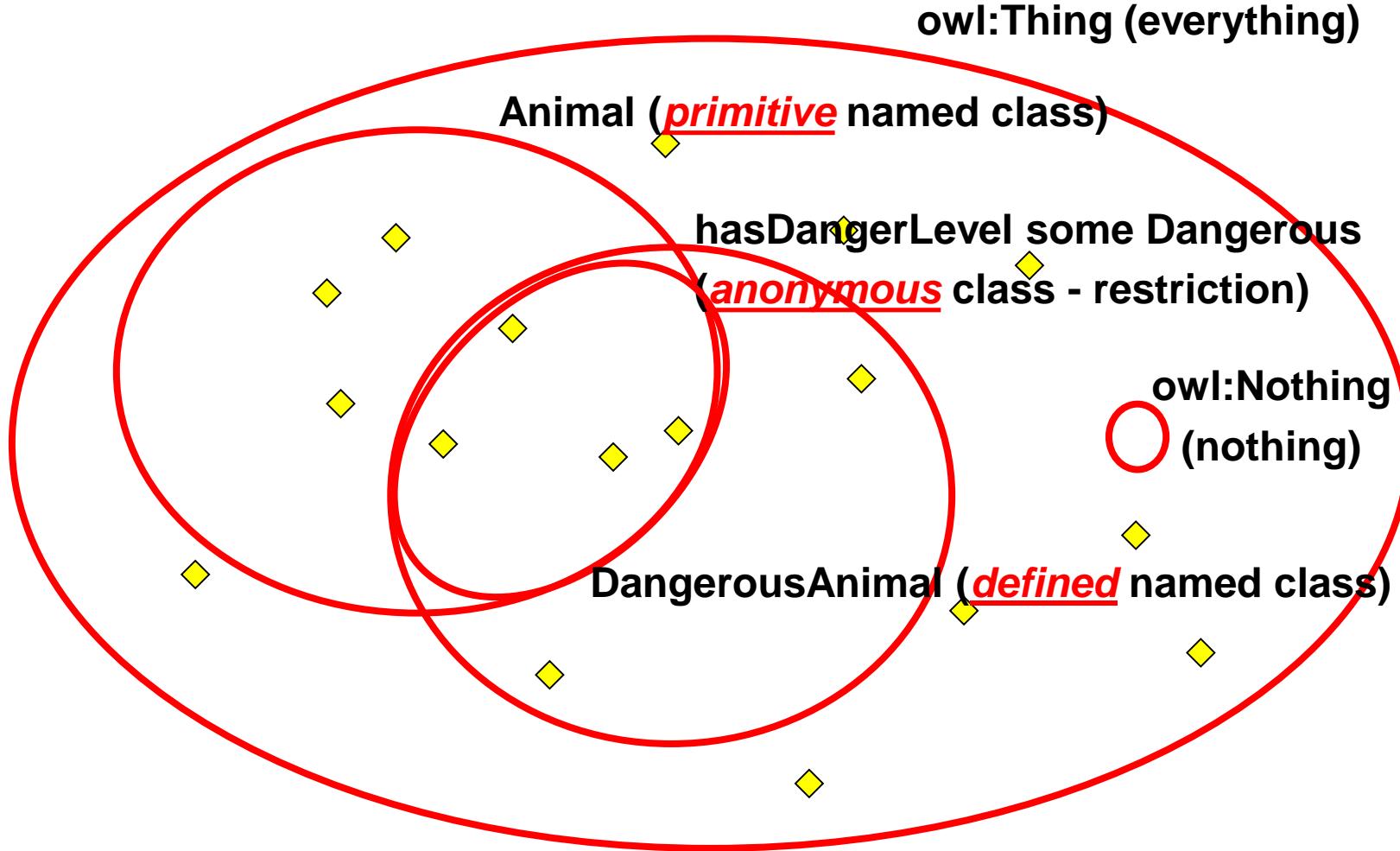
## Disjointness in OWL

Ако два класа са disjoint – тогава...

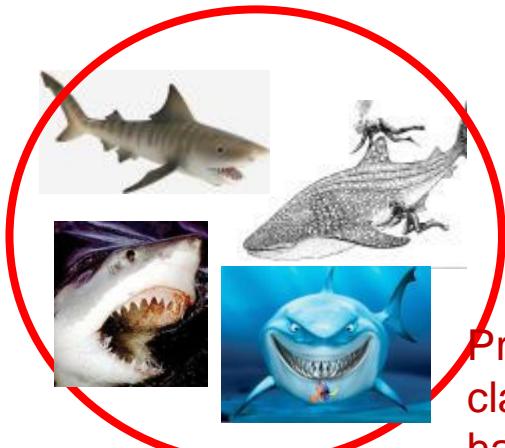


...нешо не може да бъде **Animal** и **Hot Air Balloon**  
едновременно

# Типове класове



# Примитивни спрямо дефинирани класове



Sharks

“Natural Kinds”

Описва задължителните свойства на членовете на класа, напр.

**live underwater:**

“All sharks live underwater, but not everything that lives underwater is a shark”

Defined classes have necessary and sufficient conditions



Blue Things

“Smart Class” – като заявка

Както примитивните, но описват и достатъчните свойства за членство в класа, напр.

**have colour Blue:**

“**All** things that have colour blue are members of this class”

# Анонимни класове

- Създадени от логически изрази
  - Unions and Intersections (Or, And)
  - Complements (Not)
  - Enumerations (specified membership)
  - Restrictions (related to Property use)
- Членовете на анонимен клас са множество от индивиди, които отговарят на неговата логическа дефиниция

# Релации в OWL

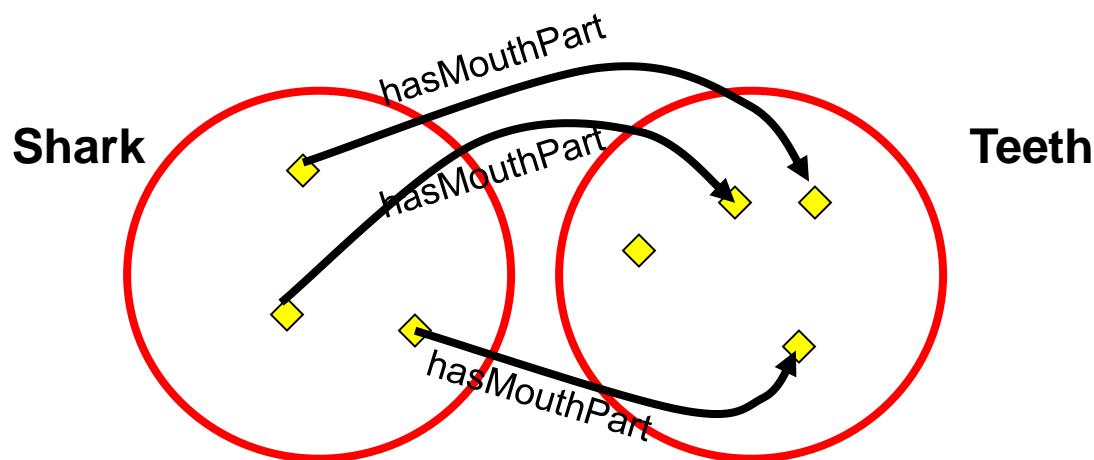
- В OWL-DL, взаимоотношения могат да се формират само между индивидите или между даден индивид и стойност на данни.  
(в OWL-Full, класовете могат да имат релации помежду си, но с тях не може да се разсъждава)
- Релациите се задават чрез свойства (**Properties**)
- Можем да ограничим използването на Properties:
  - глобално – чрез указване на неща за самото Property
  - локално – чрез ограничаване на използването им в даден Class

# Ограничения (Restrictions)

- Ограничения са тип анонимен клас
- Те описват отношенията, които трябва да притежават членовете на този клас

# Пример

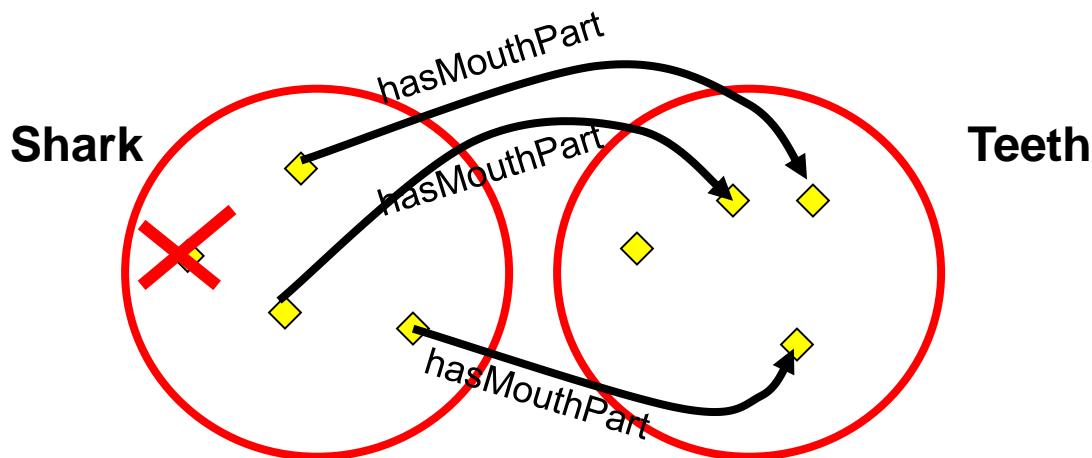
Ограничение за примитивния клас **Shark**:  
задължително член на **Shark** *hasMouthPart* **Teeth**



“Every member of the **Shark** class must have *at least one* mouthpart from the class **Teeth**”

# Пример

Екзистенциално ограничение за примитивния клас **Shark**:  
задължително член на **Shark** *hasMouthPart* **Teeth**



“There can be no member of **Shark**, that does not have *at least one hasMouthPart relationship* with a member of class **Teeth**”

# Типове ограничения

$\exists$	Existential, someValuesFrom	“Some”, “At least one”
$\forall$	Universal, allValuesFrom	“Only”
$\exists$	hasValue	“equals x”
$=$	Cardinality	“Exactly n”
$\leq$	Max Cardinality	“At most n”
$\geq$	Min Cardinality	“At least n”

*"Red wine is a subclass of all things  
that have a red color."*

```
:RedWine
 a owl:Class ;
 rdfs:subClassOf
 [a owl:Restriction ;
 owl:onProperty :color ;
 owl:hasValue
 red^^<http://www.w3.org/2001/XMLSchema#string>
].
```

## :Person

```
a owl:Class ;
rdfs:subClassOf
[a owl:Restriction ;
 owl:onProperty :creatorOf ;
 owl:allValuesFrom :Artefact
].
```

all things created  
by persons are  
artefacts

artefacts are  
things that are  
created by at least  
one person

## :Artefact

```
a owl:Class ;
owl:equivalentClass
[a owl:Restriction ;
 owl:onProperty :createdBy ;
 owl:someValuesFrom :Person
].
```

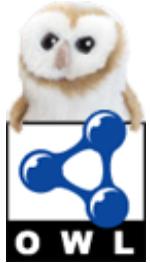


# Типове свойства

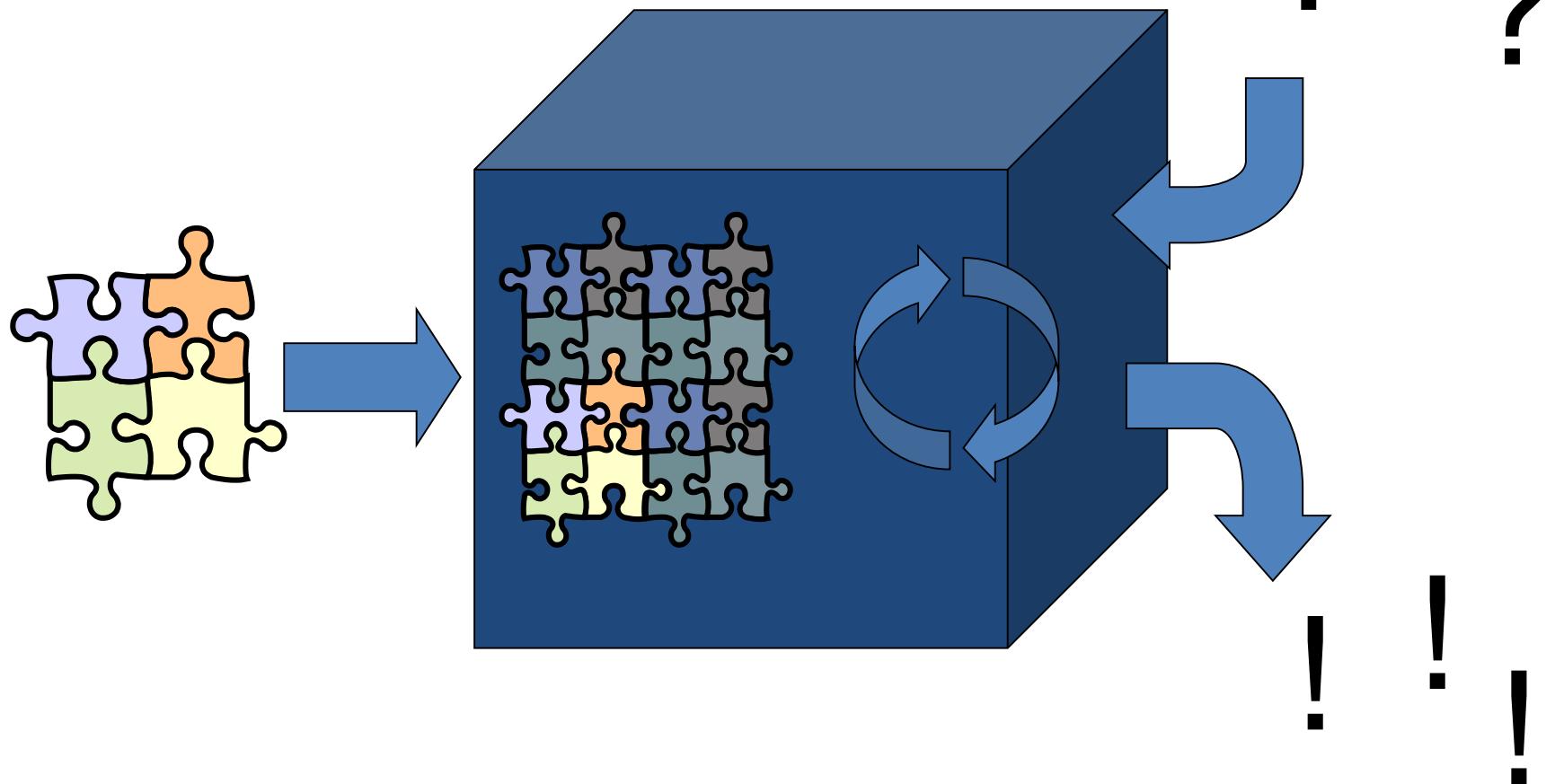
- Different Types:
  - Object Property
    - свързва членове на класа (individuals) с членове
  - Datatype Property
    - свързва членове с данни (int, string, float etc)
  - Annotation Property
    - прикрепва метаданни към classes, individuals или properties

# Характеристики на свойствата

- Домейн (Domain) и обхват (range)
- Йерархия от свойства
- Обратни свойства (Inverse properties)
- Свойствата могат да бъдат:
  - Transitive
  - Functional
  - Inverse Functional
  - Symmetric



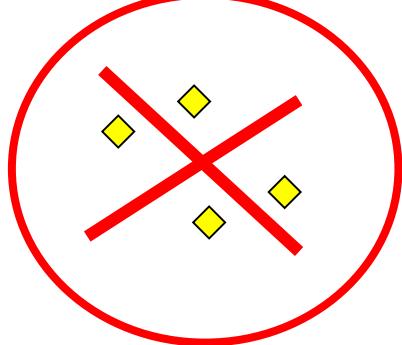
# Разъждения (Reasoning) & изводи (Inference)



# Reasoners: Inference

- Reasoners се използват за заключения относно информация, която не е изрично налична в онтологията
- Наричат се още Класификатори
- Стандартни услуги на reasoner са:
  - Проверка за консистентност - Consistency Checking
  - Проверка за включване - Subsumption Checking (Automatic Subsumption)
  - Проверка за еквивалентност - Equivalence Checking
  - Проверка за инстанциране - Instantiation Checking

# Проверка за консистентност

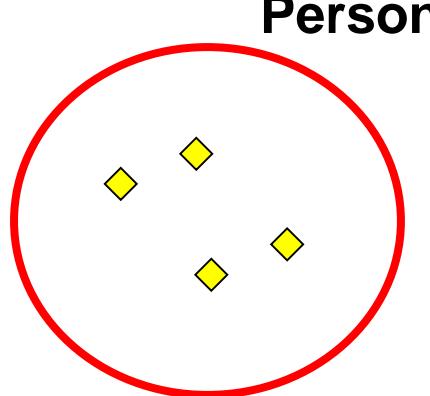


**Shark** (primitive class)

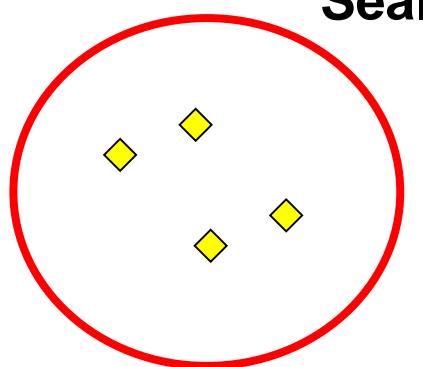
Animal and

eats some (Person and Seal)

**Inconsistent** = cannot contain any individuals



**Person**



**Seal**

XML

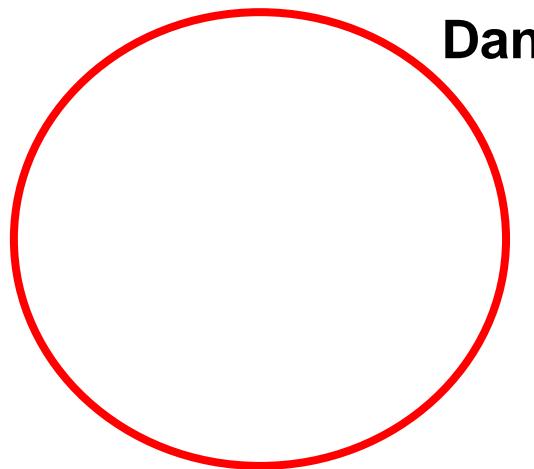
OWL

**Disjoint** (Person, Seal)  
Person **and** Seal = empty  
Cannot have some empty



# Автоматична класификация

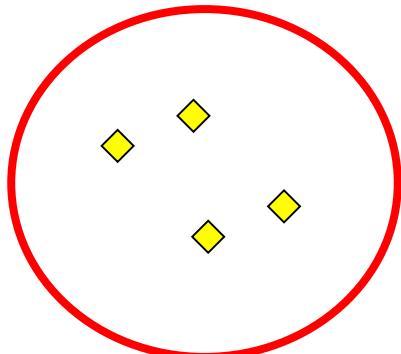
## Тривиален пример



**DangerousAnimal** (defined class)

Animal and

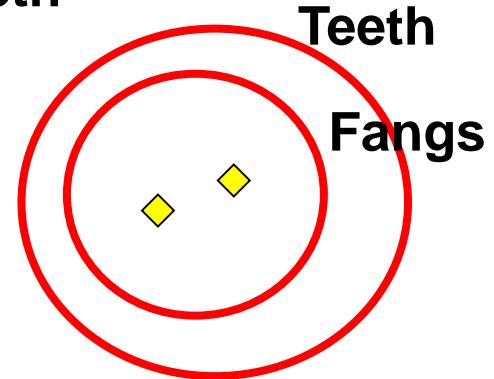
hasMouthPart some **Teeth**



**Shark** (primitive class)

Animal and

hasMouthPart some **Fangs**



# Кога използваме Reasoner

- При разработка - използваме компилатор на онтологии за проверка на значението
- При публикуване - за изводи, направени за потребителски приложения
- По време на изпълнение на приложения - като механизъм за заявки (особено полезно за по-малките онтологии)

# Грешки при моделиране с онтологии

- Основни грешки:
  - Неправилно използване на домейн и обхват
  - Неразбиране на intersections и други конструкции
  - Неразбиране на Open World Assumption
  - Злоупотреба или липса на disjoints
- За допълнителен прочит:
  - OWL Pizzas: Common errors & common patterns  
<http://www.co-ode.org/resources/papers/>

# OWL редактори



<http://www.xml.com/pub/a/2004/07/14/onto.html>



- Представлява среда за моделиране на знания
- Безплатна, софтуер с отворен код
- Разработена от Станфорд (департамент по медицинска информатика)
- Разполага с голяма потребителска общност

<http://protege.stanford.edu>



**SUBCLASS EXPLORER**

For Project: pizza.owl

**Asserted Hierarchy**

- owl:Thing
- DomainConcept
  - Country
  - IceCream
  - Pizza
    - CheeseyPizza
    - InterestingPizza
    - MeatyPizza
    - NamedPizza
      - NonVegetarianPizza
      - RealItalianPizza
      - SpicyPizza
      - SpicyPizzaEquivalent
      - VegetarianPizza
      - VegetarianPizzaEquivalent1
      - VegetarianPizzaEquivalent2
  - PizzaBase
  - PizzaTopping
    - CheeseTopping
    - FishTopping
      - AnchoviesTopping
      - MixedSeafoodTopping
      - PrawnsTopping
    - FruitTopping
    - HerbSpiceTopping
    - MeatTopping

**CLASS EDITOR**

For Class: RealItalianPizza (instance of owl:Class)  Inferred View

**Annotations**

Property	Value	Lang
rdfs:comment	This defined class has conditions that are part of the definition: ie any Pizza that has the country of origin, Italy is a RealItalianPizza. It also has conditions that merely describe the members - that all RealItalianPizzas must only have ThinAndCrispy bases.	en
rdfs:label	PizzaitalianaReal	pt

**Asserted Conditions**

- Pizza NECESSARY & SUFFICIENT
- hasCountryOfOrigin has Italy NECESSARY
- hasBase only ThinAndCrispyBase INHERITED [from Pizza]
- hasBase some PizzaBase

**Disjoints**

Logic View Properties View

# Програмиране с OWL

- Protégé OWL API
- Wonderweb OWL API
- Jena
- OWL API



# Reasoners

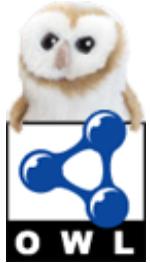
- FaCT++
- Pellet
- RACER

# ПРИМЕРНИ ОНТОЛОГИИ

- ОВО – Open BioMedical Ontologies
- The Gene Ontology
- Bio tutorial and Pizza tutorial examples on the CO-ODE site
- Libraries are commonly published on OWL editor websites
- Search using Google or Swoogle

# Примерни приложения

- PizzaFinder (dummy query application)
- COHSE – dynamic hyperlinking using ontologies
- Protein Phosphatase Modelling – ask Robert Stevens
- OWL Validator
- GONG (Gene Ontology Next Generation)
- AKT  
<http://www.aktors.org/>
- The Semantic Web Challenge  
<http://challenge.semanticweb.org/>



# Вместо заключение

- Примери и наръчник



pizza tutorial

<http://owl.cs.manchester.ac.uk/research/co-ode/>