XML - eXtensible Markup Language

- Tópicos:
  - XML para transferência de dados
  - Estrutura hierárquica do XML
  - DTDs e XML Schema
  - Consultas de documentos XML:
    - Xpath
    - XQuery
  - Transformação de documentos XML: XSLT
  - Mapeamento entre documentos XML e Bases de Dados relacionais
- Bibliografia:
  - Capítulo 23 do livro recomendado

O que é o XML?

- XML significa eXtensible Markup Language
- XML é uma linguagem de marcação (markup language) tal como o HTML.
- XML foi desenhado para descrever dados
- Marcadores (tags) em XML não estão predefinidos. Têm que ser definidos pelo programador
- O XML usa um Document Type Definition (DTD) ou um XML Schema para descrever os tipos de dados
- XML com DTD ou XML Schema está desenhado para ser auto-descritivo
- XML é uma recomendação do W3C.
- Vem no seguimento do SGML (Standard Generalized Markup Language), mas é mais simples de usar

XML vs. HTML

- O HTML foi desenhado para mostrar os dados, focando **no seu aspecto**
- O XML foi desenhado para descrever dados, focando **no que eles são**
- No HTML, a sintaxe é fixa (o conjunto de tags está predefinido) e os marcadores definem simultaneamente os dados e a sua visualização.
- No XML, os dados, a sintaxe em que são descritos, e a forma como são visualizados são definidos separadamente.
- Ao contrário do HTML (que também deriva do SGML), o XML é **extensível**
- No XML, o utilizador pode adicionar novos tipos de *tags* e definir depois, separadamente, como lidar com as novas *tags* (nomeadamente para lidar com a apresentação).

Exemplo de XML

```
1.      <?xml version="1.0" encoding="ISO-8859-1"?>
2.        <recado>
3.              <de>Maria</de>
4.              <para>Manuel</para>
5.              <titulo>Lembrete</titulo>
6.              <corpo>Jantar hoje!</corpo>
7.        </recado>
```

- 1. contém o cabeçalho
- 2. contém o elemento raiz do documento (significa que este documento é um recado)
- 3. a 6. contêm 4 elementos filhos da raiz
- 7. indica o fim da raiz
- **E o que é que isto tem a ver com Bases de Dados?!?**

XML para Bases de Dados

- A possibilidade de especificar novas tags, e de criar uma estrutura imbricada de tags, torna o XML uma ótima linguagem para troca de dados.
- As tags tornam os dados, de certa forma, auto-documentados
  - E.g.

```
<banco>
    <conta>
        <num-conta> A-101    </num-conta>
        <balcao>    Lx       </balcao>
        <saldo>     500      </saldo>
    </conta>
    <depositante>
        <num-conta>    A-101  </num-conta>
        <nome-cliente> Paulo  </nome-cliente>
    </depositante>
</banco>
```

XML: Motivação

- A troca de informação é uma tarefa crítica e com importância crescente
  - Exemplos:
    - Banca: transferência de fundos
    - Aplicações B2B (processamento de compras entre empresas)
    - Comunicação de dados a autoridades (e.e. SAFT)
    - Dados científicos
      - Química: ChemML, …
      - Genética: BSML (Bio-Sequence Markup Language), …
  - O fluxo de papel entre organizações tende a ser substituído por fluxo (eletrónico) de informação.
- Cada área de aplicação tem os seus próprios standards para representar a informação
- O XML tornou-se a base para a geração de formatos de troca de dados

XML: Motivação (Cont.)

- Cada standard baseado em XML define quais os elementos válidos a transmitir através de:
  - Linguagens de especificação de tipos em XML
    - DTD (Document Type Definition)
    - XML Schema
  - Descrição textual (ou formal via RDF, OWL) da semântica
- O XML permite que sejam definidas novas tags:
  - Isto é restringido pelos DTDs
- Existe atualmente uma grande variedade de produtos para fazer parsing, transformar (em HTML) e pesquisar dados em documentos XML

Estrutura dos dados em XML

- **Tag**: label para uma secção de dados
- **Elemento**: seção de dados começada por *<tagname>* e terminada pelo correspondente *</tagname>*
- Os inícios e fins de secções têm que estar bem emparelhados (como estrutura de parêntesis)
  - Exemplo correto
    - <conta> … <saldo> …. </saldo> </conta>
  - Exemplo incorrecto
    - <conta> … <saldo> …. </conta> </saldo>
  - Formalmente: A cada tag de início deve corresponder exatamente uma tag de fim, que está no mesmo contexto (onde um contexto é tudo o que está entre uma tag de início e a correspondente tag de fim)
- Um documento tem um único elemento de nível mais alto (raiz)

Exemplo de elementos imbricados

```
<banco-1>
    <cliente>
      <nome-cliente> Luís              </nome-cliente>
      <rua-cliente>   5 de Outubro </rua-cliente>
      <local-cliente> Lisboa          </ local-cliente>
      <conta>
                <num-conta> A-102    </num-conta>
                <agencia>   Caparica </agencia>
                <saldo>     400      </saldo>
      </conta>
      <conta>
                <num-conta> A-103    </num-conta>
                <agencia>   Lisboa </agencia>
                <saldo>     600      </saldo>
      </conta>
      <conta>
                …
      </conta>
    </cliente>
      …
</banco-1>
```

Motivação para dados imbricados

- A imbricação de dados não é suportada em bases de dados relacionais
  - Pode causar dados redundantes (e.g. informação de contas com vários clientes)
  - A normalização substitui estruturas imbricadas por existência de várias tabelas com restrições de chaves estrangeiras
  - As bases de dados objeto-relacional suportam dados imbricados
- Mas imbricação de dados é importante na transferência de dados
  - A aplicação externa pode não ter acesso à tabela referenciada pelas chaves estrangeiras!

Estrutura dos dados em XML(Cont.)

- O XML permite misturar texto com sub-elementos
  - Exemplo:

```
<conta>
        Esta conta já quase não é usada.
        <num-conta> A-101   </num-conta>
        <balcao>    Lx      </balcao>
        <saldo>     500     </saldo>
</conta>
```

  - Isto é útil para marcação de documentos, mas **desencorajado** em representação de dados…

Atributos

- Os elementos podem ter **atributos**

```
<conta tipo-conta = “ordem” >
    <num-conta> A-101    </num-conta>
    <balcao>    Lx       </balcao>
    <saldo>     500      </saldo>
</conta>
```

- Os atributos são especificados dentro da tag de início, por pares da forma *nome=valor*
- Um elemento pode ter vários atributos, mas cada nome de atributo só pode ocorrer uma vez em cada tag
  - <conta tipo-conta = “ordem” limite-negativo=“500”>

Atributos versus sub-elementos

- Distinção entre subelementos e atributos
  - No contexto de um documento, os atributos são partes da marcação, e subelementos são partes dos dados do documento propriamente dito.
  - No contexto de representação de dados, a diferença não é tão clara
    - A mesma informação pode ser representada das duas formas
      - <conta num-conta = “A-101”> …. </conta>
      - <conta>
        <num-conta> A-101</num-conta> …
        </conta>
  - Sugestão: usar atributos para identificadores (ou chaves) de elementos, e usar subelementos para outros atributos

Mais sobre sintaxe XML

- A sintaxe de elementos sem sub-elementos nem texto, pode ser abreviada terminando a tag de início /> e evitando a tag de fim
  - <conta num-conta="A-101" agencia="Lx" saldo="200" />
- Para guardar strings que tenham a forma de tags (sem as querer interpretar como tal) usar
  - <![CDATA[<conta> … </conta>]]>
    - Aqui, <conta> e </conta> são simplesmente strings de dados

Namespaces

- Os dados em XML podem servir para ser transferidos entre empresas
- A mesma tag pode ter significados diferentes em empresas diferentes, o que pode causar confusão na troca de documentos
- Solução: usar *nome-único*:*elemento-único*
- Para evitar a geração de nomes únicos grandes por todo o documento, podem usar-se os Namespaces do XML

```
<banco xmlns:FB='http://www.FirstBank.com'>
    …
     <FB:agencia>
         <FB:nome-agencia> R. Ouro</FB:nome-agencia>
         <FB:local-agencia> Lx</FB:local-agencia>
     </FB:agencia>
    …
</banco>
```

Esquemas de Documentos XML

- Em bases de dados, os esquemas definem que informação pode ser armazenada, com que estrutura, e quais os tipos dos vários valores
- Os documentos XML não têm obrigatoriamente um esquema associado
- Mas, quando para uso em transferência de dados, é importante associar esquemas a documentos XML
  - Caso contrário, como poderia um site interpretar de forma automática dados recebidos de outro site?
- Dois mecanismos para especificar esquemas de XML:
  - **Document Type Definition (DTD)**
    - Ainda bastante usado, mais simples
  - **XML Schema**
    - Mais potente, e com uso cada vez maior

Document Type Definition (DTD)

- O tipo (esquema) de um documento XML pode ser especificado por um DTD
- Um DTD define a estrutura de dados permitida em XML
  - Que elementos podem aparecer
  - Que atributos pode ter (ou tem que ter) cada um dos elementos
  - Que subelementos podem (ou têm que) ocorrer dentro de cada elemento, e quantas vezes
- Os DTD não servem para restringir tipos de dados
  - Todos os valores são representados como strings
- Sintaxe de um DTD
  - <!ELEMENT elemento (especificação-subelementos) >
  - <!ATTLIST elemento (atributos) >

Especificação de elementos em DTDs

- Um subelemento pode ser especificado como:
  - nomes de elementos, ou
  - #PCDATA (parsed character data), i.e., strings
  - EMPTY (nenhum subelemento) ou ANY (tudo pode aparecer como subelemento)
- Exemplo

  <!ELEMENT depositante (nome-cliente, num-conta)>
  <!ELEMENT nome-cliente (#PCDATA)>
  <!ELEMENT num-conta (#PCDATA)>
- A especificação de subelementos pode conter expressões regulares:

  <!ELEMENT banco ((conta | cliente | depositante)+)>
  - Notação:
    - "|" - alternativas
    - "+" - 1 ou mais ocorrências
    - "*" - 0 ou mais ocorrências
    - "?" - 0 ou 1 ocorrência

Exemplo de DTD

```
<!DOCTYPE banco [
    <!ELEMENT banco ( ( conta | cliente | depositante)+)>
    <!ELEMENT conta (num-conta, agencia, saldo)>
    <!ELEMENT cliente (nome-cliente, rua-cliente, local-cliente)>
    <!ELEMENT depositante (nome-cliente, num-conta)>
    <!ELEMENT num-conta (#PCDATA)>
    <!ELEMENT agencia (#PCDATA)>
    <!ELEMENT saldo (#PCDATA)>
    <!ELEMENT nome-cliente (#PCDATA)>
    <!ELEMENT rua-cliente (#PCDATA)>
    <!ELEMENT local-cliente (#PCDATA)>
]>
```

Especificação de atributos em DTDs

- Para cada atributo, especifica-se
  - Nome
  - Tipo do atributo
    - CDATA
    - ID (identificador) ou IDREF (referência para ID) ou IDREFS (múltiplos IDREFs)
  - Restrições adicionais
    - obrigatório (#REQUIRED)
    - Com valor por defeito (valor),
    - Nenhum dos dois (#IMPLIED)
- Exemplos
  - <!ATTLIST conta tipo-conta CDATA “ordem”>
  - <!ATTLIST cliente
    id-cliente ID #REQUIRED
    contas IDREFS #REQUIRED >

IDs e IDREFs

- Um elemento tem no máximo um atributo do tipo ID
- O valor dum atributo de tipo ID tem que ser único em todo o documento XML
  - O atributo com tipo ID é um identificador do objeto (elemento)
- Um atributo do tipo IDREF tem que conter um valor que exista num atributo de tipo ID, no mesmo documento XML
- Um atributo do tipo IDREFS contém um conjunto de (0 ou mais) valores, onde cada um desses valores tem que existir como ID de algum elemento do mesmo documento

Exemplo de DTD com atributos

```
<!DOCTYPE banco-2[
    <!ELEMENT conta (agencia, saldo)>
    <!ATTLIST conta
            num-conta       ID          #REQUIRED
            clientes        IDREFS #REQUIRED>
     <!ELEMENT cliente (nome-cliente, rua-cliente,
                                        local-cliente)>
     <!ATTLIST cliente
            id-cliente      ID          #REQUIRED
            contas          IDREFS #REQUIRED>
     ... ...
]>
```

Documento XML com atributos ID e IDREF

```
<?xml version = "1.0" standalone = "no"?>
<!DOCTYPE banco-2 SYSTEM "http://centria.fct.unl.pt/~jja/banco2.dtd">
<banco-2>
    <conta num-conta="A-401" clientes="C100 C102">
        <agencia> Caparica </agencia>
        <saldo>500 </saldo>
    </conta>
    <cliente id-cliente="C100" contas="A-401">
        <nome-cliente> Luís </nome-cliente>
        <rua-cliente> R. República </rua-cliente>
        <local-cliente> Lx </local-cliente>
    </cliente>
    <cliente id-cliente="C102" contas="A-401 A-402">
        <nome-cliente> Maria </nome-cliente>
        <rua-cliente> R. 5 de Outubro </rua-cliente>
        <local-cliente> Porto </local-cliente>
    </cliente>
</banco-2>
```

Erro!

Limitações de DTDs

- Não é possível especificar tipos de elementos e atributos
  - Todos os tipos são strings. Não há inteiros, reais, etc.
- É difícil especificar conjuntos (não ordenados) de subelementos
  - Em bases de dados a ordem é (normalmente) irrelevante
  - (A | B)* permite especificar um conjunto não ordenado, mas não permite garantir que cada um de A e B ocorre pelo menos uma vez
- Os IDs e IDREFs são não tipados
  - O atributos *cliente* de uma conta pode ter referências a outras contas (o que não faz sentido!)
    - *clientes* deveria apenas conter referências para identificadores de clientes

XML Schema

- XML Schema é uma outra linguagem (mais sofisticada) que endereça os problemas dos DTDs.
- Tem mecanismos para
  - Definir tipos de dados para elementos e atributos
    - E.g. inteiros, string, etc
    - Valores min/max
  - Tipos de dados para elementos e atributos *definidos pelo utilizador*
  - Ao contrário do que acontece com DTDs, a sintaxe do XML Schema é a sintaxe do XML, tornando-os extensíveis a futuras adições
  - Integrado com namespaces
  - Outras características
    - Listas, restrições de chave primária e estrangeira, herança, etc...
- Desvantagem: mais complicado do que os DTDs

Versão XML Schema do exemplo anterior

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="banco" type="TipoBanco"/>
<xsd:element name="conta">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="num-conta" type="xsd:string"/>
            <xsd:element name="agencia"   type="xsd:string"/>
            <xsd:element name="saldo"     type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
….. Definições para cliente e depositante….
<xsd:complexType name="TipoBanco">
    <xsd:sequence>
        <xsd:element ref="conta"     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="cliente"   minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="depositante" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Versão XML Schema do exemplo anterior

- A escolha de “xsd:” é nossa – qualquer outro prefixo do espaço de nomes (namespace) podia ter sido escolhido.
- O elemento “banco” tem o tipo “TipoBanco” que é definido separadamente.
  - xsd:complexType é usado a seguir para criar o tipo de dados complexo, com nome, “TipoBanco”
- O elemento “conta” tem o seu tipo definido in-line

Outras Características de XML Schema

- Atributos especificados por *tag* xsd:attribute:
  - <xsd:attribute name = “num-conta”/>
  - Adicionando o atributo use = “required” significa que o valor tem que ser especificado
- Restrições de chaves: “números de conta formam a chave de elementos *conta* debaixo do elemento raiz *banco*”:

```
<xsd:key name = “accountKey”>
  <xsd:selector xpath = “/banco/conta”/>
  <xsd:field xpath = “num-conta”/>
</xsd:key>
```

- Restrições de chaves estrangeira de depositante para conta:

```
<xsd:keyref name = “depositorAccountKey” refer=“accountKey”>
  <xsd:selector xpath = “/banco/depositante”/>
  <xsd:field xpath = “num-conta”/>
</xsd:keyref>
```

Transformação e Consulta de dados XML

- Tradução de informação de um esquema XML para outro
- Relacionado com pesquisa de informação em documentos XML (e tratado pelas mesmas ferramentas)
- Linguagens para transformação/pesquisa em documentos XML
  - XPath
    - Linguagem simples, que consiste em path expressions
  - XQuery
    - Linguagem mais complexa de pesquisa de informação em documentos XML (com algumas semelhanças com SQL)
  - XSLT
    - Linguagem desenhada para tradução de documentos XML para XML e XML para HTML
- Existem outras linguagens:
  - XML-QL, Quilt, XQL, Xcerpt, …

Modelo em árvore de documentos XML

- A maior parte das linguagens de transformação e pesquisa são baseadas no **modelo em árvore** de documentos XML
- Um documento XML pode ser visto como uma árvore, onde os nós correspondem a elementos e a atributos
  - Os nós-elementos têm filhos, que podem ser atributos ou subelementos
  - Texto num elemento é visto como um nó de tipo texto (sem filhos)
  - Os filhos dum nó são ordenados de acordo com a ordem em que aparecem no documento XML
  - Os nós de elementos e atributos têm sempre um único pai.
  - A raiz da árvore contém um único filho, correspondendo ao elemento raiz do documento

Documento XML de exemplo

```
<?xml version = “1.0” standalone = “no”?>
<!DOCTYPE banco-2 SYSTEM “http://centria.fct.unl.pt/~jleite/banco2.dtd”>
<banco-2>
    <conta num-conta=“A-401” clientes=“C100 C102”>
        <agencia> Caparica </agencia>
        <saldo>500 </saldo>
    </conta>
    <cliente id-cliente=“C100” contas=“A-401”>
        <nome-cliente> Luís </nome-cliente>
        <rua-cliente> R. República </rua-cliente>
        <local-cliente> Lx </local-cliente>
    </cliente>
    <cliente id-cliente=“C102” contas=“A-401”>
        <nome-cliente> Maria </nome-cliente>
        <rua-cliente> R. 5 de Outubro </rua-cliente>
        <local-cliente> Porto </local-cliente>
    </cliente>
</banco-2>
```

XPath

- O XPath serve para selecionar partes de documento usando para tal **path expressions**
- Uma *path expression* é uma sequência de passos, separados por "/"
  - Semelhante a nome de ficheiros numa hierarquia de diretorias
- Resultado duma *path expression*:
  - Conjunto de nós, e correspondentes subelementos, e atributos quando for caso disso, que correspondem ao caminho (*path)* dado

Exemplo de Xpath

```
<banco-2>
        <conta num-conta="A-401" clientes="C100 C102">
                <agencia> Caparica </agencia>
                <saldo>500 </saldo>
        </conta>
        <cliente id-cliente="C100" contas="A-401">
                <nome-cliente> Luís </nome-cliente>
                <rua-cliente> R. República </rua-cliente>
                <local-cliente> Lx </local-cliente>
        </cliente>
        <cliente id-cliente="C102" contas="A-401">
                <nome-cliente> Maria </nome-cliente>
                <rua-cliente> R. 5 de Outubro </rua-cliente>
                <local-cliente> Porto </local-cliente>
        </cliente>
</banco-2>
```

- A path expression /banco-2/cliente/nome-cliente devolve (os nome dos clientes):

  <nome-cliente> Luís </nome-cliente>

  <nome-cliente> Maria </nome-cliente>

Exemplo de Xpath

```
<banco-2>
        <conta num-conta="A-401" clientes="C100 C102">
                <agencia> Caparica </agencia>
                <saldo>500 </saldo>
        </conta>
        <cliente id-cliente="C100" contas="A-401">
                <nome-cliente> Luís </nome-cliente>
                <rua-cliente> R. República </rua-cliente>
                <local-cliente> Lx </local-cliente>
        </cliente>
        <cliente id-cliente="C102" contas="A-401">
                <nome-cliente> Maria </nome-cliente>
                <rua-cliente> R. 5 de Outubro </rua-cliente>
                <local-cliente> Porto </local-cliente>
        </cliente>
</banco-2>
```

- A path expression /banco-2/cliente/nome-cliente/text( ) devolve (o texto dos nomes dos clientes):

  Luís

  Maria

Exemplo de Xpath

- A path expression /banco-2/cliente devolve (os clientes):

```
<cliente id-cliente="C100" contas="A-401">
        <nome-cliente> Luís </nome-cliente>
        <rua-cliente> R. República </rua-cliente>
        <local-cliente> Lx </local-cliente>
</cliente>
<cliente id-cliente="C102" contas="A-401">
        <nome-cliente> Maria </nome-cliente>
        <rua-cliente> R. 5 de Outubro </rua-cliente>
        <local-cliente> Porto </local-cliente>
</cliente>
```

XPath (Cont.)

- O "/" inicial denota a raiz do documento
- As *path expressions* são avaliadas da esquerda para a direita
  - Cada passo é aplicado ao conjunto de instâncias produzidas pelo passo anterior
- Podem-se usar predicados de seleção (entre [ ]) em qualquer dos passos do path.
- E.g. /banco-2/conta[saldo > 400]
  - Devolve os elementos de todas as contas com saldo superior a 400
  - /banco-2/conta[saldo] devolve os elementos de todas as contas que contêm um subelemento saldo
- Pode-se aceder aos atributos, usando "@"
  - E.g. /banco-2/conta[saldo > 400]/@num-conta
    - Devolve os números das contas cujo saldo é maior que 400
  - Os atributos IDREF não são desreferenciados automaticamente (mais sobre este assunto mais à frente)

Funções em XPath

- O XPath fornece várias funções:
  - E.g. função count() aplicada a uma expressão, conta o número de elementos do conjunto gerado pelo path
    - E.g. /banco-2/conta[count(cliente) > 2]
      - Devolve conjunto de nós conta com pelo menos 3 subelementos cliente (conjunto vazio para o exemplo dado)
  - Também há funções para testar a posição de um nó relativamente aos seus irmãos, somar valores, operadores sobre strings, inteiros, etc. Exemplos:
    - sum(), contains(st1,st2), concat(st1,st2,st), position(), last(), round(num), ...
- Nos predicados podem-se usar os conectivos Booleanos and e or e a função not()

Funções em XPath (cont.)

- As IDREFs podem-se desreferenciar usando para tal a função id()
  - id() pode também ser aplicado a conjuntos de referências (IDREFS e strings de IDREFs separadas por espaços)
  - E.g. /banco-2/conta/id(@clientes)
    - Devolve todos os clientes referenciados por contas, no seu atributo clientes.
  - E.g. /banco-2/conta[@num-conta="A-401"]/id(@clientes)
    - Devolve todos os clientes da conta A-401
  - Semelhante às *path expression* das bases de dados objeto-relacional

Mais características do XPath

- Operador “|” para uniões
  - E.g. /banco-2/conta/id(@clientes) | /banco-2/emprestimo/id(@clientes)
    - Devolve os clientes com contas ou empréstimos
    - NOTA: O “|” não pode estar imbricado noutros operadores.
- Operador “//” para saltar vários níveis de uma árvore de uma só vez
  - E.g. /banco-2//nome
    - Devolve qualquer subelemento com nome nome que esteja dentro do elemento /banco-2, independentemente do número de níveis entre os dois.

Mais características do XPath

- Um passo no caminho (path) pode ir para o pai, irmãos, antecessores, descendentes (e não apenas para os filhos, como vimos até aqui). E.g.:
  - O "//", acima, denota todos os descendentes
  - ".." denota o pai.
  - "." denota o próprio nó.
- doc(nome) retorna a raiz do documento com nome "*nome*"
  - *E.g.* se o exemplo do banco estivesse contido num ficheiro banco.xml, então, a *path expression* doc('banco.xml')/banco-1/conta devolveria todos os elementos conta.
  - Permite a especificação de *path expressions* sobre outros documentos.