

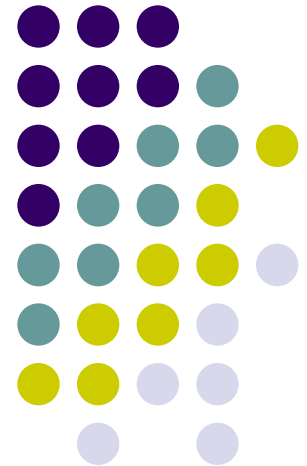
Advanced Aspects and New Trends in XML (and Related) Technologies

RNDr. Irena Holubová, Ph.D.

holubova@ksi.mff.cuni.cz

Lecture 1. Modelling and generating of XML data, XML benchmarking

<http://www.ksi.mff.cuni.cz/~svoboda/courses/171-NPRG039/>

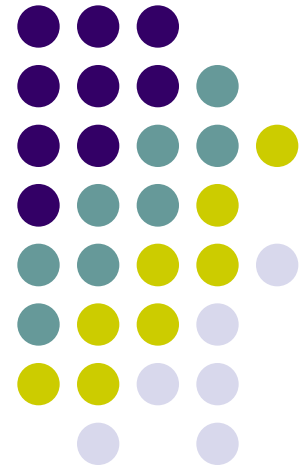


Problem: How to Design/Create XML Data?



- We need: XML schema + XML data
 - Design of structure + creation of instances
- Simple structures
 - XML data editor (highlighting, hints, well-formedness/validity checking, ...)
- Complex structures
 - Modelling of XML data
- Instances
 - Created by applications themselves
 - XML data generators – testing purposes
 - XML benchmarking

Design of XML Data



Structure of XML Data

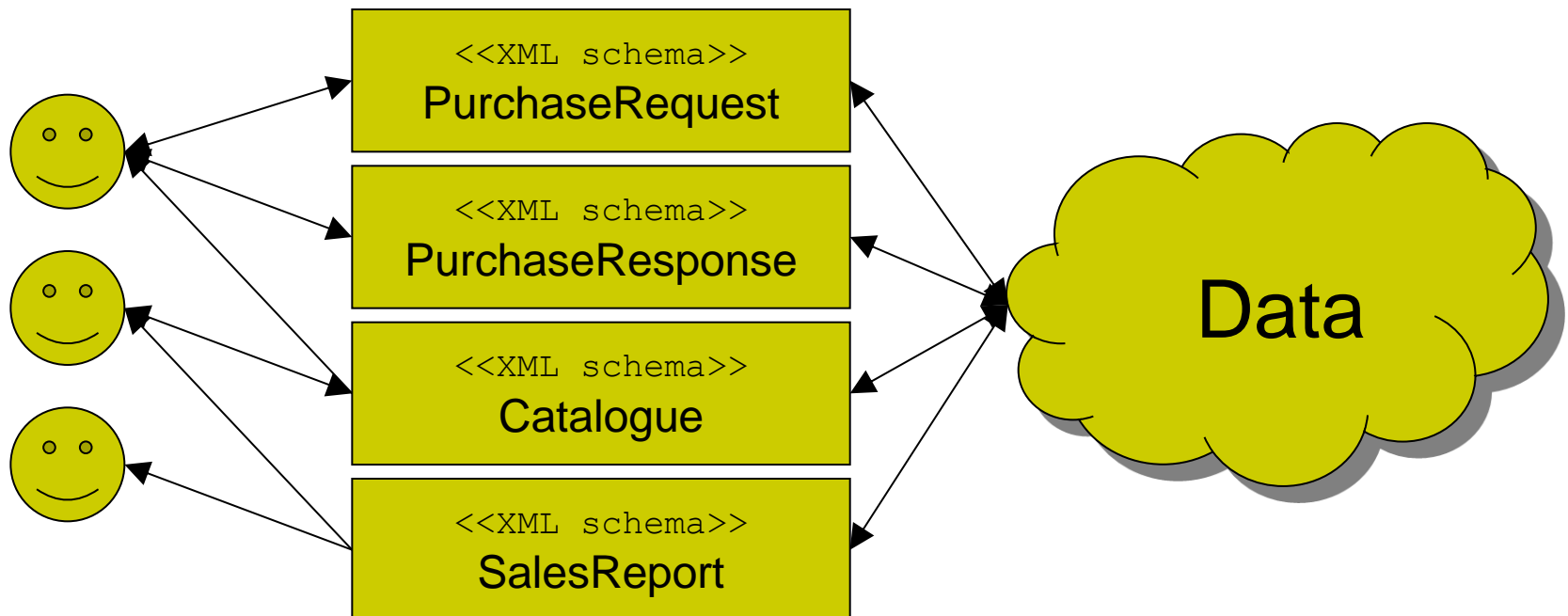


- XML schemas
 - DTD, XML Schema, RELAX NG, Schematron
 - Specification of structure of XML documents
 - What elements and attributes can be used
- Problems
 - Complex to learn
 - Too technical for non-technical people
 - Dealing with technical details (special syntax, well-formedness, ...)
 - Absence of semantics
 - We describe just structure + integrity constraints

XML Schema Languages



- Real world
 - Different groups of users → various types of XML documents ("XML views") in system



XML View of Data



- One real-world concept (e.g., customer or product) is represented in various XML formats in different ways
 - Description distributed across various XML schemas
 - Redundancy & incompleteness
- Lack of complete & non-redundant description

Example: Standard XML Schema Formats



- HL7 (Health Level Seven)
 - Exchanging medical records
- OASIS UBL (Universal Business Language)
 - Exchanging business data
- ISO20022
 - Exchanging financial data
- opentravel.org
 - Data in travel business
- Google AdWords Web Services
 - Advertising via Google

- Common characteristics:
 - Hundreds of XML schemas
 - Related, overlapping
 - Changed regularly



Visualization of XML Schemas

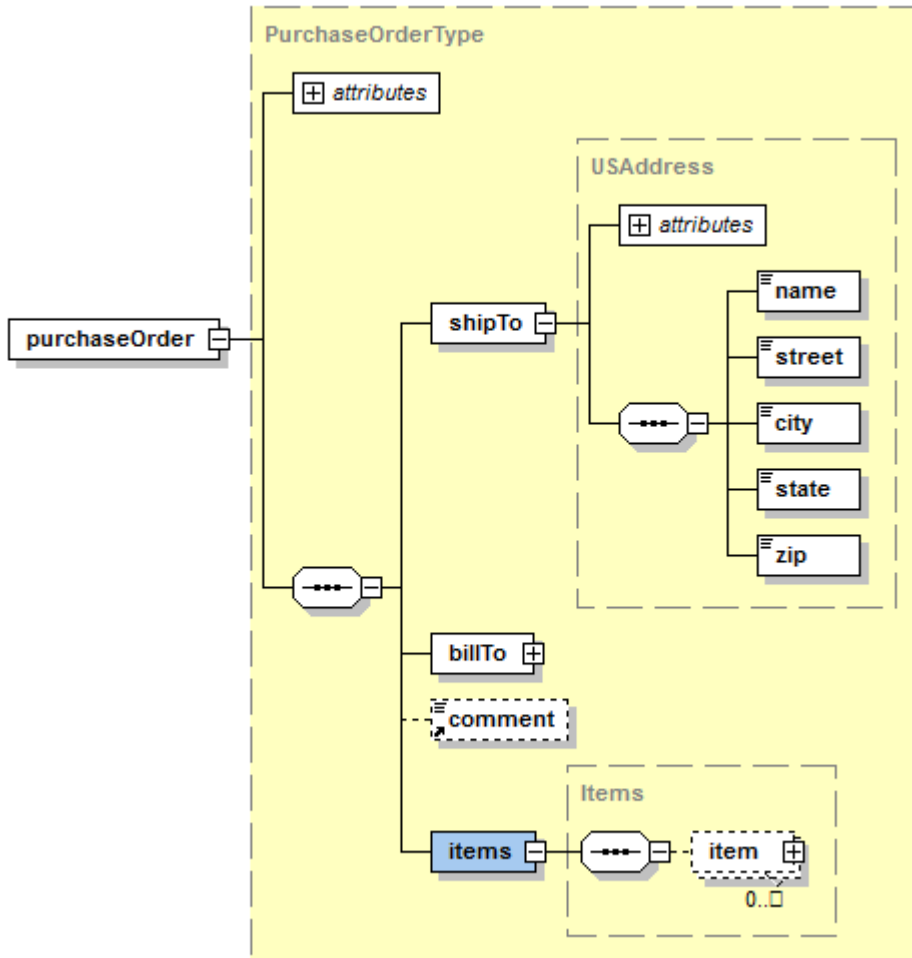


- **Altova XML Spy**
- Stylus Studio
- Oxygen XML Editor
- ...



- Visualization of each construct of an XML schema language
 - Usually XML Schema

Visualization of XML Schemas



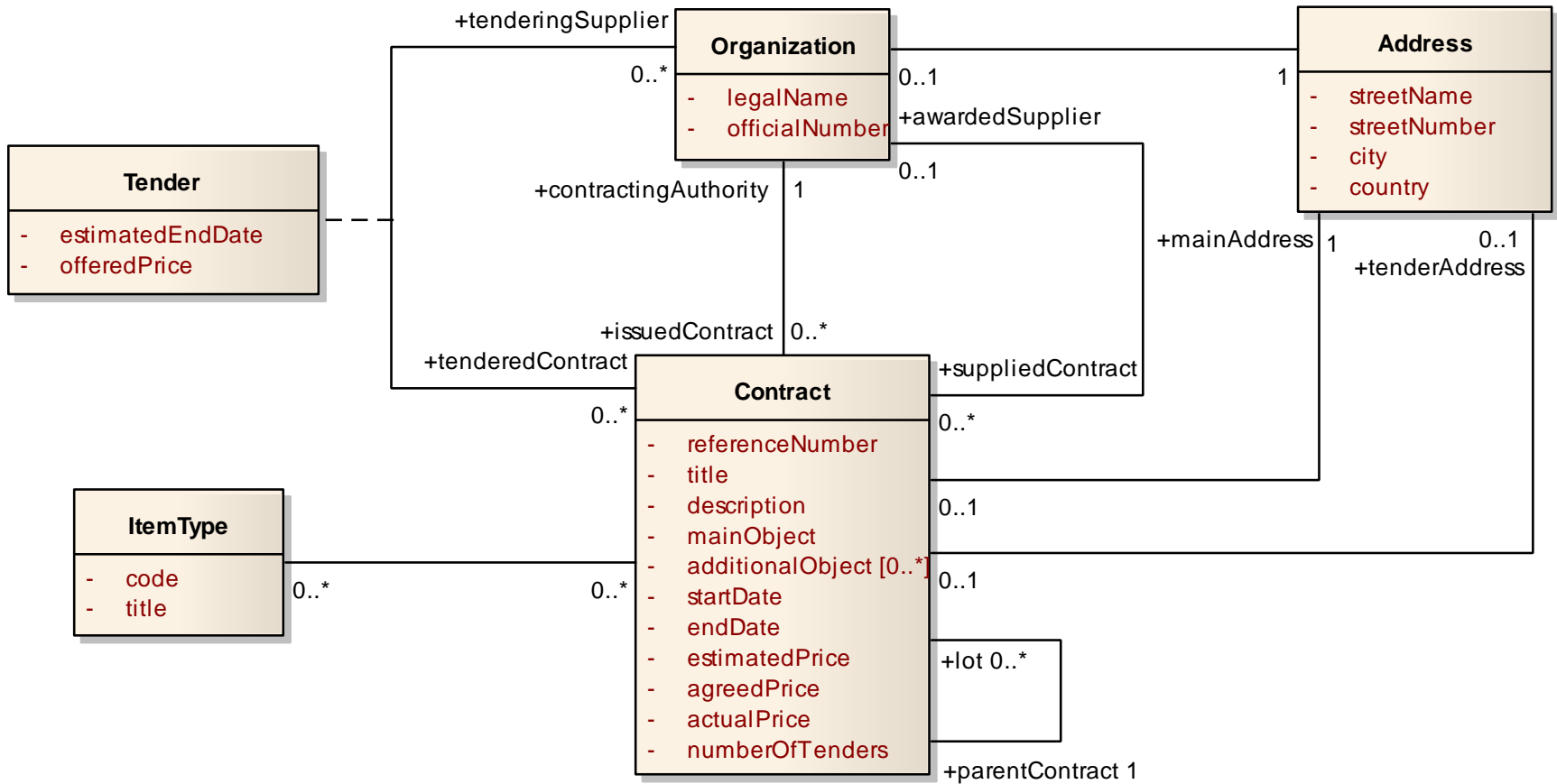
- Easier to understand XML schemas
- Just visualization – does not provide complete & non-redundant description

Solution: Model-Driven Architecture (MDA)



- Considers description of data at various abstraction levels
- **PIM** (Platform-Independent Model)
 - Description of data independent of any data model and particular user view
 - We describe entities, their attributes and mutual associations
- **PSM** (Platform-Specific Model)
 - Description of data from particular user view
 - Description of implementation in particular logical data model
 - Relational, XML, object, graph, ...

PIM Diagram



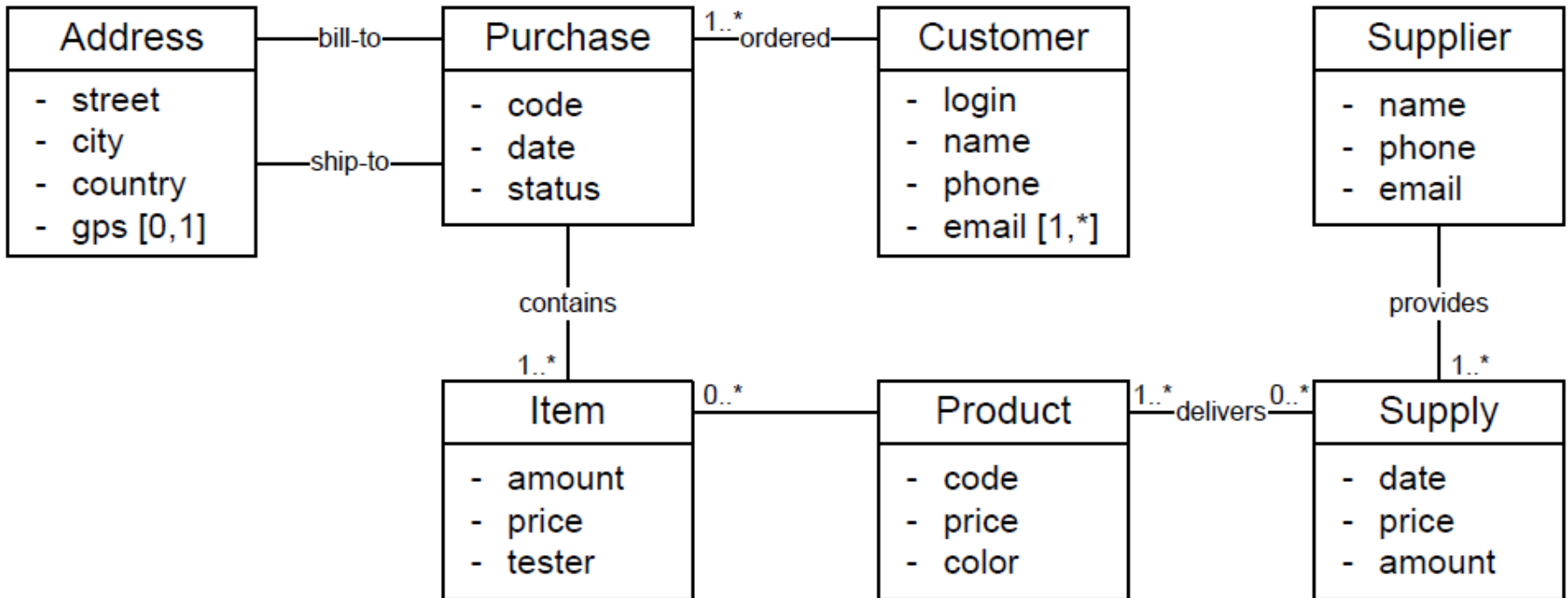
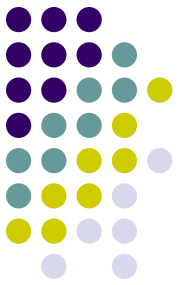
General UML

Model-Driven Architecture

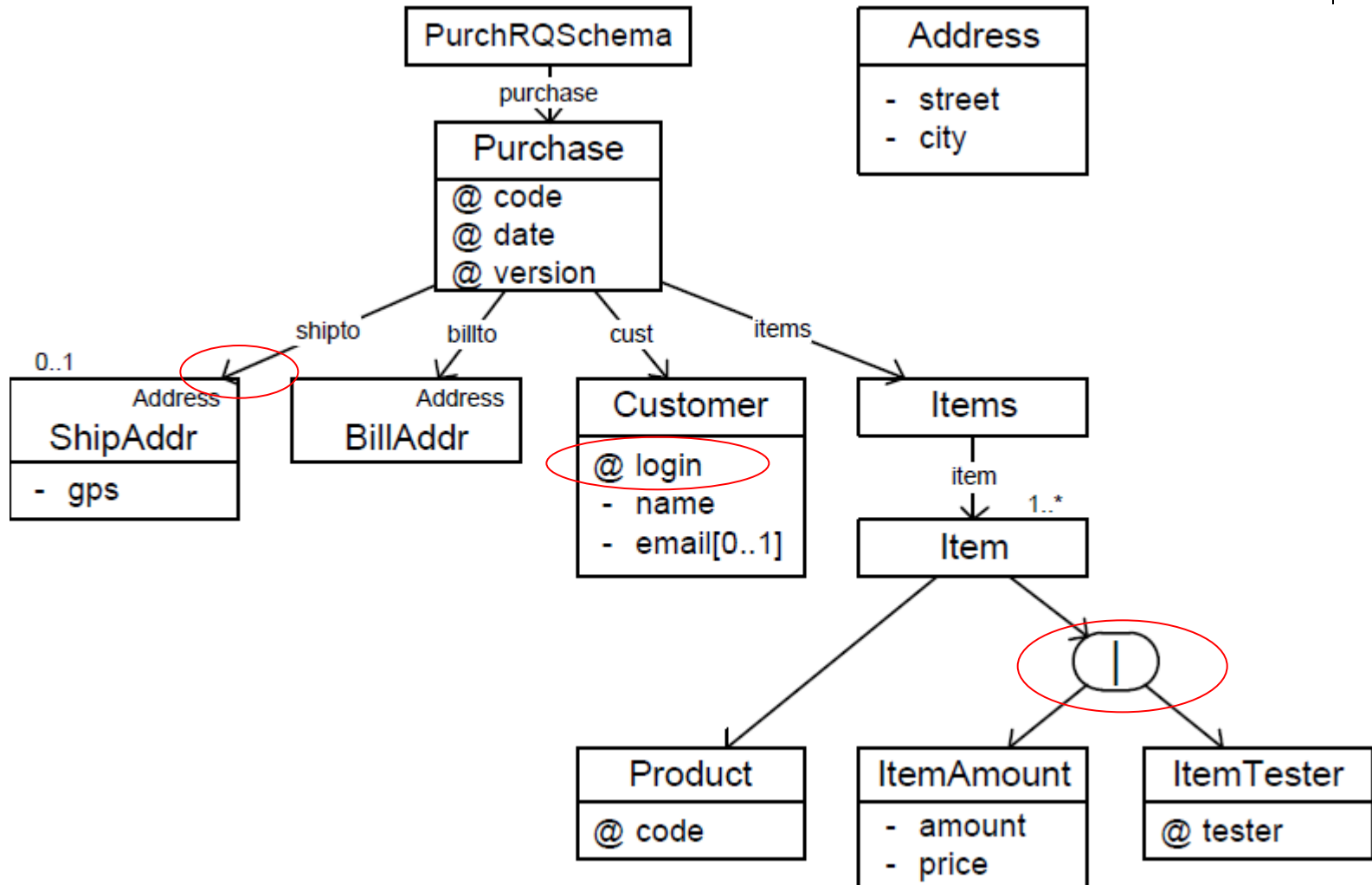


- Can be naturally applied on designing XML as well
- PSM = UML class model extended with set of stereotypes for modelling constructs of particular XML schema language
 - PSM for XML Schema

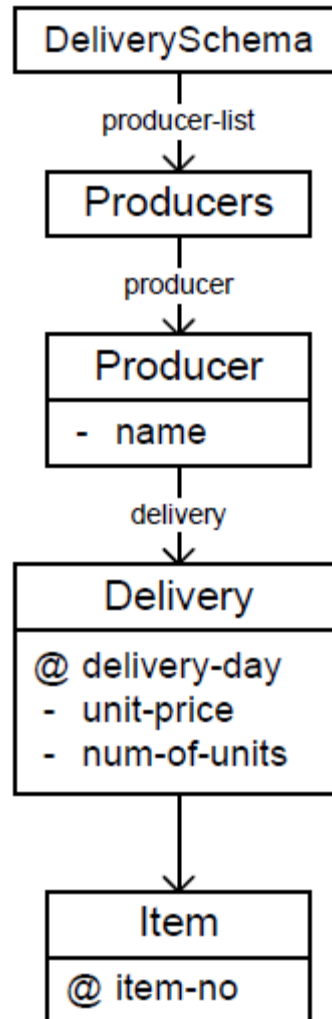
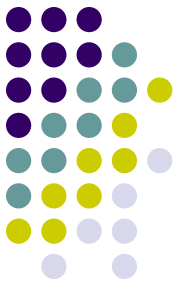
PIM Diagram



PSM Diagram – XML Schema



PSM Diagram – XML Schema




Model-Driven Architecture and XML



- Allows describing a problem domain independently of XML format
 - Complete and non-redundant description
- Problem: No or weak binding between PIM and PSM level in current tools
- ➔ Evolution of applications is hard to manage
 - Applications are usually dynamic
 - User requirements change, new information come, ...
 - Structure of data needs to be changed
 - Old schema S_{old} , new schema S_{new}
 - We still want to work with both old and new data
 - Without any loss if possible

Evolution of XML Applications



- Approaches:
 - XML schema/data evolution 
 - The old data valid against S_{old} are transformed to be valid against S_{new}
 - User poses queries over S_{new}
 - XML data versioning
 - We must preserve all versions V_1, V_2, \dots, V_n of the data
 - User poses queries over any $V_i; 1 \leq i \leq n$
 - Retrieves data from all versions V_1, V_2, \dots, V_n
 - Retrieves data from V_1, V_2, \dots, V_i

Evolution of XML Applications

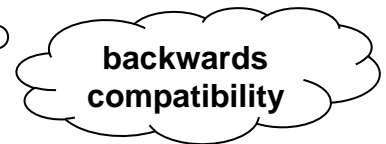


- Situations:
 - We are provided with S_{old} and S_{new} and we look for an optimal transformation sequence
 - We are provided with S_{old} and a sequence of changes $C = \{c_1, c_2, \dots, c_m\}$ made by a user
 - We can follow users steps in C
 - Problem: C is usually not optimal

Example: Oracle Types of XML Schema Evolution



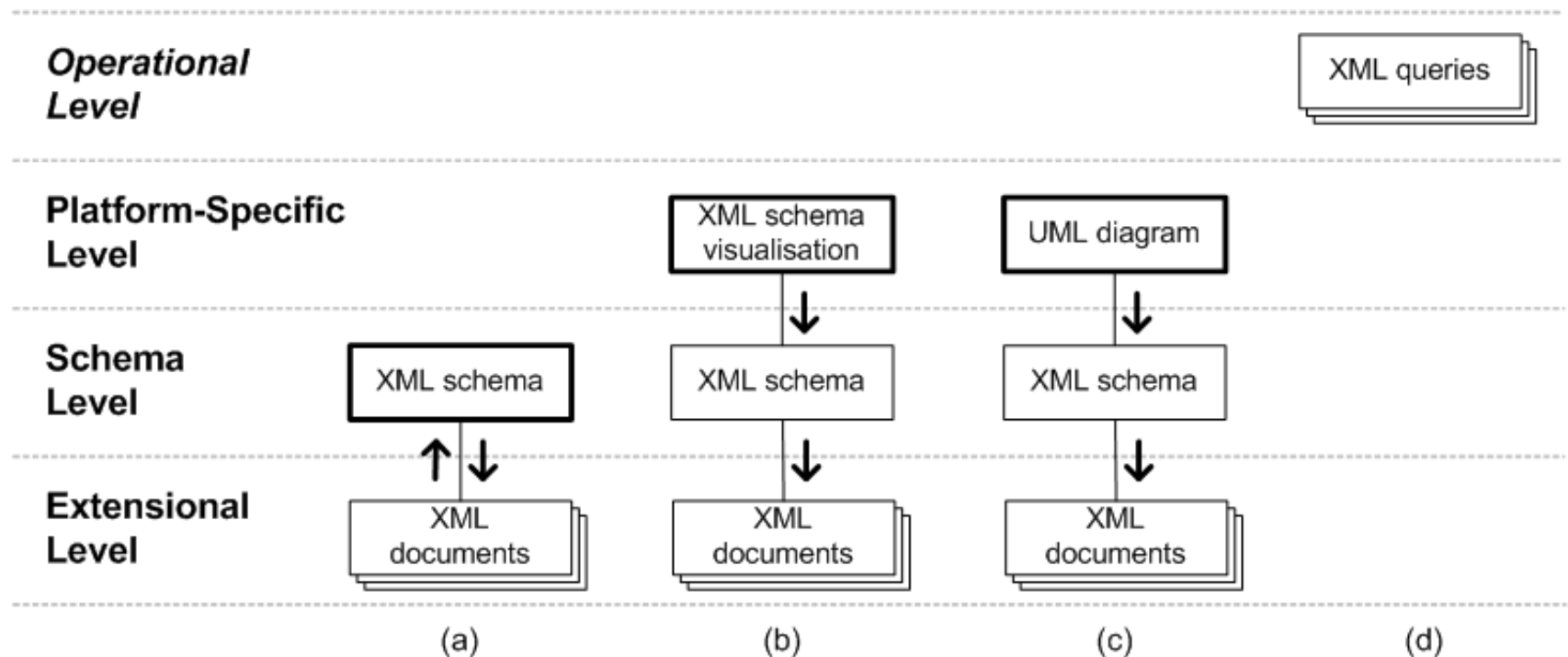
- Copy-based:
 - All documents that conform to S_{old} are copied to a temporary location
 - S_{old} is deleted
 - S_{new} is registered
 - Instance documents are transformed and inserted into their new locations from the temporary area
- In-place:
 - Does not require copying, deleting, inserting existing data
 - Much faster, but restricted:
 - Changes do not invalidate existing documents
 - Not changing the storage model
 - DB2 + MS SQL server support only this



Modelling, Evolution and Current XML Approaches



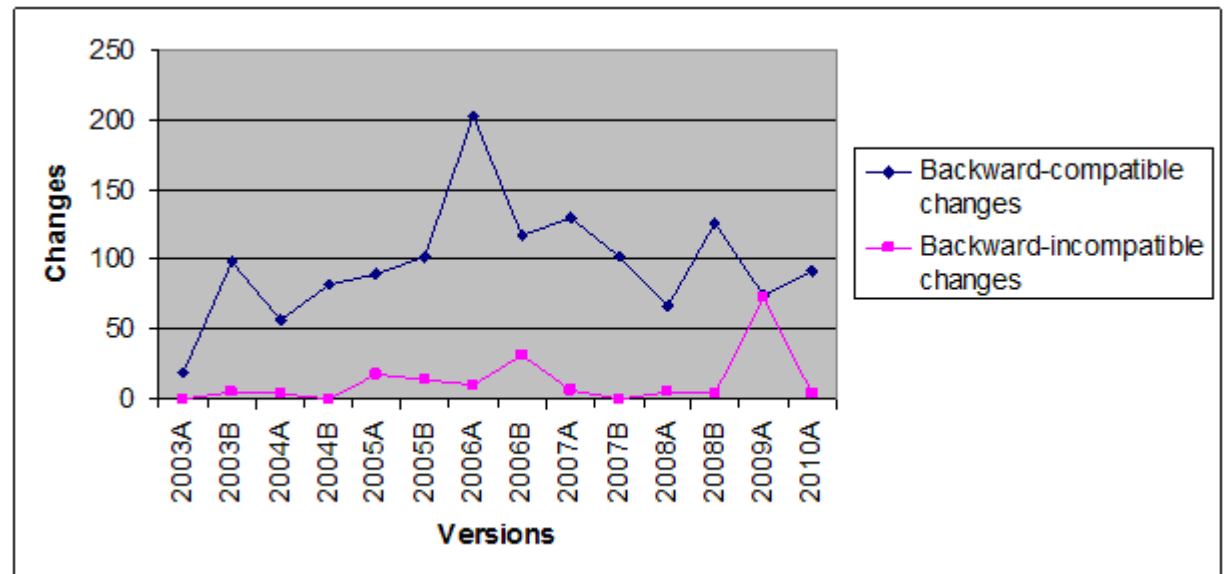
- Differ in
 - The level where user specifies the modifications
 - The way of propagation of modifications



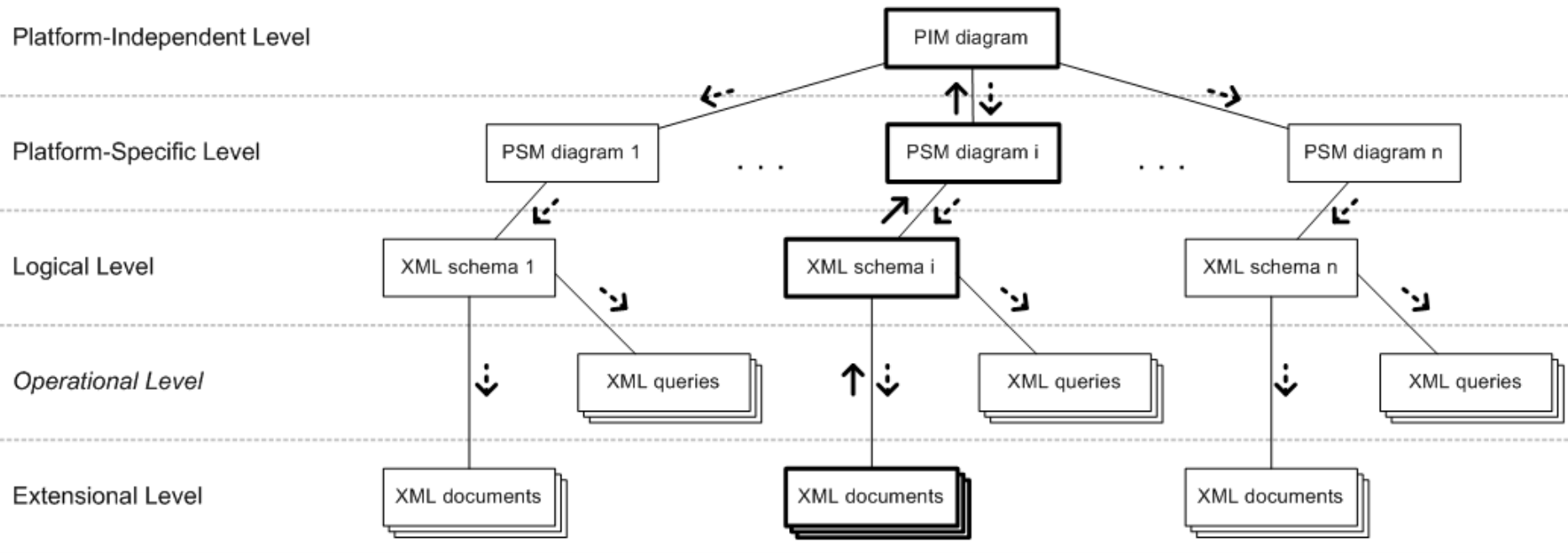
What If We Have Multiple Schemas?



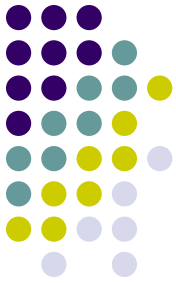
- **OpenTravel.org**
 - 2012: 319 XML schemas
 - Changes: twice a year
 - Solution: preserve backward compatibility as much as possible
- **Problems:**
 - Artificial schemas, strange structure
 - Backward compatibility is not always possible



Exploitation of MDA

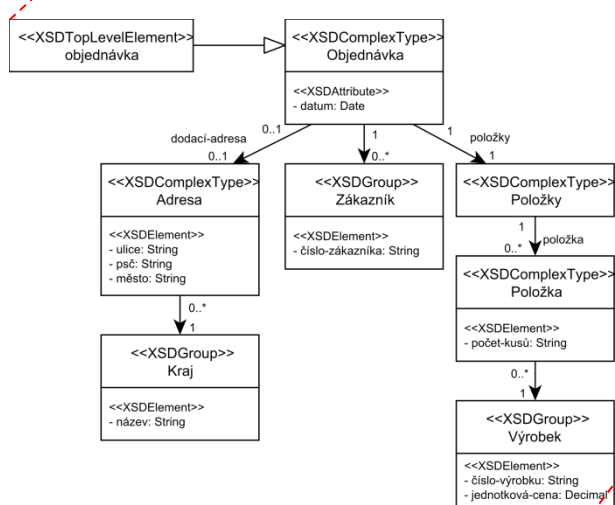
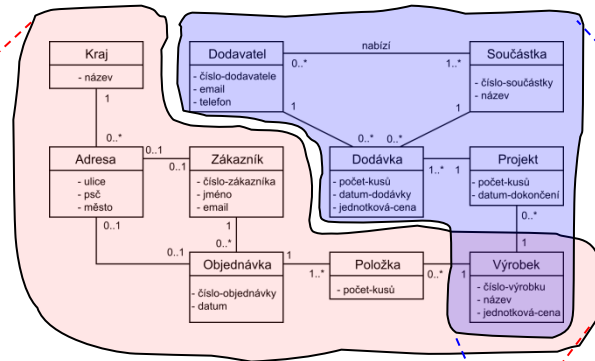


- Not many to many mappings, just one-to-many
 - Preserving of relations between all the levels
- Upwards and downwards propagation

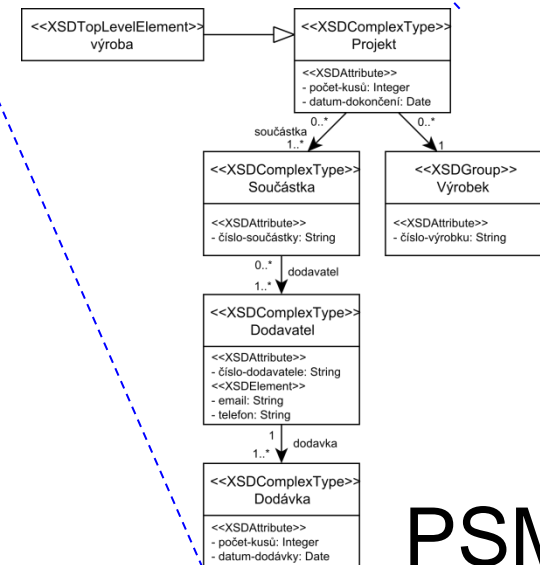


Exploitation of MDA

PIM



PSM 1



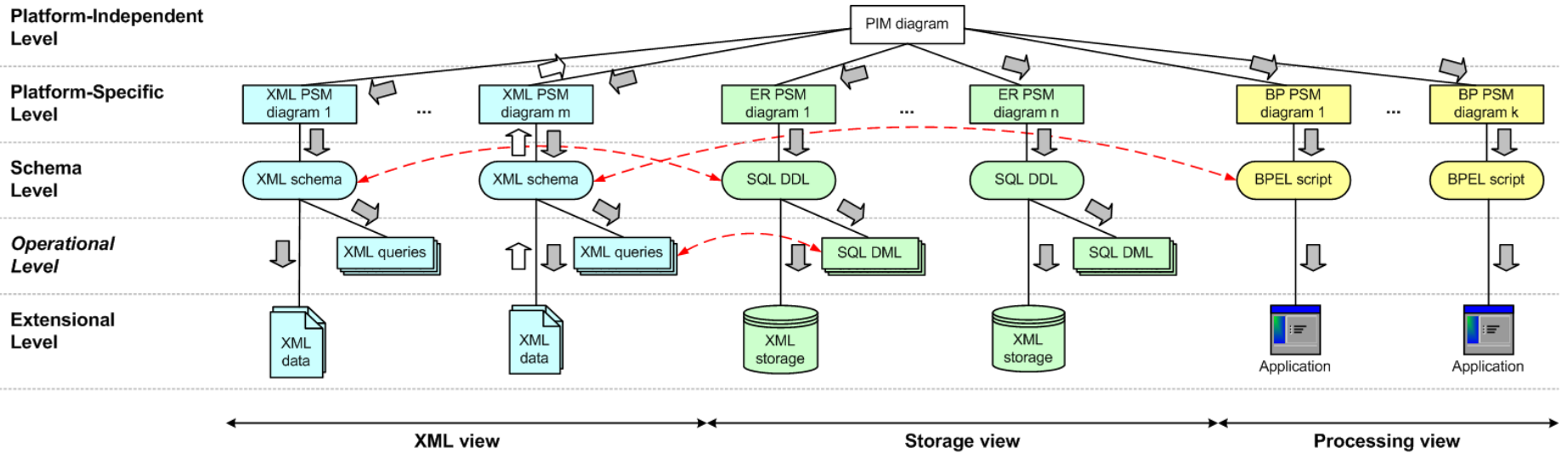
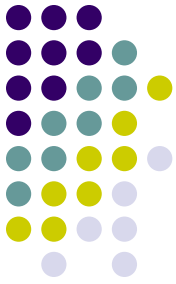
PSM 2



Related Problems

- **Forward Engineering**
 - The user starts with PIM, then derives PSM from PIM etc.
 - Top-down approach
- **Reverse Engineering**
 - A new schema (or a system of schemas) must be integrated
 - Bottom-up approach
 - We must find the mapping – schema matching problem
- **Adaptation of XML documents**
 - Semi-automatic generating of XSLT scripts
- **Adaptation of Queries**
 - Difficult problem

And We Can Go Farther...

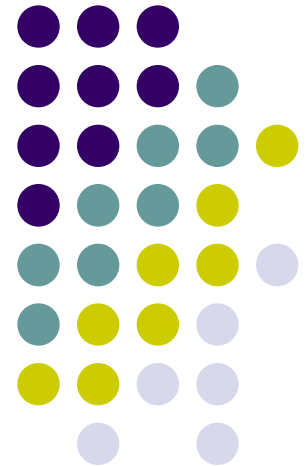




So...

- ...now we know how to create XML schemas
 - simple – XML editors
 - complex/evolving – XML modeling tools
- Where do we get the instances (XML data)?
 - If they are not created by the applications themselves
 - Typically for testing purposes

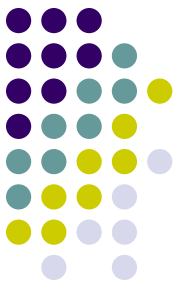
XML Benchmarking, XML Data Generators



What is the Purpose of Benchmarking?



- There exists a huge amount of systems for XML data management
 - Operations with XML data: parsing, validation, storing, querying, updating, transformation, compressing, ...
- Problems:
 - User: Searching for a system optimal for a given application
 - Vendor: Testing of correctness/performance of own system, comparison with other systems
 - Aim: Finding advantages of a particular system
 - General analysis: Comparison of various systems from various points of view
 - Aim: Objective comparison
- Solution: We can find results of a respective analysis
- But:
 - The development of systems is fast \Rightarrow results are soon obsolete
 - The found results usually do not cover exactly what we want \Rightarrow We need to prepare **own tests**



What is a Benchmark?

- Benchmark = a set of test cases = data + operations (+ expected results)
- We test: correctness of operations, functionality, efficiency, ...
- Our case:
 - Data = XML documents (possibly with a schema)
 - Operations = any (XML) operation
- Classification according to the **type of data**
 - Real vs. synthetic
 - Fixed vs. dynamic
- Classification according to the **type of operations**
 - Parsing, validation, querying, ...
- Classification according to the **type/version** of the tested **technology**
 - DTD vs. XML Schema, XPath vs. XQuery, XPath 1.0 vs. XPath 2.0, ...

Fixed Sets of Real-World XML Data



- The simplest approach, rather interesting than useful
 - The Bible in XML, Shakespeare's plays in XML, ...
- Exports of XML databases
 - The most usual data, but just data-oriented documents
 - e.g., **IMDb** (films and actors), **DBLP** (research papers), **Medical Subject Headings** (medical terms), ...
- Data stores of real-world XML data
 - Sometimes involve XML data which were not originally in XML
 - e.g., **INEX**, **Ibiblio**, ...
- Disadvantages: Usually simple structure and no operations
- Special collections of real-world XML data
 - Unusual structures
 - e.g., protein sequences, astronomical data from NASA, linguistic trees, ...



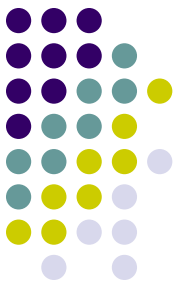
Example: The Bible in XML

```
<?xml version="1.0" ?>
- <bible>
- <testament value="Old Testament">
- <book title="Genesis">
- <chapter number="1">
  <verse number="1">In the beginning God created the heaven and the earth.</verse>
  <verse number="2">And the earth was without form, and void; and darkness [was] upon the face of the
    deep. And the Spirit of God moved upon the face of the waters.</verse>
  <verse number="3">And God said, Let there be light: and there was light.</verse>
  <verse number="4">And God saw the light, that [it was] good: and God divided the light from the
    darkness.</verse>
  <verse number="5">And God called the light Day, and the darkness he called Night. And the evening and
    the morning were the first day.</verse>
  <verse number="6">And God said, Let there be a firmament in the midst of the waters, and let it divide
    the waters from the waters.</verse>
  <verse number="7">And God made the firmament, and divided the waters which [were] under the
    firmament from the waters which [were] above the firmament: and it was so.</verse>
  <verse number="8">And God called the firmament Heaven. And the evening and the morning were the
    second day.</verse>
  <verse number="9">And God said, Let the waters under the heaven be gathered together unto one
    place, and let the dry [land] appear: and it was so.</verse>
  <verse number="10">And God called the dry [land] Earth; and the gathering together of the waters
    called he Seas: and God saw that [it was] good.</verse>
  <verse number="11">And God said, Let the earth bring forth grass, the herb yielding seed, [and] the
    fruit tree yielding fruit after his kind, whose seed [is] in itself, upon the earth: and it was so.</verse>
  <verse number="12">And the earth brought forth grass, [and] herb yielding seed after his kind, and the
    tree yielding fruit, whose seed [was] in itself, after his kind: and God saw that [it was] good.</verse>
  <verse number="13">And the evening and the morning were the third day.</verse>
  <verse number="14">And God said, Let there be lights in the firmament of the heaven to divide the day
    from the night; and let them be for signs, and for seasons, and for days, and years:</verse>
  <verse number="15">And let them be for lights in the firmament of the heaven to give light upon the
    earth: and it was so.</verse>
```

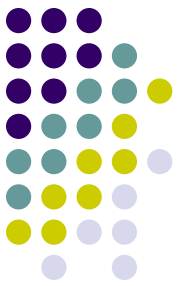


5 MB

Example: Shakespeare's Plays in XML



```
<!ENTITY amp "&#38;#38;">
<!ELEMENT PLAY      (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
                     PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE     (#PCDATA)>
<!ELEMENT FM        (P+)>
<!ELEMENT P         (#PCDATA)>
<!ELEMENT PERSONAE  (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP    (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA   (#PCDATA)>
<!ELEMENT GRPDESCR  (#PCDATA)>
<!ELEMENT SCNDESCR  (#PCDATA)>
<!ELEMENT PLAYSUBT  (#PCDATA)>
<!ELEMENT INDUCT    (TITLE, SUBTITLE*, (SCENE+ | (SPEECH | STAGEDIR | SUBHEAD) +))>
<!ELEMENT ACT       (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
<!ELEMENT SCENE     (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD) +)>
<!ELEMENT PROLOGUE  (TITLE, SUBTITLE*, (STAGEDIR | SPEECH) +)>
<!ELEMENT EPILOGUE  (TITLE, SUBTITLE*, (STAGEDIR | SPEECH) +)>
<!ELEMENT SPEECH    (SPEAKER+, (LINE | STAGEDIR | SUBHEAD) +)>
<!ELEMENT SPEAKER   (#PCDATA)>
<!ELEMENT LINE      (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR  (#PCDATA)>
<!ELEMENT SUBTITLE  (#PCDATA)>
<!ELEMENT SUBHEAD   (#PCDATA)>
```



Example: IMDb – DTD

```
<!ELEMENT W4F_DOC (Movie)>
<!ELEMENT Movie (Title,Year,Directed_By,Genres,Cast)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Directed_By (Director)*>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT Genres (Genre)*>
<!ELEMENT Genre (#PCDATA)>
<!ELEMENT Cast (Actor)*>
<!ELEMENT Actor (FirstName,LastName)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>

<!ELEMENT W4F_DOC (Actor)>
<!ELEMENT Actor (Name,Filmography)>
<!ELEMENT Name (FirstName, LastName)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT Filmography (Movie)*>
<!ELEMENT Movie (Title,Year)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
```



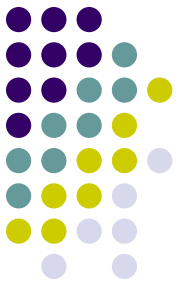
Example: DNA in XML

```
<SNP>
  <SNP_START>107</SNP_START>
  <SNP_END>107</SNP_END>
  <SNP_DB-REFERENCE_DBSNP>3751853</SNP_DB-REFERENCE_DBSNP>
  <SNP_BASE-DBSNP>T/C</SNP_BASE-DBSNP>
  <SNP_BASE-CDNA>T/C</SNP_BASE-CDNA>
  <SNP_STRAND>+</SNP_STRAND>
</SNP>
<EXON-CDNA>
  <EXON-CDNA_START>1</EXON-CDNA_START>
  <EXON-CDNA_END>7604</EXON-CDNA_END>
</EXON-CDNA>
<EXON-GENOME>
  <EXON-GENOME_START>31123283</EXON-GENOME_START>
  <EXON-GENOME_END>31130886</EXON-GENOME_END>
</EXON-GENOME>
</CDNA-INFO>
<SEQUENCE>
  <NUCLEOTIDE_ORIGINAL>GAGCAGATTCTACTGTTCTTAAAGGACAGTAATGCCTTTTGGAGTCTGGTCTGAAGAACATAACAGGTCTGTGATC
AGAAGTAGGTTGCATCTCTCTCAACTTTAATTTCCCTTAGCTATACCTGTAGGGATGACTTAAGCCTAGGGGAGCTCCTATATTTGGGAAGCTTGTGCACAGG
GAAGCCTTAAATGATGGTGCCTGCAGATTGGATCTAGTAGAAATTAGGTCCTTGGGCATGGATGCTTGGGGAACCTCTCAGTGACCTCAGGTGAACCTTGTG
CTCGTAGAGCCAAGAGGCCAAGTTAATTCAGGCCTTCCCTTTTGACCACTGCCCCCTCTTCCCTAGGCCTTGGCCCCCTCCACCAGAGGAAGGTGCTGCCACGTG
TCTGCTCCTTCTGAACCTCCAGGTTTCTGCTACGTTGCCCATGGAGGACACACCCCCCTCACTCAGCTGCTCCGACTGTCAGCGCCACTTTCCAGCCTCC
CAGAGCTCTCTCGGCACCGAGAAGTCTCCATCCATCTCCAACCCAGGACAGTGAGGAGGCTGACAGCATCCCTCGGCCCTACCGTTGTCAGCAGTGTGGGC
```

Analysis of Real-World XML Data

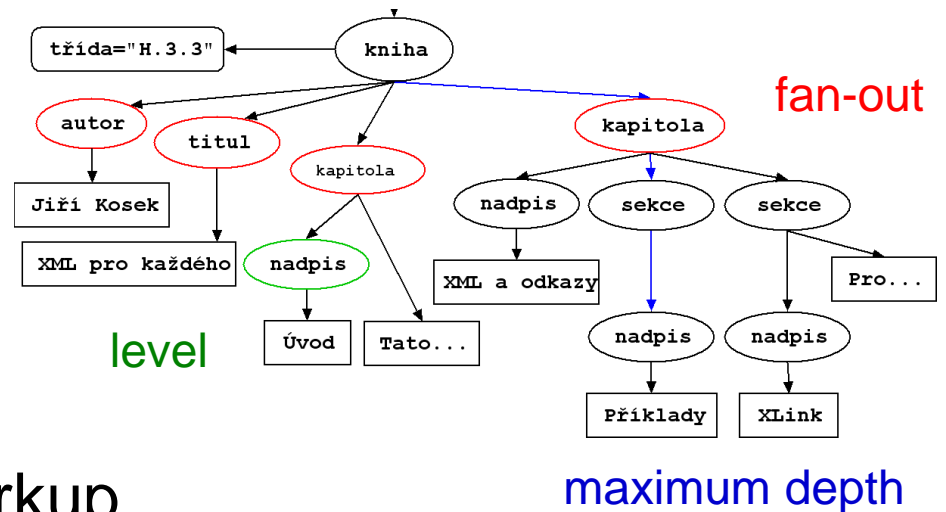


- How do the real-world XML data look like?
 - In general: They are very simple
- How to find out exact values:
 1. Gather a representative set of XML data
 - XML documents + XML schemas, XML queries, XSLT scripts ...
 2. Measure the characteristics
- Current analyses:
 - Analyze XML documents, schemas, documents vs. schemas, DTDs vs. XSDs
 - There seem to exist no analyses of XML operations
 - Not simple gathering of queries \Rightarrow we need special tools



Characteristics of XML Data

- XML document = tree
 - Number of nodes
 - Element, attribute, text, mixed-content
 - Distance of nodes
 - Distance from root
 - Node level
 - Tree depth
 - Fan-out
 - Lengths of text nodes
 - Amount of text vs. markup
- Minimum, maximum, average



Characteristics of XML Schemas



- The same as in XML data but with regard to schemas
 - i.e., what is allowed (not what is used in instances)
- Complexity and types of content models (regular expressions):
 - e.g., **depth of content model** α is defined as follows:
 - $\text{depth}(\emptyset) = \text{depth}(\varepsilon) = 0$
 - $\text{depth}(\text{pcdata}) = \text{depth}(a) = 1$
 - where $\forall a \in \Sigma$
 - $\text{depth}(\alpha_1, \alpha_2, \dots, \alpha_n) = \text{depth}(\alpha_1 | \alpha_2 | \dots | \alpha_n) = \max(\text{depth}(\alpha_i)) + 1; 1 \leq i \leq n$
 - where α_i are regular expressions
 - $\text{depth}(\beta^*) = \text{depth}(\beta^+) = \text{depth}(\beta?) = \text{depth}(\beta) + 1$
 - where β is a regular expression



DTD Analysis (1)

- Paper [1] 2001: 12 real-world DTDs
 - Number of elements, attributes, entities
 - Depth of content model
 - Mixed-content elements
 - ANY, ID, IDREF(S)
 - Attributes: implied, required and fixed
- Findings:
 - Depth < 6
 - ID and IDREF(S) are not used often
 - Real-world DTDs contain lots of mistakes
 - Apparently they are not used properly
 - Not for checking validity, but for documentation
 - General observations of limits of DTD
 - To be solved in XML Schema

DTD Analysis (2)



- Paper [2] 2002: 60 DTDs
 - Local characteristics: Different types of elements, depth of content model, non-determinism
 - Global characteristics: Reachability, recursion, simple paths, cycles, fan-in
- Conclusions:
 - Local: depth < 9, non-deterministic content models occur often, though the standard does not allow them
 - Another proof of incorrect usage
 - Global: Unreachable elements do not occur often, recursion occurs in 58% of DTD
 - Many methods do not support recursion



Analysis of DTD vs. XSD

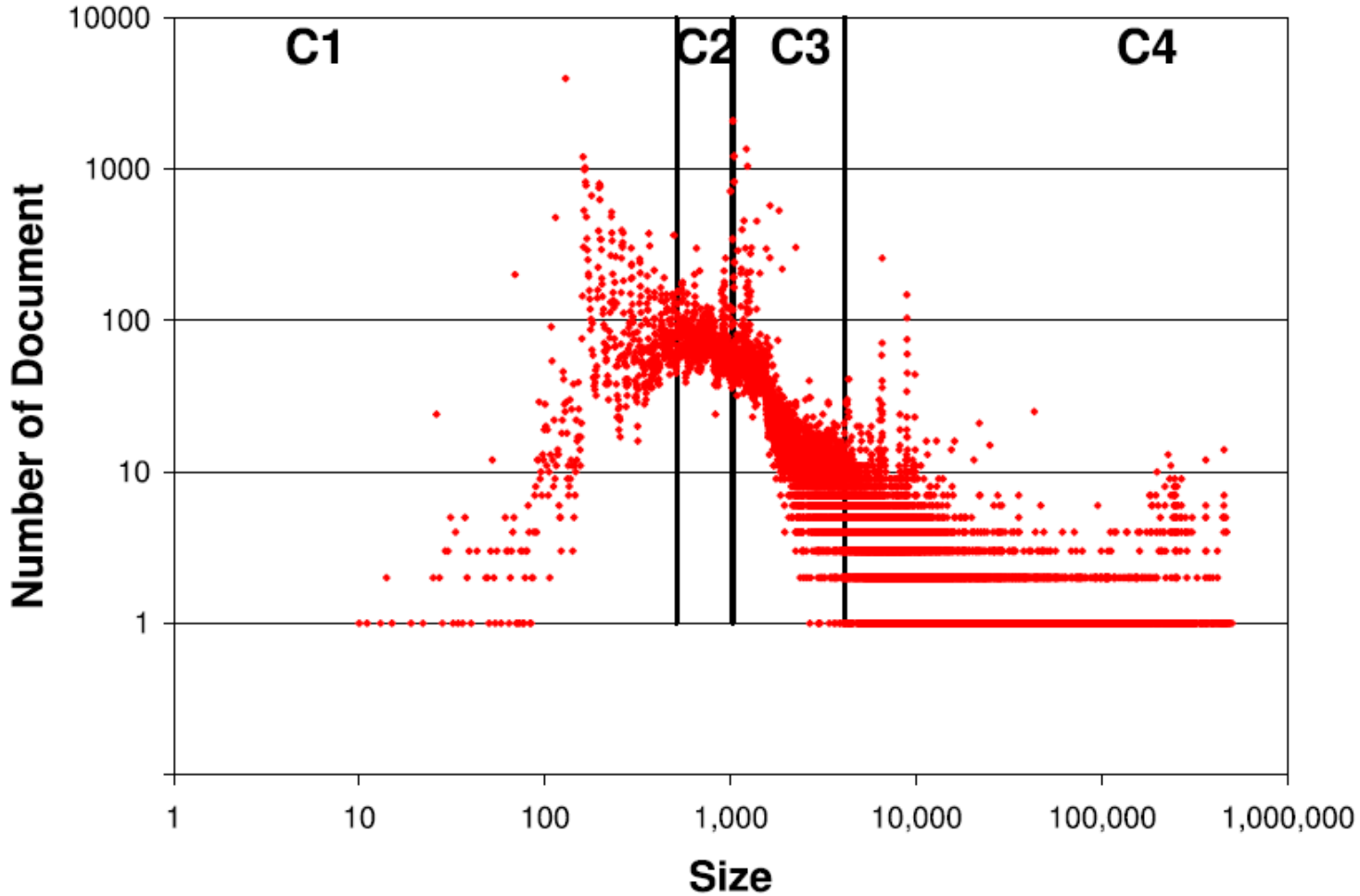
- Paper [3] 2004: 109 DTD, 93 XSD (30% of XSDs incorrect)
 - Aim: Which of XML Schema constructs are used in practise?
- Exploitation of XML Schema constructs:
 - restriction (73%) of simple types
 - extension (37%) and restriction (7%) of complex types
 - substitution groups (11%), unordered sequences (4%)
 - unique (7%), key/keyref (4%) constructs
 - namespaces (22%)
 - 85% XSD = local tree grammars
 - Expressive power of DTD
- Complexity of regular expressions:
 - Similar results to the previous ones

Analysis of XML Documents



- Paper [4] 2003: 200 000 XML documents
- Analysis of XML web:
 - Clustering according to domains (.com, .edu, .net, ...) and geography (Asia, EU, ...)
 - Number of documents per zone
 - DTD (48%) and XSD (0.09%) exploitation
 - Namespaces exploitation (40%)
 - Types of documents (.rdf, .rss, .wml, .xml, ...)
- Structural analysis:
 - Size of documents (4,6kB on average), text vs. markup (50:50), mixed content (72%), recursion (15%), ..., depth (< 4), ...

Distribution of Documents by Size



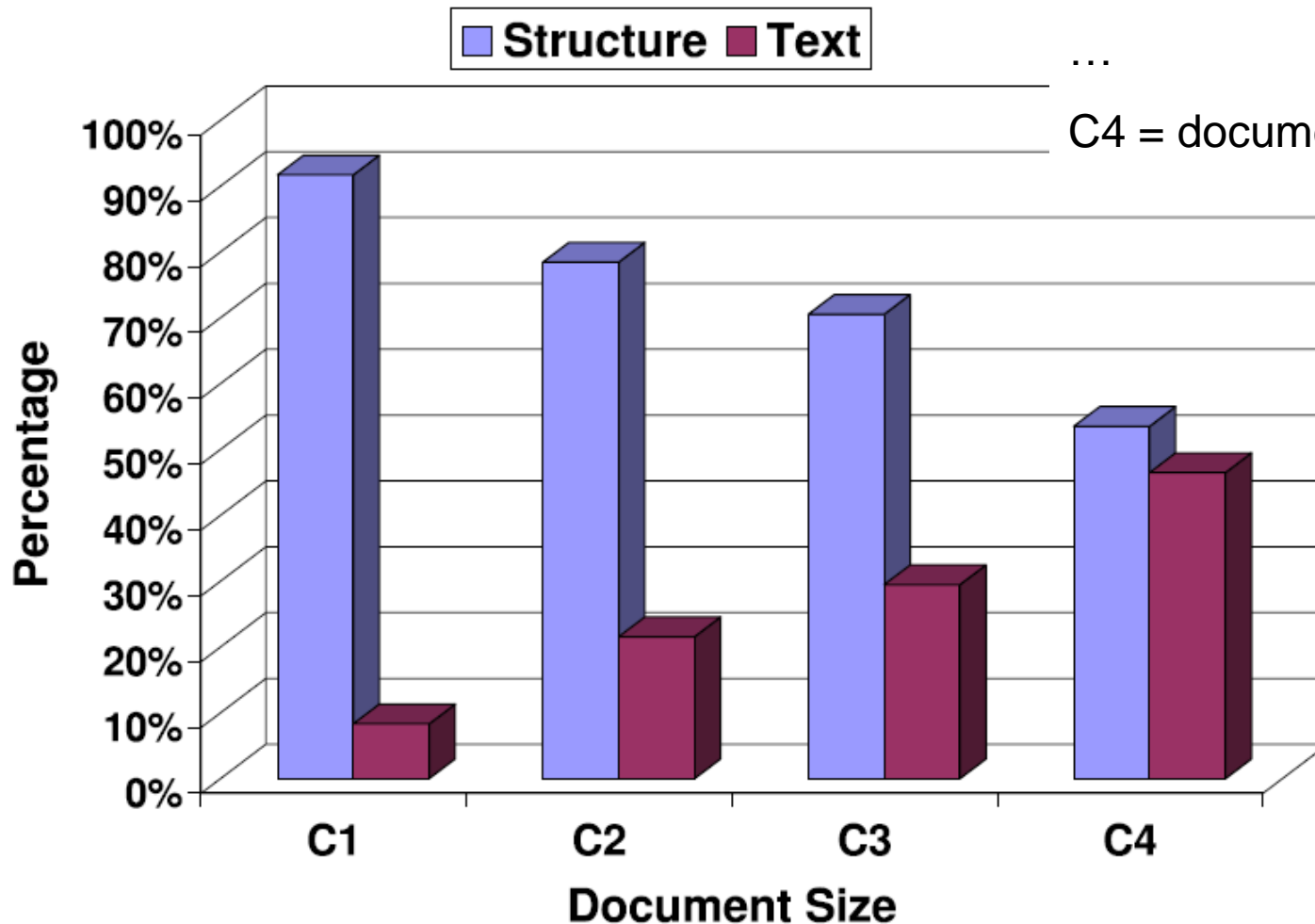
Size of Structural vs. Textual Content by Document's Cluster



C1 = documents < 512B

...

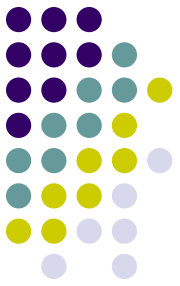
C4 = documents > 4096B



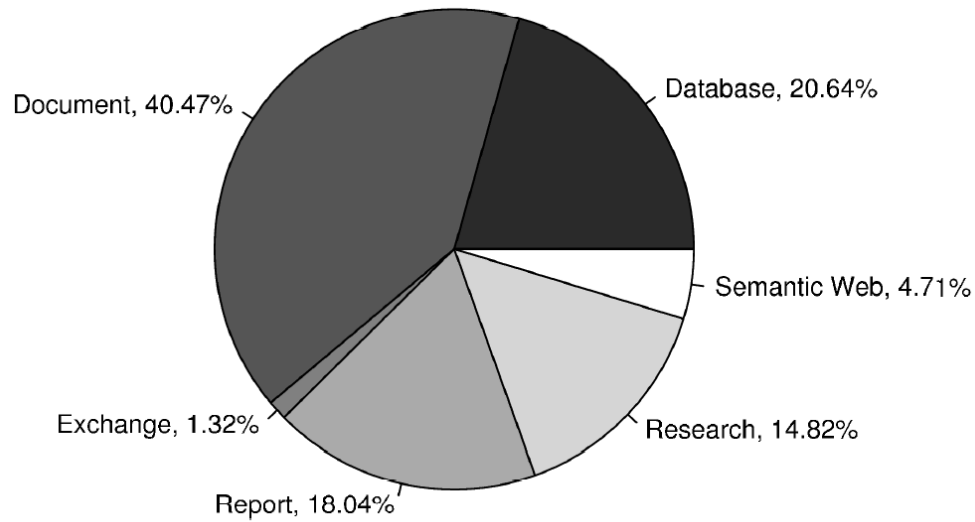
Analysis of Documents vs. Schemas (1)



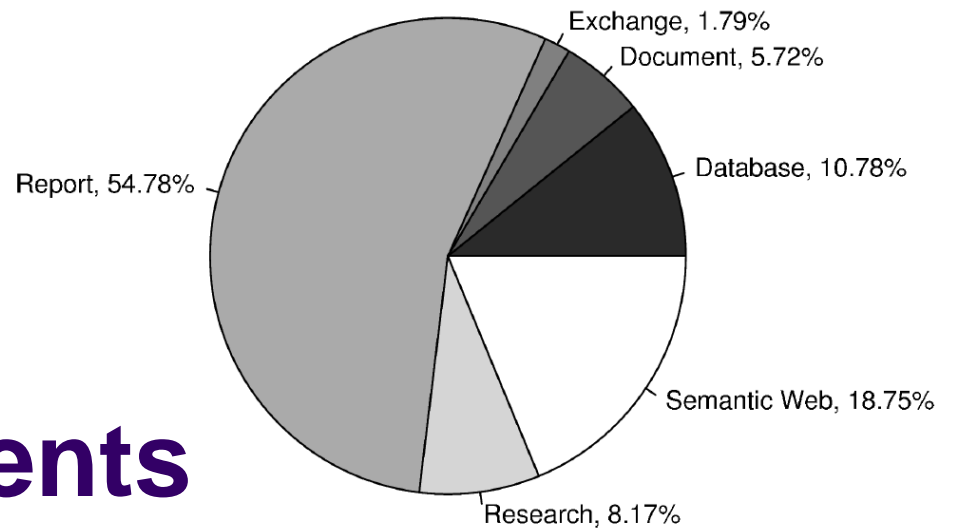
- Paper [5] 2006: 16 500 documents from 133 collections divided into 6 categories
- Problem: Randomly downloaded data are trivial (2000 documents have depth 0 [4]) ⇒ semi-automatic collection of data
- Collections:
 - data-centric documents (**dat**)
 - e.g., database exports, lists of employees, lists of IMDb movies and actors, etc.
 - document-centric documents (**doc**)
 - e.g., Shakespeare's plays, XHTML documents, novels in XML, DocBook documents, etc.
 - documents for data exchange (**ex**)
 - e.g., medical information on patients and illnesses, etc.
 - reports (**rep**), i.e., overviews or summaries of data
 - research documents (**res**)
 - e.g., protein sequences, DNA/RNA structures, NASA findings, etc.
 - semantic web documents (**sem**), i.e., RDF documents



Number of Files



Sizes of Documents



Analysis of Documents vs. Schemas (2)

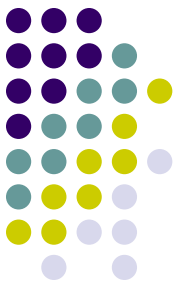


- Conclusions:
 - Most of the data contains mistakes and errors
 - 74.6% of DTDs, 38.2% of XSDs – more precise
 - XML documents contain much simpler data than respective schemas allow
 - Recursion, mixed content
 - Depth of mixed-content elements is < 3 on average
 - Recursion is quite common, but simple
 - Single element, no branching
 - Nesting < 5
 - Most common XML Schema constructs are simple types, default values, ID, IDREF(S), unordered sequences



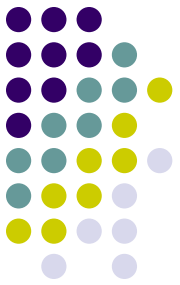
XML Data Generators

- Solution to problems of real-world collections: synthetic XML data
- Classification:
 - **Schema-less generators** – e.g., NiagDataGen, MemBeR
 - General parameters: depth of XML document, number of elements, ...
 - **Schema-based generators** – ToXgene, VeXGene
 - Input: annotated schema
 - Distribution of attributes, distribution of lengths of textual values, values of data types, ...
- Aim: As realistic structure as possible
 - Zipf's law, Markov chains, statistical distributions, ...
- Disadvantages: Too many parameters \Rightarrow user unfriendly



MemBeR Example

```
<Tree xmlns="http://microbenchmarks.org/treegen"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://microbenchmarks.org/treegen TreeGen.xsd"
      depth="6"
      size="10000"
      distribution="normal">
  <Tag name="t01" levels="1" frequency="1" fanout="10"/>
  <Tag name="t02" levels="2" frequency="0.5" fanout="3"/>
  <Tag name="t03" levels="2" frequency="0.5" fanout="1"/>
  <Tag name="t04" levels="3-4" frequency="0.33" fanout="6"/>
  <Tag name="t05" levels="3-4" frequency="0.33" fanout="6"/>
  <Tag name="t06" levels="3-4" frequency="0.33" fanout="3"/>
  <Tag name="t06" levels="5" frequency="1" fanout="0"/>
</Tree>
```



ToXgene Example

- Toronto XML Server Data Generator
 - <http://www.cs.toronto.edu/tox/toxgene/>
 - http://www.cs.toronto.edu/tox/toxgene/docs/ToXgene_manual.pdf
 - Installation, all constructs
- Annotated “XML schema”
 - Not true XSD
 - `tsl` file
- Root element `tox-template`:

```
<!ELEMENT tox-template  
  (tox-distribution |  
   simpleType | complexType |  
   tox-list |  
   tox-document) *>
```

tox-distribution



```
<tox-distribution name="age" type="normal"  
  minInclusive="18" maxInclusive="127"  
  mean="30" variance="15">  
</tox-distribution>
```

```
<tox-distribution name="watches" type="exponential"  
  minInclusive="0" maxInclusive="10" mean="4">  
</tox-distribution>
```

```
<tox-distribution name="discount" type="user-defined"  
min="0" max="30">  
  <enumeration value="0" tox-percent="50"/>  
  <enumeration value="5" tox-percent="25"/>  
  <enumeration value="10" tox-percent="15"/>  
  <enumeration value="30" tox-percent="10"/>  
</tox-distribution>
```

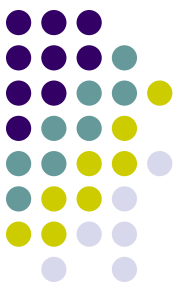
simpleType



```
<simpleType name="pick_category">  
  <restriction base="nonNegativeInteger">  
    <tox-number tox-distribution="c1"/>  
  </restriction>  
</simpleType>
```

```
<simpleType name="lname">  
  <restriction base="string">  
    <tox-string type="lname"/>  
  </restriction>  
</simpleType>
```

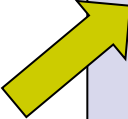
```
<simpleType name="year">  
  <restriction base="string">  
    <tox-value>1942</tox-value>  
  </restriction>  
</simpleType>
```



tox-string types

Type	Description
text	A textual string generated according to the TPC-H benchmark [5] rules.
word	A word, extracted from the list of words used in the XMark benchmark [4].
xmrk_text	A textual string generated according to the XMark benchmark rules.
fname	A first name, extracted from the list of first names in the XMark benchmark.
lname	A last name, extracted from the list of last names in the XMark benchmark.
city	A city name, extracted from the list of cities in the XMark benchmark.
province	A province name, extracted from the list of provinces in the XMark benchmark.
country	A country name, extracted from the list of countries in the XMark benchmark.
domain	An internet domain name, extracted from the list of domains in the XMark benchmark.
email	An email address of the form $x.y@z$, where x is an instance of <code>fname</code> , y is an instance of <code>lname</code> and z is an instance of <code>domain</code> .
gibberish	A random string in $\{a, \dots, z, \dots, A, \dots, Z, 0, \dots, 9\} \cup \{\#, \wedge, *, (,), _ , -, =, \$, +, \{, \}, [,], ?, \dots, /, \sim, \}$.

complexType



```
<complexType name="PurchaseOrderType">
  <element name="shipTo" type="USAddress"/>
  <element name="billTo" type="USAddress"/>
  <element name="comment" type="string"/>
  <element name="items" type="Items"/>
  <attribute name="orderDate">
    <simpleType>
      <restriction base="string">
        <tox-string type="city"/>
      </restriction>
    </simpleType>
  </attribute>
</complexType>
```

tox-document



```
<tox-document name="output/review" copies="1000"  
    starting-number="0">  
  <element name="purchaseOrder"  
    type="PurchaseOrderType"  
    minOccurs="1" maxOccurs="1"/>  
</tox-document>
```

XML Benchmarking: Parsing and Validation



- Primary operation with XML data
- W3C: XML Conformance Test Suites
 - <http://www.w3.org/XML/Test/>
 - Check against: W3C XML 1.0 Recommendation, Extensible Markup Language (XML) 1.0 (Second Edition), Extensible Markup Language (XML) 1.0 (Third Edition), Extensible Markup Language (XML) 1.1, Extensible Markup Language (XML) 1.0 (Fourth Edition), Extensible Markup Language (XML) 1.1 (Second Edition), Proposed Extensible Markup Language (XML) 1.0 (Fifth Edition), Namespaces in XML 1.0, Namespaces in XML 1.0 (Second Edition), Namespaces in XML 1.1, Namespaces in XML 1.1 (Second Edition)
 - 2 000 XML documents
 - Binary tests
 - Parser must correctly accept/refuse document
 - Output tests
 - Parser must correctly identify a mistake in document

Example: XML Conformance Test Suites



```
<!DOCTYPE doc [  
<!ELEMENT doc (#PCDATA)>  
<!ENTITY e "<![CDATA[&foo;]]>">  
>  
<doc>&e;</doc>
```



```
<doc>&amp;foo;</doc>
```

„Test demonstrates that all text within a valid CDATA section is considered text and not recognized as markup.“

XML Benchmarking: Querying



- W3C:
 - XML Query Use Cases
 - <http://www.w3.org/TR/xquery-use-cases/>
 - Not a benchmark but a set of examples of XML queries
 - XML Query Test Suite
 - <http://www.w3.org/XML/Query/test-suite/>
 - 15 000 test examples (query + result)
 - Tests full support for XQuery
- Lots of existing benchmarks
 - XMark, XOO7, XMach-1, MBench, XBench, XPathMark, TPoX
 - Test the amount of supported constructs + efficiency
 - Assumption: tested systems return correct results

Example: XML Query Use Cases



„For each book found at both `bstore1.example.com` and `bstore2.example.com`, list the title of the book and its price from each source.“

```
<books-with-prices>
{
  for $b in doc("http://bstore1.example.com/bib.xml")//book,
    $a in doc("http://bstore2.example.com/reviews.xml")//entry
  where $b/title = $a/title
  return
    <book-with-prices>
      { $b/title }
      <price-bstore2>{ $a/price/text() }</price-bstore2>
      <price-bstore1>{ $b/price/text() }</price-bstore1>
    </book-with-prices>
}
</books-with-prices>
```

Example: XML Query Test Suite



`upper-case()`

`upper-case("string", "wrong param")`

`upper-case(()) eq ""`

`lower-case("ABc!D") eq "abc!d"`

`sub-string("a string")`

`sub-string("a string", 1, 2, "wrong param")`

`substring((), 1, 3) eq "`

`substring("12345", -42, 1 div 0E0) eq "12345"`

`substring("metadata", 4, 3) eq "ada"`

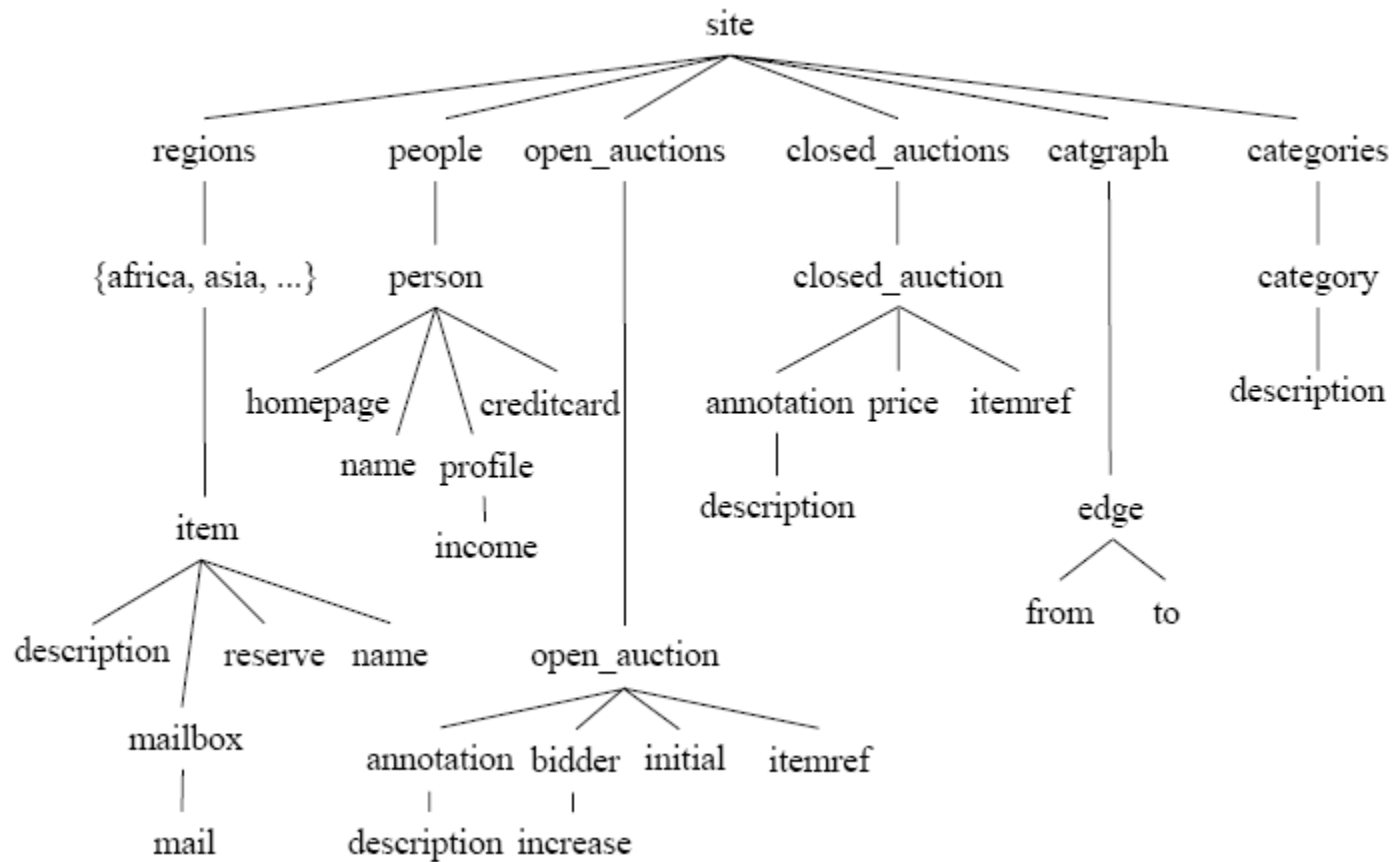
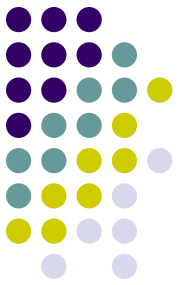


XMark

- The most popular benchmark
 - Simple
- 1 DTD – Internet auction
 - Text content: 17 000 most often words from Shakespeare's plays
- 20 XQuery queries
- 1 XML document + generator
 - Parameter: data size
 - Default: 100MB

<http://www.xml-benchmark.org/>

XMark – Elements



	XMark	XOO7	XMach-1	MBench	XBench	XPathMark	TPoX
Type	Application	Application	Application	Micro	Application	Application	Application
# users	1	1	More	1	1	1	More
# applications	1	1	1	1	4	1	1 complex
# documents	1	1	More	1	1/ More	1	More
XML schema	DTD of an Internet auction	DTD derived from a relational schema	DTD of a document with chapters, sections and paragraphs	DTD / XSD of a recursive element	DTD / XSD	DTD	XSD
# schemas	1	1	More	9	1	2	1 consisting of more
Data generator	yes	yes	yes	yes	yes	yes	yes
Key data parameters	Size	Depth, fan out, length of textual data	Number of documents, elements, words	Size	Size	Size	Size + number of users
Default data	One 100MB document	3 documents (small, middle, large) with pre-defined parameters	4 collections of 10 000 / 100 000 / 1 000 000 / 1 000 000 documents	One document with 728 000 nodes	Small (10MB) / normal (100MB) / big (1GB) / huge (10GB) document	1 XMark document a 1 sample document (books in a library)	XS (3.6 millions of documents, 10 users), S, M, L, XL, XXL (360 billions of documents, 1 million of users)
# queries	20	23	8	49	19,17,14,16	47 + 12	7
Query language	XQuery	XQuery	XQuery	SQL, XPath	XQuery	XPath	XQuery
# updates	0	0	3	7	0	0	10

Comparison of Benchmarks (1)



- Type
 - **Application** – comparison of various applications \Rightarrow queries differ a lot
 - Most benchmarks
 - **Micro** – efficiency of one application in various use cases \Rightarrow similar queries, differ, e.g., in selectivity
 - MBench
- Simulated situation
 - Number of users, applications, documents
 - Typically: 1 user, 1 application, 1 document
 - XBench – 4 classes of applications
 - Document/data-oriented with a single/multiple document(s)
 - XMach-1, TPoX – multi-user \Rightarrow test other aspects
 - Concurrent access, transactions, network characteristics, ...

Comparison of Benchmarks(2)



- Data sets
 - All benchmarks have DTD/XSD + simple generator
 - Typical parameter: size of data
- Operations
 - Except for XPathMark all involve XQuery queries
 - Some benchmarks involve also general description of queries \Rightarrow can be expressed in any language
 - XMach-1, MBench, TPoX – update operations
 - XMach-1, TPoX (multi-user) \Rightarrow also non-XML operations



Other XML Technologies

- Benchmarking of basic XML operations solved
 - Parsing, validation, querying
- What about other?
 - Transformation, compression, updating, new versions of query languages, ...
 - Currently: No or obsolete benchmarks
- Example 1. **XSLTMark**
 - From 2000, no update \Rightarrow XSLT 1.0
- Example 2: **XQuery Update benchmark**
 - Not much supported \Rightarrow not much tested
- Query: Do we need new benchmarks?
 - NO: We can test basic XML technologies from which others result
 - YES: Their exploitation can vary, only a subset/an extension is used within other technologies, ...
 - e.g., XPath within XSLT

References



1. A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask (Extended Abstract). In Selected papers from the 3rd International Workshop WebDB 2000 on The World Wide Web and Databases, pages 171-183, London, UK, 2001. Springer-Verlag.
2. B. Choi. What are real DTDs like? In *WebDB '02, Proceedings of the 5th International Workshop on the Web and Databases*, pages 43-48, Madison, Wisconsin, USA, 2002. ACM Press.
3. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a Practical Study. In *WebDB '04, Proceedings of the 7th International Workshop on the Web and Databases*, pages 79-84, New York, NY, USA, 2004. ACM Press.
4. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW '03, Proceedings of the 12th international conference on World Wide Web*, Volume 2, pages 500-510, New York, NY, USA, 2003. ACM Press.
5. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD'06: Proc. of the 13th Int. Conf. on Management of Data*, pages 20-31, New Delhi, India, 2006. Tata McGraw-Hill Publishing Co. Ltd.