Change Management

Versions and Versioning

XML and Data on the Internet

XML
Semi-Structured Data
Querying XML
Structured Data
Data Interchange

Based upon slides by Earl and Denise Ecklund
for INF312, IfI/UiO, høsten 2002

Overview

Change Management

- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet

- XML
- Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange
Applications Domains

- Design applications
  - electronic design (eCAD)
  - mechanical design (mCAD)
  - manufacturing (CAM)
  - software engineering (CASE)
  - technical documentation
  - ...

- Temporal applications
  - business & commerce
  - science
  - statistics
  - ...

Versions in Engineering

- Versioned Objects
- Workspaces
- Collaborative Engineering
- Parts Libraries
- Configurations

Architecture for design and process management
Workspaces and Developers

- Versioned Objects
  - Updates are non-destructive
    - Check-out/check-in transactions
    - "update" creates a new version (successor)
  - Currency (current versions)
  - Structural objects and versions

- Workspaces
  - A place to check out a group of objects
  - Private, team, group, development, released, project, ...
  - Hierarchical checkout, nearest version visibility
  - Active workspace

- Collaborative Development
  - Sharing objects within a workspace
  - Multiple concurrent writers (?)!!

Libraries and Configurations

- Parts Libraries
  - Library parts are versioned objects
    - Functionally equivalent variants
    - Context-dependent version selection rules
      - Path search
      - Working context rules
        » Work group
        » Work space

- Configurations
  - Compound objects
    - Aggregation of components (part-of)
    - Part specifies version selection rules (make)
  - Bound Configurations
    - A specific version is identified for each component
  - Configurations may themselves be versioned
System Requirements for Versioning

- Versioning is applied to objects (seldom to classes).

- All persistent objects can have versions and there should (in theory) be no predefined limit on the number of versions an object should have.

- Applications should be able to access either the current version (which should be the default) or any specified version. Only the current version of an object can be updated. Access to older versions should normally be read-only.

Overview

Change Management
- Overview
- Notions
  - Schema Versioning
  - Extension Versioning
  - Implementation Concepts

XML and Data on the Internet
- XML
  - Semi-Structured Data
  - Structured Data
  - Querying XML Data
  - Data Interchange
Notions

- Transaction time
- Schema evolution & schema versioning
- Extension versioning
- Version:
  - revision
  - alternative
  - variant
  - representation
  - configuration

Transaction Time

- 3 kinds of time:
  - User-defined time:
    - semantics of time only known by the user.
  - Valid time:
    - the time a fact was true in reality.
  - Transaction time:
    - the time the fact was stored in the database.

- Valid time and transaction time are supported by the DBMS.

- Considering transaction time: an important distinction
  - Extension versioning
  - Schema versioning
Overview

Change Management
- Overview
- Notions
  - Schema Versioning
  - Extension Versioning
  - Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange

Schema Evolution

- Schema can change over time in response to the varying needs of the application.

- 2 ways to implement schema evolution:
  - eager: all instances are changed immediately according to the schema changes made.

  - lazy: when schema is modified, no disk-resident instances need to be updated. Instead, when an instance is referenced by an application program and fetched into main memory, it is transformed into an instance conforming to the schema currently in effect.
Schema Changes

- **Change definition of class/type:**
  - add/change/drop an attribute/relationship
  - add/change/drop a method
  - inherit different attribute/method definition through changes in the structure of the class/type hierarchy

- **Change structure of the class/type hierarchy:**
  - add/change/drop class/type
  - make a class/type S a new superclass/type of a class/type C
  - remove a class/type S as a superclass/type of a class/type C
  - change inheritance ordering (only for multiple inheritance)

Schema Versioning

- Multiple schemata logically in effect.

- **3 general approaches:**
  - Entire schema is timestamped with a transaction time.
  - Specifying the system catalog as transaction time relations.
  - Viewing the entire schema as versioned object with status information and version-derivation hierarchy.
Overview

Change Management
- Overview
- Notions
- Schema Versioning
  - Extension Versioning
  - Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange

Extension Versioning

- 3 general approaches:
  - Use model directly (no changes to data model and query language).
  - General extensions to data model and query language to support time-varying information.
  - Data model and query language are modified to explicitly support transaction time.
Component Hierarchies (PART_OF) - Aggregation

- Associated with each design object are primitives of its type, such as layout geometries, logic schematics, or functional descriptions.

- A particular representation of a complete design is a component hierarchy of representation objects rooted at the object that describes the top level of the design.

```
Datapath.Layout
  ALU.Layout
  Shifter.Layout
  RFile.Layout
```

Version Histories (IS_A, DERIVED_FROM)

- Incorporating the time dimension into the representation of design objects by adding a version number to the name and type associated with each object.

- Version histories explicitly record the ancestor/descendent interrelationships among versions through distinguished DERIVED_FROM relationships.

```
ALU[1].Layout
ALU[2].Layout
ALU[3].Layout
ALU[4].Layout
```

Derivation
Versions

- Explicit representation of timely/causal history of objects

Example: low-polluting car

Body: B1
Drive: 4WD
Catalyst: ✓
Consumption: 101

Body: B1
Drive: 4WD
Catalyst: —
Consumption: 71

Body: B1
Drive: 4WD
Catalyst: ✓
Consumption: 81

Generic information common to all versions
+ specific information for each version

```
Generic information:
Body: B1
Drive: 4WD

Catalyst: ✓
Consumption: 101

Catalyst: ✓
Consumption: 81

Catalyst: —
Consumption: 71
```
Kinds of Versions

- **Revisions**
  - Explicit representation of different object states resulting from subsequent modifications

- **Alternatives**
  - Different approaches/ways for the development of an object

Kinds of Versions - 2

- **Variants**
  - Versions that are equal under a specific abstraction, distinguishable through certain parameters

- **Representations (equivalences)**
  - Different views of the same object
Kinds of Versions - 3

- Configurations
  - Describe the composition and structure of compound objects
  - The object is composed of selected versions of its components

Currency

- It is frequently the case that the default or current version is not the most recently created version.
  - It is useful to separate current and last and to make currency update explicit.
  - Operations are needed to position currency explicitly anywhere within the version history.
    - Including forking new update branches from the current version when it is not a leaf node in the version tree!
Dynamic Configurations

- It is useful to be able to create dynamic configurations: configurations in which references to components are not resolved until the PART_OF relationships are actually traversed.

- Variety of implementation approaches

```
```

Workspaces

- Workspaces are named repositories for design objects.

- 3 kinds of workspaces:
  - Archive
  - Group
  - Private

```
ARCHIVE

GROUP
Check-out

PRIVATE
```

```

V[5] Check-in

V[5]
V[5]
```
Other Currency Labels

- What is the "current" version may depend on the context from which it is referenced or how it is to be used
  - The current version in:
    - product manufacturing
    - new product development
    - research and development
- Labeled derivation paths: mfg, dev, r&d, ...
- A particular path may be used in a particular workspace

Logical Versus Physical Representations

Implementation of version model relationships
Change/Constraint Propagation

- **Change propagation**: process that automatically incorporates new versions into configurations.

- **2 key issues**:
  - How to limit the scope of propagation?
  - How to disambiguate the path of changes?

- **Constraint propagation**: enforcement of equivalence constraints be procedurally regenerating new versions.

---

Group Check-In / Group Check-Out

- Check-in \((B_1, C_1)\)
- Group Check-in versus Individual Check-in
- Check-out \((C_1)\); Check-out \((E_1)\)
- Check-out \((C_1, E_1)\)

---

Individual Check-out versus Group Check-out
Change/Constraint Propagation - 2

Logical

Physical

(a) Initial Configuration

Con A0

Eq

Con C0

Eq

Con B0

Eq

Con E0

Eq

Con D0

Eq

(b) Check-out B0 to create B1

(c) Propagate change to A1

(d) Spawn E1

(e) Propagate change to D1

(f) Final logical view

Inheritance

- Mechanism to model versions:
  Allows defining default operations and values for new versions

Problems with type-instance inheritance
Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
  - Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange

Realization of Versions

- Database Systems support the management of version graphs
  - part of the data model
    - similar to subobjects
      - existence of versions depend on existence of generic object
    - realized through references or relationships
      - explicit relationships "version_of" and "has_version"
    - operations for navigation in version graphs
  - no pre-defined semantics of "version"
    (e.g., "alternative" or "revision")
  - by means of class hierarchies

Further Functionality
- merging versions
- change notification
- version percolation
Approaches for Versioning in OODBS

- As part of the data model (different approaches)

object type CELL
  attributes  cell_id: string(20),
  cell_name: string(20)
  versions  treelike
  (attributes cdata: ...
    structure is PATH)
end CELL;

object type PATH
 ...
...
end PATH;

Approaches for Versioning in OODBS - 2

- With help of the class/type hierarchy
  -> versions as subobjects
Approaches for Versioning in OODBs - 3

• By predefined classes (using Inheritance)
Mechanism to model versions:
Version.<operation>
and de-referencing

Also allows defining
default operations
and default values
for new versions

Approaches for Versioning in OODBs - 4

• “Version percolation”
Approaches for Versioning in OODBS - 5

- **Intension**

```
interface versionable
    current()
    current4(label)
    version(#)
    ...
```

```
class layout
```

```
interface version
    derivedFrom()
    version(#)
    ...
```

```
class structural layout
```

```
class version of layout
```

Approaches for Versioning in OODBS - 5

- **Extension**

```
 ALU
   |
   v
 ALU[0]
```

```
 ALU[1]
   |
   v
 ALU[2]
```

```
 ALU[2]
```

```
 ALU[3]
```

```
 ALU[4]
```

```
 ALU[5]
```

```
Current Labeled Version
```

```
Current Version
```

```
mfg
dev
r&d
```
Change Management Summary

- semantics of time domain important for many applications

- significant amount of research has been expended on temporal data models and versioning concepts

- various versioning concepts, temporal data models and query languages have been proposed
  - "If we have three versions of a chip, how many instances of type CHIP are we talking about? Reasonable answers include one, three, and four." -- W. Kent

- nearly all OODBs provide versioning concepts

Versioning Literature


Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet
- XML
  - Semi-Structured Data
  - Structured Data
  - Querying XML Data
  - Data Interchange

XML
Extensible Markup Language, World Wide Web Consortium (W3C)

- Standard of the W3C in 1998
- Documents (content)
- XML Schema or DTD (structure)
- Namespaces
- XSL, XSLT (stylesheets, transformations)
- XPath, XPointer, XLink
- XQuery

- Tagged elements with attributes
  - Extensible, self-describing
- Tag names must be unique (use namespaces)
Namespaces

- All tags must be defined in a namespace
  - Unique global names using URI references

- W3C provides certain standard namespaces (xsd, xsi, html)
- Organizations can provide domain specific namespaces

- Each document specifies the namespaces used in its schema
  - A default namespace - a standard or domain namespace
  - Its own target namespace - for elements defined in document
  - Others ...

- Referencing a name (syntax)
  - Qualified names (namespace:name) xsd:element
  - name (default namespace implied) car

Elements

- Element declarations (see XML Schema)
  `<xsd:element name="numStudents" type="xsd:positiveInteger"/>

- Element definitions
  - start tag `<tagName>` must be paired with an
  - end tag `</tagName>`
  - `<emptyElement/>
    `<numStudents>31</numStudents>

- Sub-elements may be repeated
  - Subject to minOccurs and maxOccurs constraints (XML Schema)
    - Both default to 1 if not specified

- References
  `<xsd.element name="foo", type="xsd.string"/>
  ...
  `<xsd.element ref="foo"/>
Attributes, Entities

- Attributes (of an element)
  - Name-value pairs placed in the start tags
    - name = "value"
    - Each name may only appear once (single-valued)

  `<elevation units="feet">5440</elevation>`

- Entity
  - `&mdash;` an entity representing the character ‘—’
  - starts with & and ends with ;
  - Used to distinguish reserved words from content
    `&lt;` represents the character ‘<’ (also &gt; &amp; &apos; &quot;)
  - `<!ENTITY myPic SYSTEM "me.jpg" NDATA JPEGFORMAT>`

XML Document

```xml
<?xml version="1.0" ?>
<racingTeams xmlns="http://www.cart.org"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
               xsi:schemaLocation="http://www.cart.org team.xsd">
  <team>
    <name>"Shell"
    <car number=3><make>"Helix Ford"</make>
    <engine id="47456"></car>
    <car number=4><make>"Helix Ford"</make>…</car>
    <driver><name>"Steve Johnson"
    <points>2217</points></driver>
    <driver><name>"Paul Radisich"
    <points>2217</points></driver>
    <spares><engine id="102555"/>
    <engine id="102556"/></spares>
  </team>
  ...
</racingTeams>
```

<!-- Each document has one root element (e.g., racingTeams) whose name matches a schema element or <!DOCTYPE name> -->
XML in Action

1. Export: XML document written to file or stream
2. Delivery: Passed to import software
3. Import: Parse XML document, Validate using DTD or Schema (optional)

Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet
- XML
  - Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange
Semi-Structured Data

- Semi-structured data is data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably.
  - It generally has some structure, but does not conform to a fixed schema
  - Self-describing (e.g., in terms of XML element tags)

- Characteristics
  - Heterogeneous
  - Irregular structure
  - Large evolving schema
  - Hyperdata nodes and references

- Desirable to treat web sources like a database
  - Often XML documents are like semi-structured data

Semi-Structured Example

```
DreamHome (&1)
Branch (&2)
  street (&7) “22 Deer Rd”
  Manager &3
Staff (&3)
  name (&8)
    fName (&17) “John”
    lName (&18) “White”
  ManagerOf &2
Staff (&4)
  name (&9) “Ann Beech”
  salary (&10) 12000
  Oversees &5
  Oversees &6

PropertyForRent (&5)
  street (&11) “2 Manor Rd”
  type (&12) “Flat”
  monthlyRent (&13) 375
  OverseenBy &4

PropertyForRent (&6)
  street (&14) “18 Dale Rd”
  type (&15) 1
  annualRent (&16) 7200
  OverseenBy &4
```
Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
  - Structured Data
- Querying XML Data
- Data Interchange

XML Schema
- Purpose of XML Schema, to declare
  - The organization of instance documents
  - The datatype of each element/attribute
    (New element types go in targetNamespace)
- Features
  - Same syntax as XML
  - Integrated with namespace mechanism
    - xsd is standard default namespace for schemata
    - Included schema must have same targetNamespace
  - Rich set of built-in types (44+ data types)
  - Elements have nested scope (scoped naming)
  - Keys definable in terms of elements
  - Referential integrity constraints enforced with keys
  - Options for ordered (sequences) or unordered (sets)
  - Validation of documents against a schema
XML Schema Grammar

- Namespace
- Compositors
  - `<sequence>`, `<choice>`, `<all>`, `<union>`, `<list>`
- complexType, simpleType
-complexContent, simpleContent
- restriction, extension, enumeration
- ID, IDREF [List(IDREF) replaces IDREFS ]
- unique like a candidate key, whenever present => unique
- key minOccurs > 0, nillable = “false”
  - selector collection to which key applies (like Table)
  - field path expressions relative to selector collection
- keyref specifies foreign key constraint
- Empty elements

XML Schema Example

```xml
<?xml version="1.0"?><xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.binary.org"
  xmlns="http://www.binary.org">
  <xsd:element name="life">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="eat" type="xsd:string"/>
        </xsd:sequence>
        <xsd:choice>
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="play" type="xsd:string"/>
        </xsd:choice>
        <xsd:element name="sleep" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
life ::= ( (work, eat)*, [ work | play ], sleep )* 
```
XML Document

```xml
<?xml version="1.0" ?>
<racingTeams xmlns="http://www.cart.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
  xsi:schemaLocation="http://www.cart.org team.xsd">
  <team>
    <name>"Shell"
    <car number=3><make>"Helix Ford"</make><engine id="247456"></car>
    <car number=4><make>"Helix Ford"</make> <engine id="102456"></car>
    <driver><name>"Steve Johnson"
      <points>2217</points></driver>
    <driver><name>"Paul Radisich"
      <spares>
        <engine id="102555" />
        <engine id="102556" />
      </spares>
    </driver>
  </team>
  ...
</racingTeams>
```

<!-- Each document has one root element (e.g., racingTeams) whose name
matches a schema element or <!DOCTYPE name> -->

XML Schema Example 2

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.cart.org"
  xmlns="http://www.cart.org">
  <xsd:simpleType name="engine">
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:simpleType>
  <xsd:element name="team">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" />
        <xsd:element name="car" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="make" type="xsd:string" />
              <xsd:element name="engine" type="engine" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="driver" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:list itemType="engine"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
- Structured Data
  - Querying XML Data
  - Data Interchange

XPath

- View XML document as a “tree” of nodes and references (edges)
- XPath: sequence of steps that yields a set of nodes
  / step / step / step … step = basis + [predicates]
  (Like a unix path, but each step can be a query expression)
  / root node
   . current node
   .. parent node
   @attr access attribute “attr”
   [n] access the nth occurrence of a node at this level
   last() // descendent or self
text() text content of node
functions sum(), count(), …
[ . . . ] selection condition (predicate)

- Example:
  //student[ status="ok" and starts_with(.,/last, "E") and not (.//last = ./first)]
Xpointer

- Xpointer: an XPath expression within a URI +
  - points: (position within a document), or
  - ranges: (start-point, end-point)
  - xpointer(id('list37')/item)
  - xpointer(id('list37')/item[1]/range-to(following-sibling::item[2]))

- http://someURL#xpointer(XPath expression) (entities, escapes)
  - doc.xml#xpointer(string-range(//P,'a little hat ^^&quot;'))

- Nested elements from different namespaces (same local name)
  - xmlns(x=http://example.com/foo) xmlns(y=http://example.org/bar)
    xpointer(//x:a/y:a)

XQuery

- "The goal of the XML Query WG is to produce a data model for XML documents, a set of query operators on that data model, and a query language based on these query operators."

- Strongly influenced by OQL (+ SQL, XML-QL and XPath)

- XQuery is a functional language in which a query is represented as an expression

- XQuery expressions can be nested with full generality

- Filters can strip out fields, like relational project

- Grouping
XQuery Expressions

- Path expressions
  - XPath syntax + dereference operation (->) + range predicate
  - Path expression returns an ordered list of nodes
  - Start at document(string), /, or //
- Element constructors
- FLWR expressions
  - FOR
  - LET
  - WHERE
  - RETURN
- Expressions involving operators and functions
- Conditional expressions
- Quantified expressions
- List constructors
- Expressions that test or modify datatypes

FLWR

FOR var IN expr
LET var := expr
WHERE expr
RETURN expr

FOR and LET clauses generate a list of tuples of bound expressions, preserving document order.
WHERE clause applies a predicate, eliminating some of the tuples
RETURN clause is executed for each surviving tuple, generating an ordered list of outputs

- FOR iterates through a sequence of individual nodes out of the selected collection, in order, one at a time
- LET binds a variable to the set of nodes in the selected collection
- FOR or LET clauses are considered nested. Later clauses may reference variables bound in previous clauses
FLWR (2)

- WHERE clause is optional
  - Predicates using bound variables

- RETURN generates the output (constructor expression for output)
  - node | [ordered] forest of nodes | primitiveValue

  - The constructor is executed once for each tuple of bindings from FOR or LET clauses that satisfies the WHERE clause.
  - It preserves ordering where ordering exists
  - The RETURN expression often contains Element constructors
    - With references to bound variables and nested subelements
  - A list may be constructed by enclosing zero or more expressions in square brackets, separated by commas

XQuery Examples

Path expression example:
```
document("zoo.xml")/chapter[title = "Frogs"]//figref/@refid->fig/caption
```

FLWR expression example:
```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")/book[publisher = $p]/price)
RETURN
  <publisher>
    <name> $p/text() </name>
    <avgprice> $a </avgprice>
  </publisher>
```

Note: result(//) = result(.)+result(/)
Probably here / and // give the same result

Note: this “publisher” name is in the local scope of document("bib.xml") from the context of the path expression
And this “publisher” name must match the name of an element in the default namespace ☺
Overview

Change Management
- Overview
- Notions
- Schema Versioning
- Extension Versioning
- Implementation Concepts

XML and Data on the Internet
- XML
- Semi-Structured Data
- Structured Data
- Querying XML Data
- Data Interchange

XML Metadata Interchange

- XMI is an XML language to interchange the objects of software systems:
  - UML/MOF\(^1\) designs
  - Computer software classes and interfaces
  - Databases and database schemas
  - DTDs and XML schemas
- Added to the list of OMG adopted technologies in February, 1999 as XMI 1.0
- Most recent minor revision is XMI 1.1 (November 2000)

\(^1\) MOF = Meta-Object Facility, describing data about data, i.e., the classes and associations structure of the data
XMI: Sharing Objects

- XML - Sharing Data
- XMI - Sharing Objects
  - Creates custom
    - XML Schemas (DTDs)
    - XML documents
      from class definitions.

Application1  Application2  Key

Objects ↔  Objects ↔  XML
Data ↔  Data ↔  XML
Text ↔  Text ↔  Unicode

XMI Object Mapping

<table>
<thead>
<tr>
<th>MOF Concept</th>
<th>XML Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>XML element, odd levels</td>
</tr>
<tr>
<td>Object typed attribute</td>
<td>XML element, even levels</td>
</tr>
<tr>
<td>Containment</td>
<td>XML element, even levels</td>
</tr>
<tr>
<td>Reference</td>
<td>XML attribute IDREF</td>
</tr>
<tr>
<td>Basic attribute (int, String)</td>
<td>XML attribute CDATA</td>
</tr>
</tbody>
</table>
**XMI Mapping Example**

```java
Class Car {
    String color; // Field, String
    Person customer; // Field, Reference
    Engine standardEngine; // Field, Object
    Engine optionalEngine; // Field, Object
}
```

```xml
<Car color="Blue" customer="Joe">
    <Car.standardEngine>
        <Engine .... />
    </Car.standardEngine>
    <Car.optionalEngine>
        <Engine .... />
    </Car.optionalEngine>
</Car>
```

**Validation in XMI**

- MOF model (XMI Document) → Generated XML Schema
- Objects (XMI Document) → XML Parser
- Semantic Validation
- Syntactic Validation
- Well-formed
- Generate
- Optional

2003 INF 3180/4180, II/UO 71

2003 INF 3180/4180, II/UO 72
XML Schema: Specifies the Properties for a Class of Resources

```xml
<xsd:complexType name="Book">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:year"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

"For the class of Book resources, we identify five properties - Title, Author, Date, ISBN, and Publisher"
```

XML Instance Document: Specifies Values for the Properties

```xml
<Book>
  <Title>My Life and Times</Title>
  <Author>Paul McCartney</Author>
  <Date>July, 1998</Date>
  <Publisher>McMillin Publishing</Publisher>
</Book>

"For a specific instance of a Book resource, here are the values for the properties. Use schemaLocation to identify the companion document (i.e., the schema) which defines the Book class of resources."
Metadata Interchange using XML Schema

- XML Schema Strategy - two documents are used to provide metadata:
  - a schema document specifies the properties (metadata) for a class of resources (objects);
  - an instance document provides specific values for the properties of each object instance being transferred.

- Describing a schema for an object model using XML Schema requires another XML Schema that describes the object model’s metamodel!

- XMI uses an XML Schema for the UML metamodel 😊

XML Literature


W3C Extensible Markup Language Website (XML page and links to nearby pages) http://www.w3.org/XML/

David Maier, “Database Desiderata for an XML Query Language”
http://www.w3.org/TandS/QL/QL98/pp/maier.html

OMG XML Metadata Interchange (XMI) Specification Version 1.1,