Object Interchange with XMI

Stephen Brodsky

June 2000

Tutorial Series

- Introduction to UML
  - November 1999, Cambridge, US
- Behavioral Modeling with UML
  - January 2000, Mesa, Arizona, US
- Advanced Modeling with UML
  - March 2000, Denver, US
- Metadata Integration with UML, XMI and MOF
  - June 2000, Oslo, Norway

Tutorial Goals

- What you will learn:
  - what the UML is and what is it not
  - UML's basic constructs, rules and diagram techniques
  - how the UML can model large, complex systems
  - how the UML can specify systems in an implementation-independent manner
  - how UML, XMI and MOF can facilitate metadata integration
- What you will not learn:
  - Object Modeling
  - Development Methods or Processes
  - Metamodeling

XMI Quick Tour

- XMI is an XML language to interchange the objects of software systems:
  - UML/MOF 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 (February 2000)

XMI Goals

- Provide a general purpose object interchange
- Leverage XML technology
- Leverage UML/MOF technology
- Consistent look-and-feel
- Support data, meta-data, meta-meta-data, ...
- Extensible
- Scales from simple to enterprise

OMG XMI Evolution

- XMI 1.0
  - Uses MOF 1.1, XMI 1.0 (DTDs)
  - Finalized Oct 1998, adopted Feb 1999
  - Supported by 29+ companies
- XMI 1.1
  - Uses MOF 1.3
  - Adds XML Namespaces
  - Finalized Nov 1999, adopted Feb 2000
- XMI 2.0 - initial submission
  - Adds XML Schema
  - Adds Schema/DTD reverse engineering

XMI Topics

- XML
- XMI 1.1
- XMI Applications
  - Tools
  - Repositories
  - Messaging
- XMI for XML Schema

XML Overview

- Standard of the W3C in 1998
- Documents
- DTD
- Namespace
- APIs: DOM and SAX
- Others: XSLT, XPath, XLink
- Future: XML Schema

XML Example

- XML document

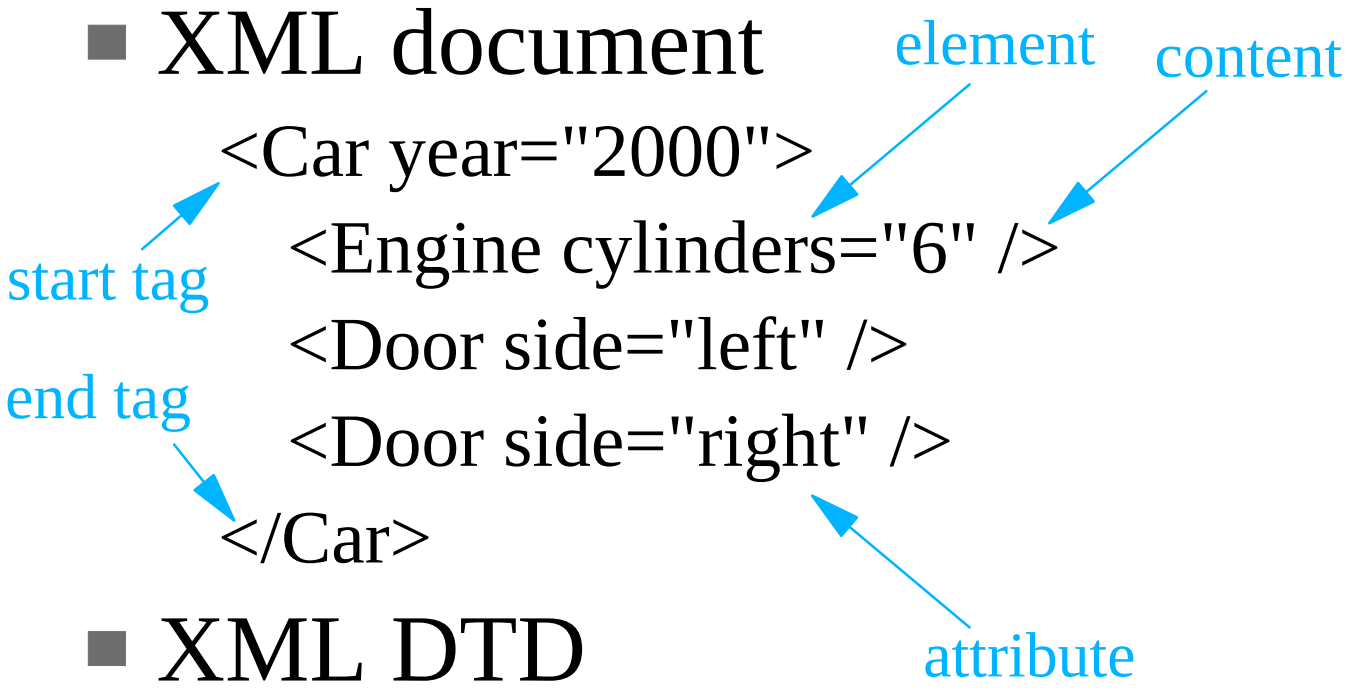
```
<Car year="2000">
    <Engine cylinders="6" />
    <Door side="left" />
    <Door side="right" />
</Car>
```

- XML DTD

```
<!ELEMENT Car (Engine, Door*)>
<!ATTLIST Car year CDATA #IMPLIED>
```

XML Example

- XML document




```
<Car year="2000">
    <Engine cylinders="6" />
    <Door side="left" />
    <Door side="right" />
</Car>
```

- XML DTD

```
<!ELEMENT Car (Engine, Door*)>
<!ATTLIST Car year CDATA #IMPLIED>
```

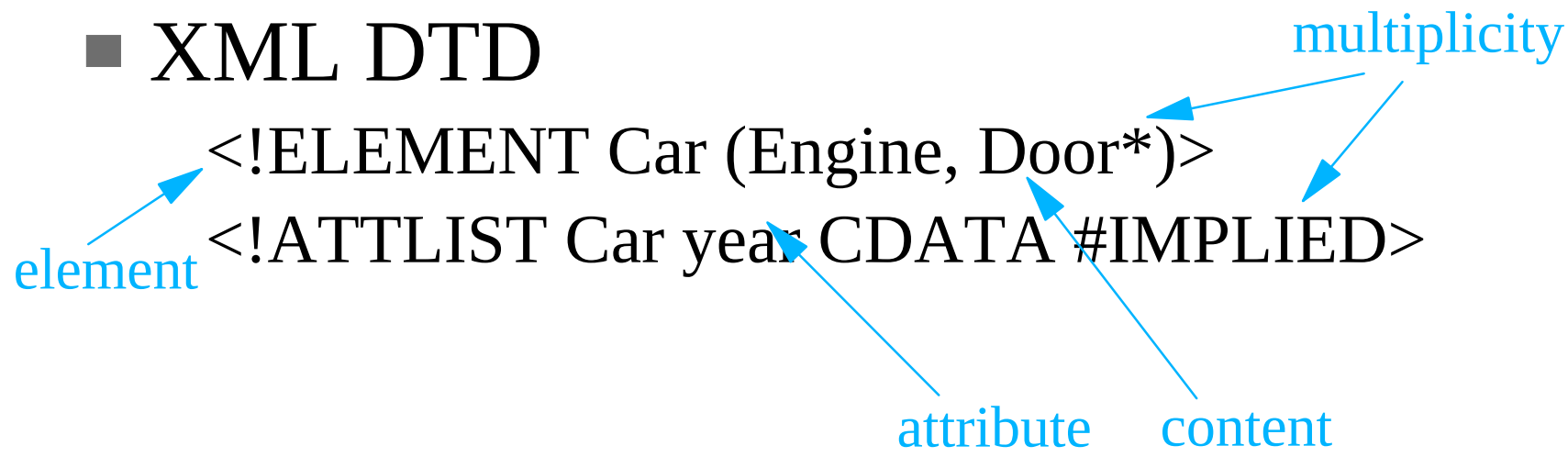
XML Example

- XML document

```
<Car year="2000">
    <Engine cylinders="6" />
    <Door side="left" />
    <Door side="right" />
</Car>
```

- XML DTD

```
<!ELEMENT Car (Engine, Door*)>
<!ATTLIST Car year CDATA #IMPLIED>
```



XML Example

- XML document

```
<Car year="2000">
    <Engine cylinders="6" />
    <Door side="left" />
    <Door side="right" />
</Car>
```

element
content
start tag
end tag
attribute

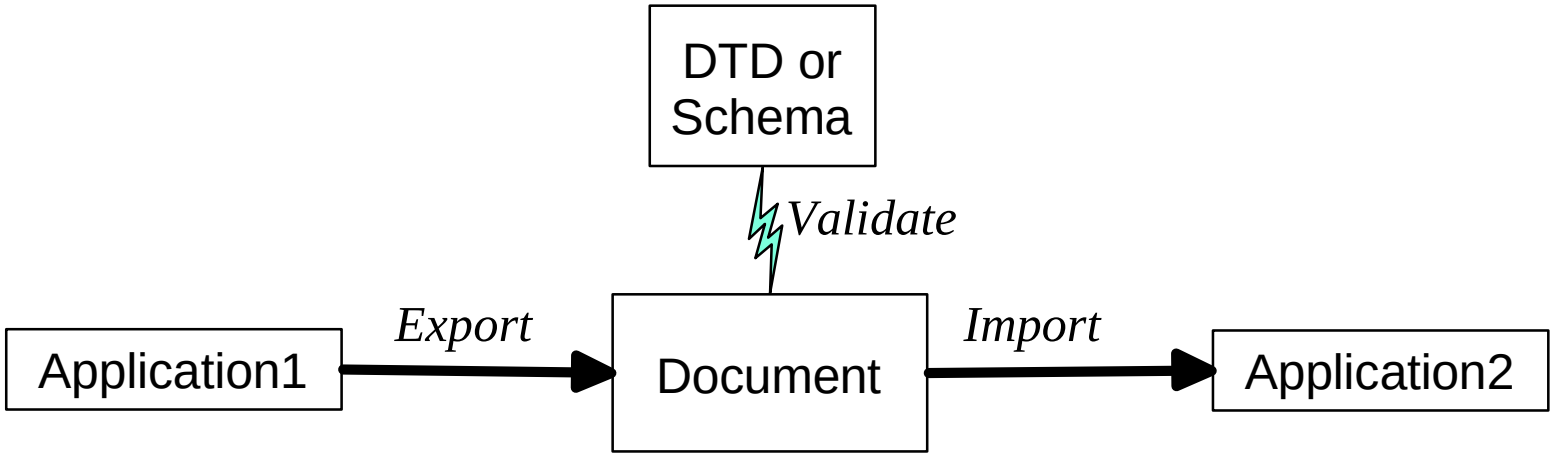
- XML DTD

```
<!ELEMENT Car (Engine, Door*)>
<!ATTLIST Car year CDATA #IMPLIED>
```

multiplicity
element
attribute
content

XML in Action

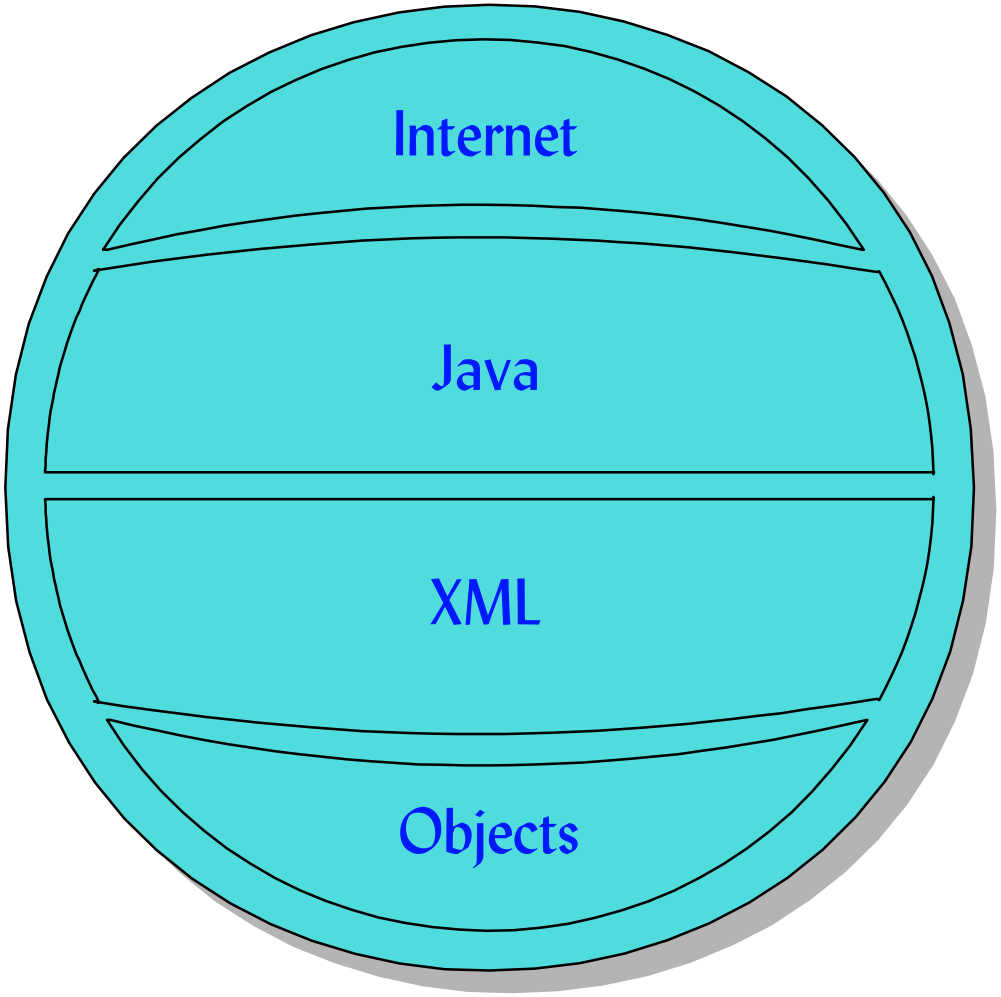
1. Export: XML document written to file or stream
2. Delivery: Passed to import software
3. Import: Parse XML document,
   - Validate using DTD or Schema *(optional)*



XMI Topics

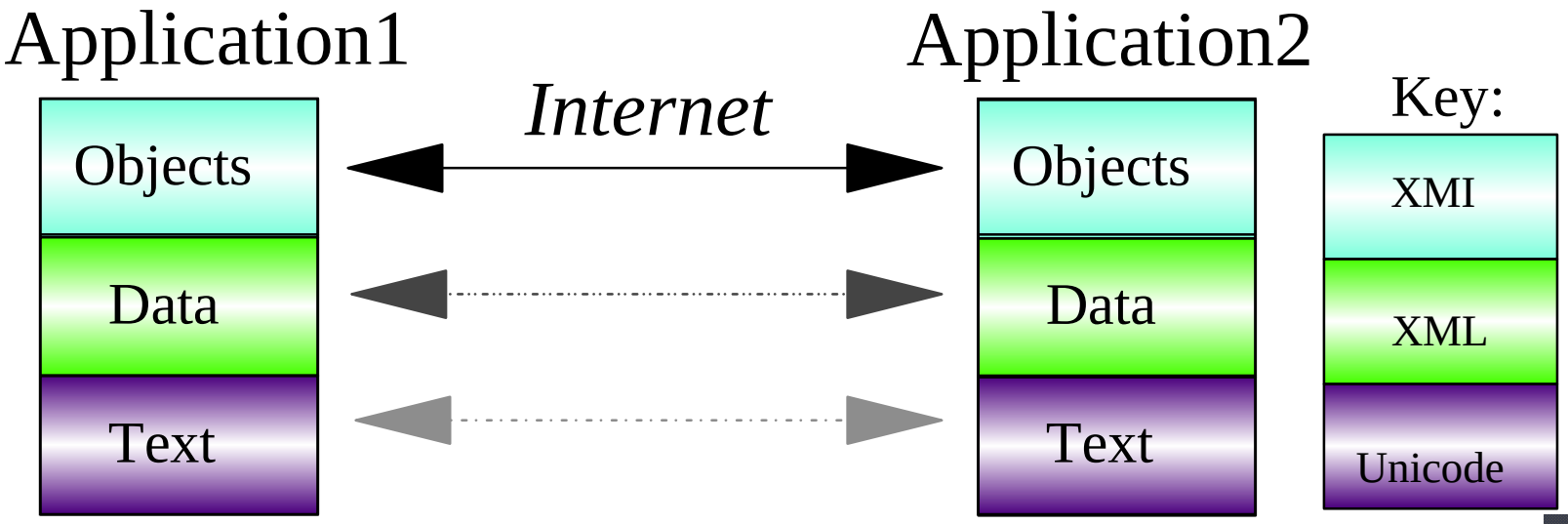
- XML
- XMI 1.1
- XMI Applications
  - Tools
  - Repositories
  - Messaging
- XMI for XML Schema

The Solution-Centric World



XMI: Sharing Objects

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



Concept-inspired XML

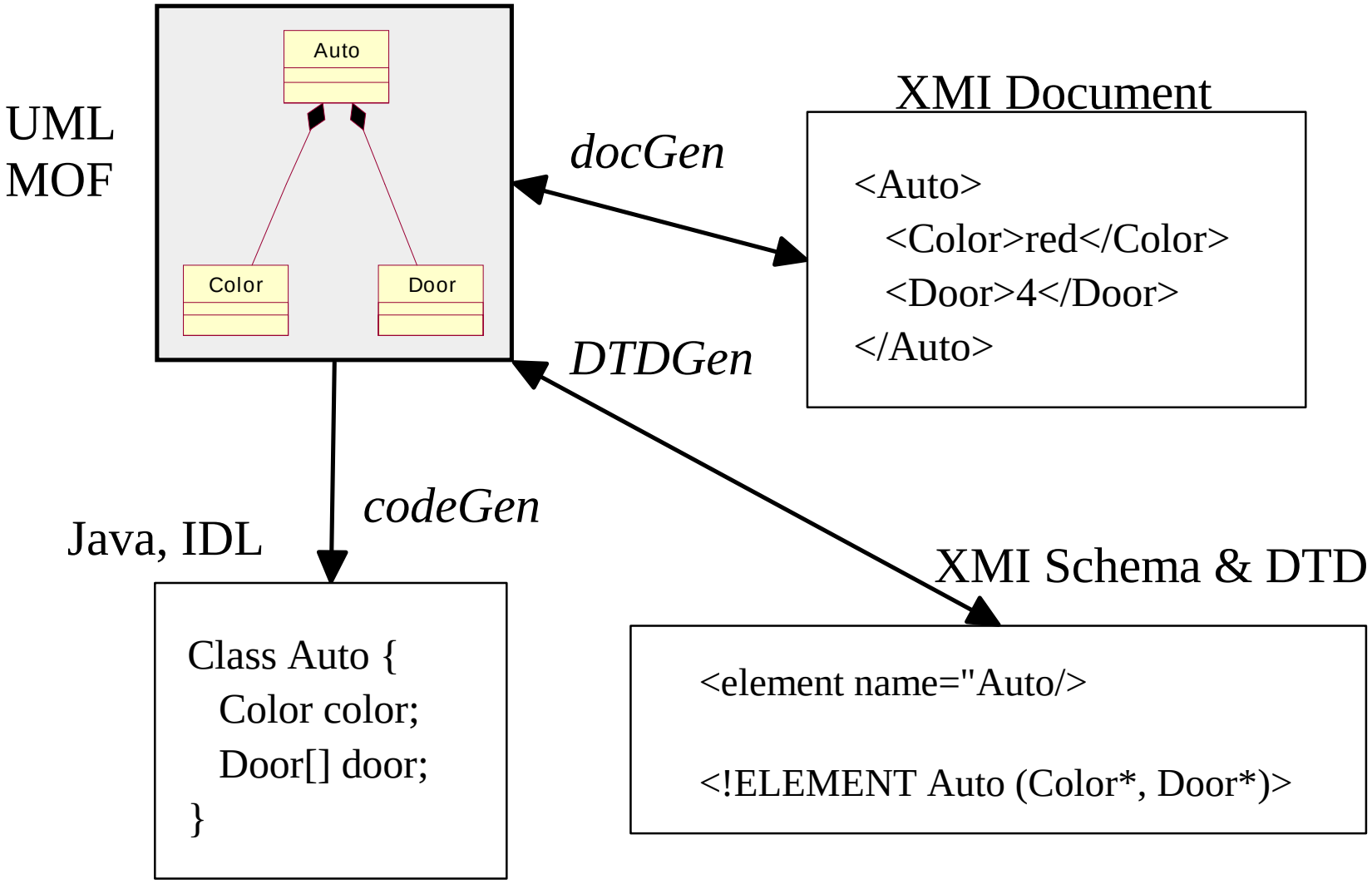
```
<Car>
  <doors>4</doors>
  <color>red</color>
</Car>
```

```
<auto doors="4" color="red"/>
```

```
Class Automobile {
  int numDoors;
  Color color;
}
```

```
<!ELEMENT Vehicle (door*)>
<!ATTLIST Vehicle
     Color CDATA #IMPLIED>
```

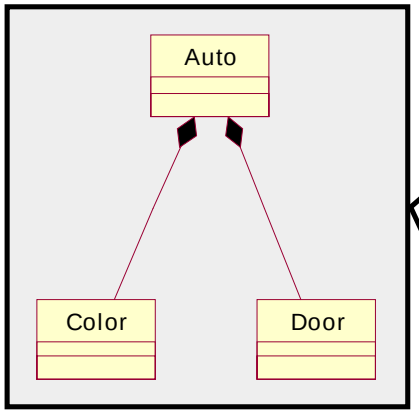
Metadata-driven XMI



Services of the XMI Toolkit

Design-driven XMI for a car

Objects and Designs



Model in XMI

```
<Class>
  <Name>Auto</Name>
</Class>
```

Model Interchange

XMI DTD, Schema

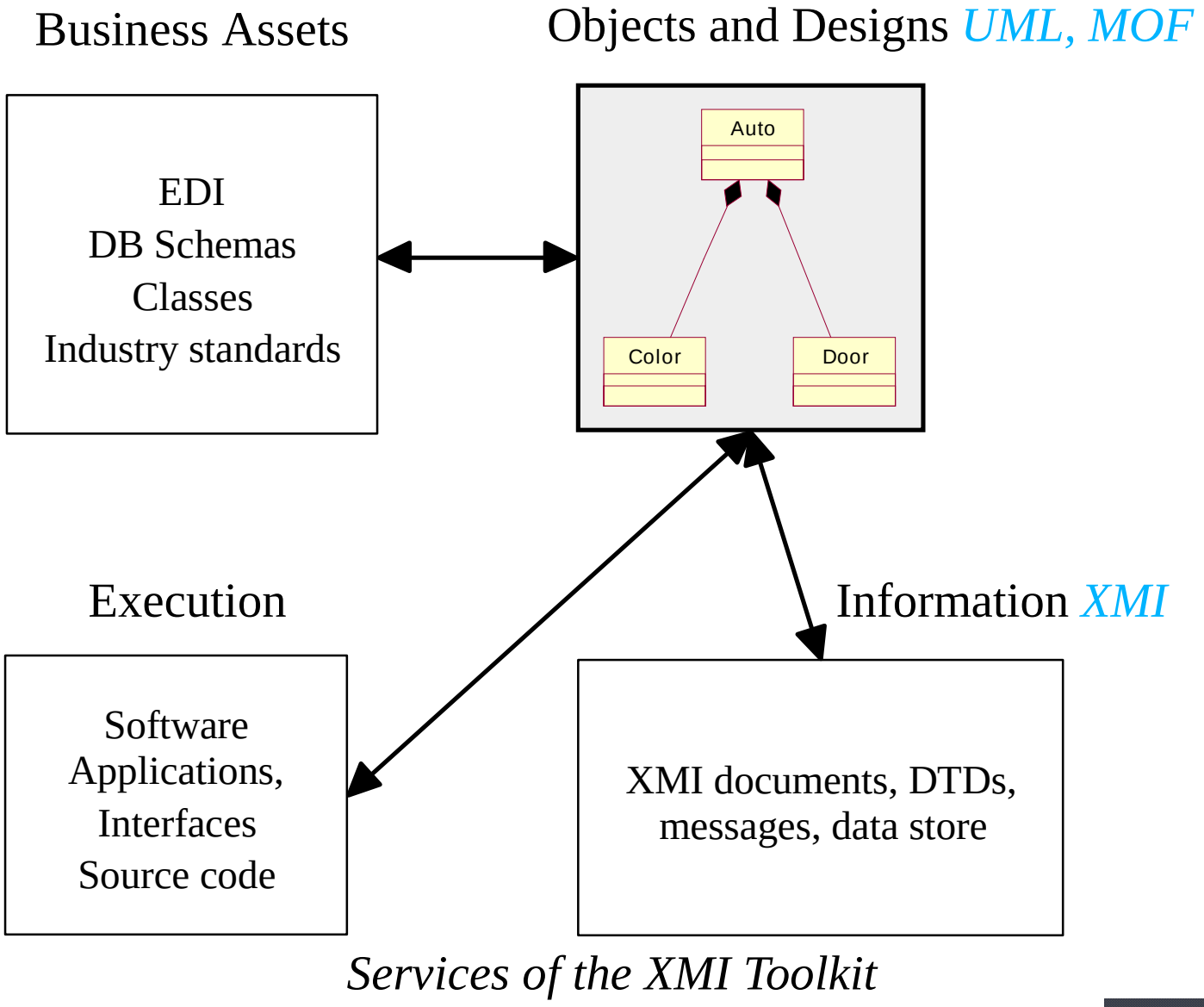
```
<element name="Auto" />
<!ELEMENT Auto (Color*, Door*)>
```

XMI Document

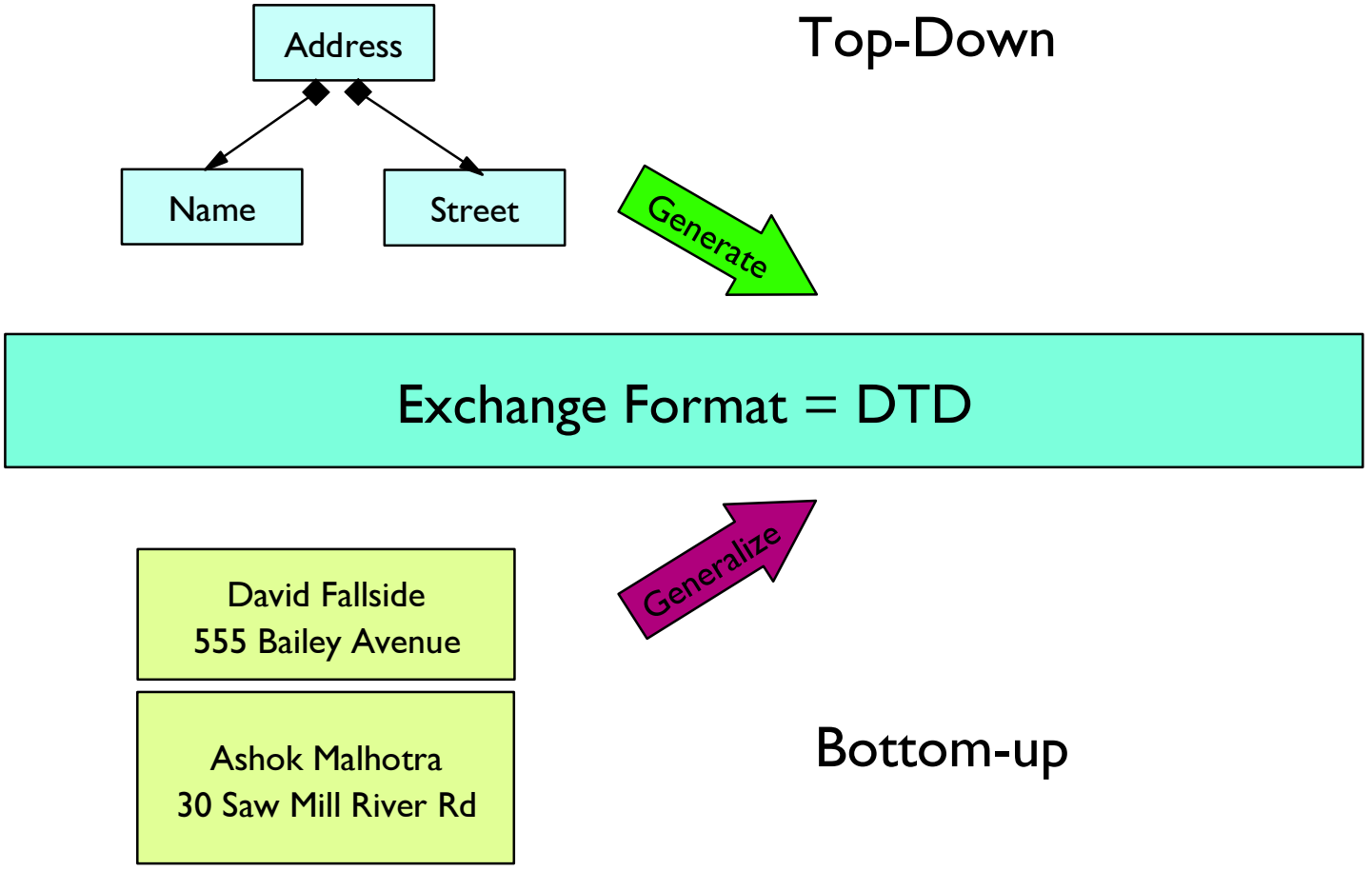
```
<Auto>
  <Color>red</Color>
  <Door>2</Door>
</Auto>
```

Instance Interchange

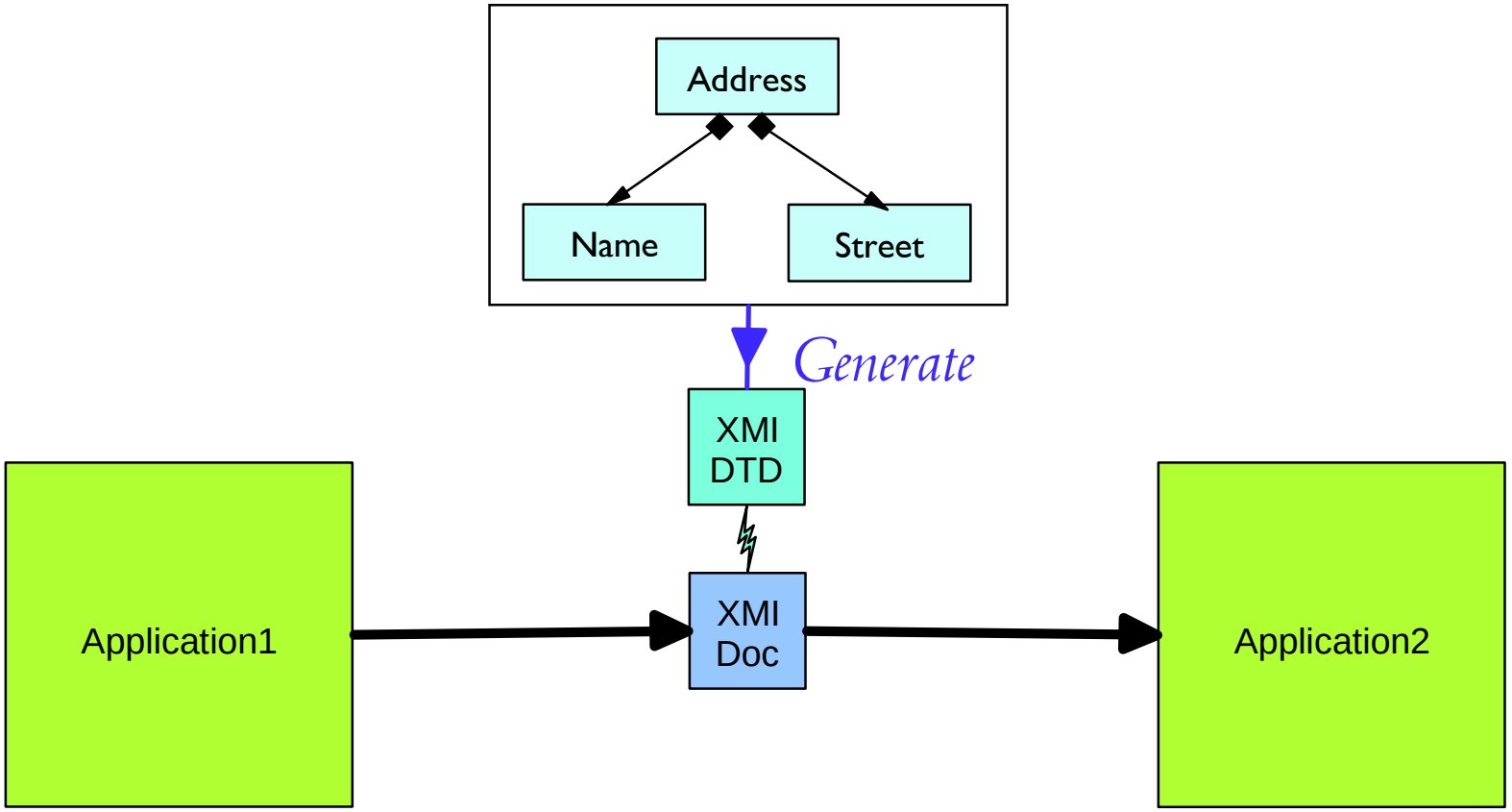
Business-driven Object Architecture



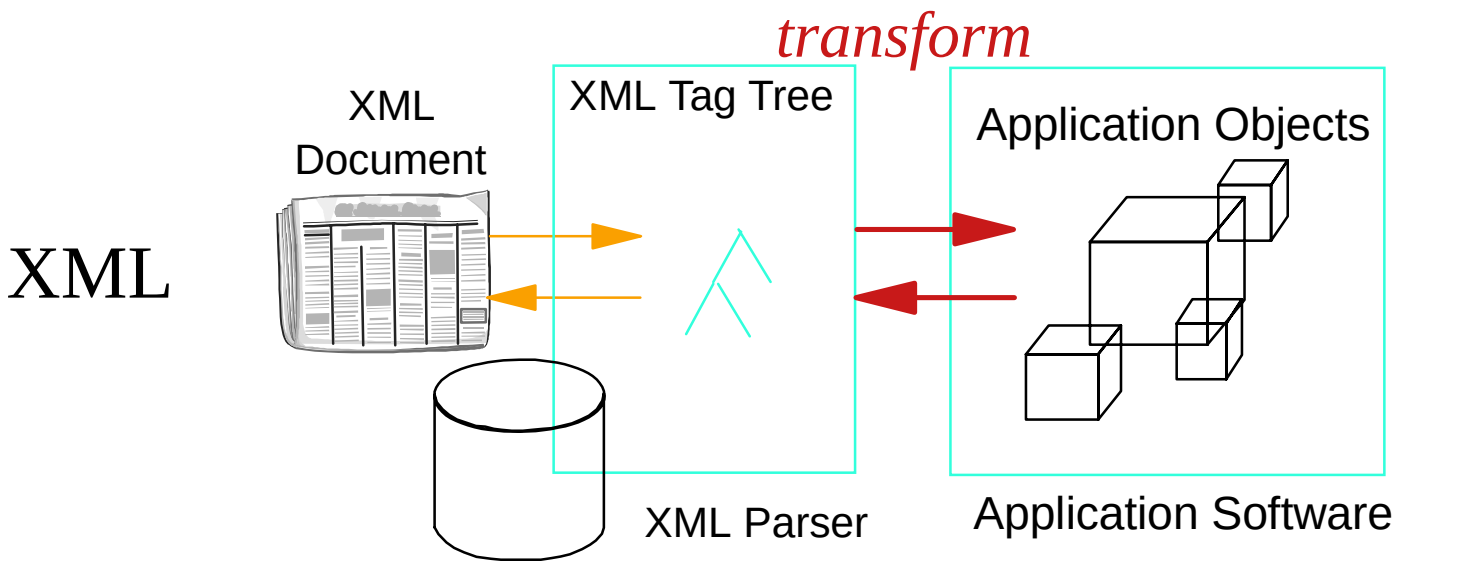
Approaches to interchange

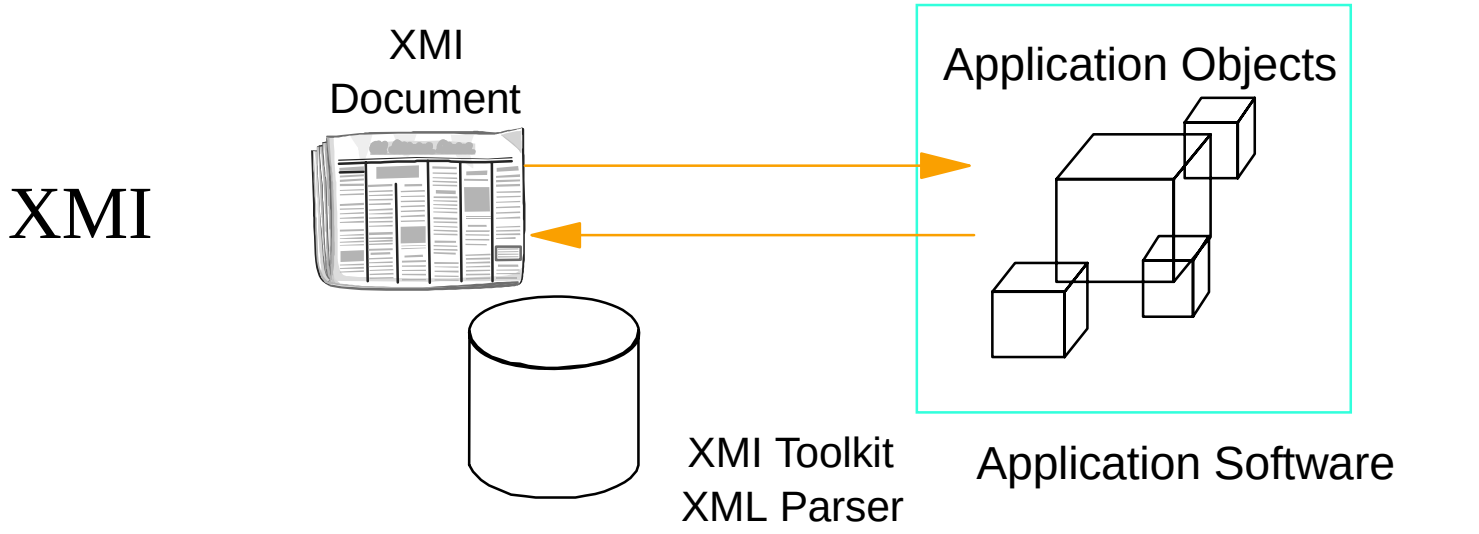


XMI Exchange Scenario

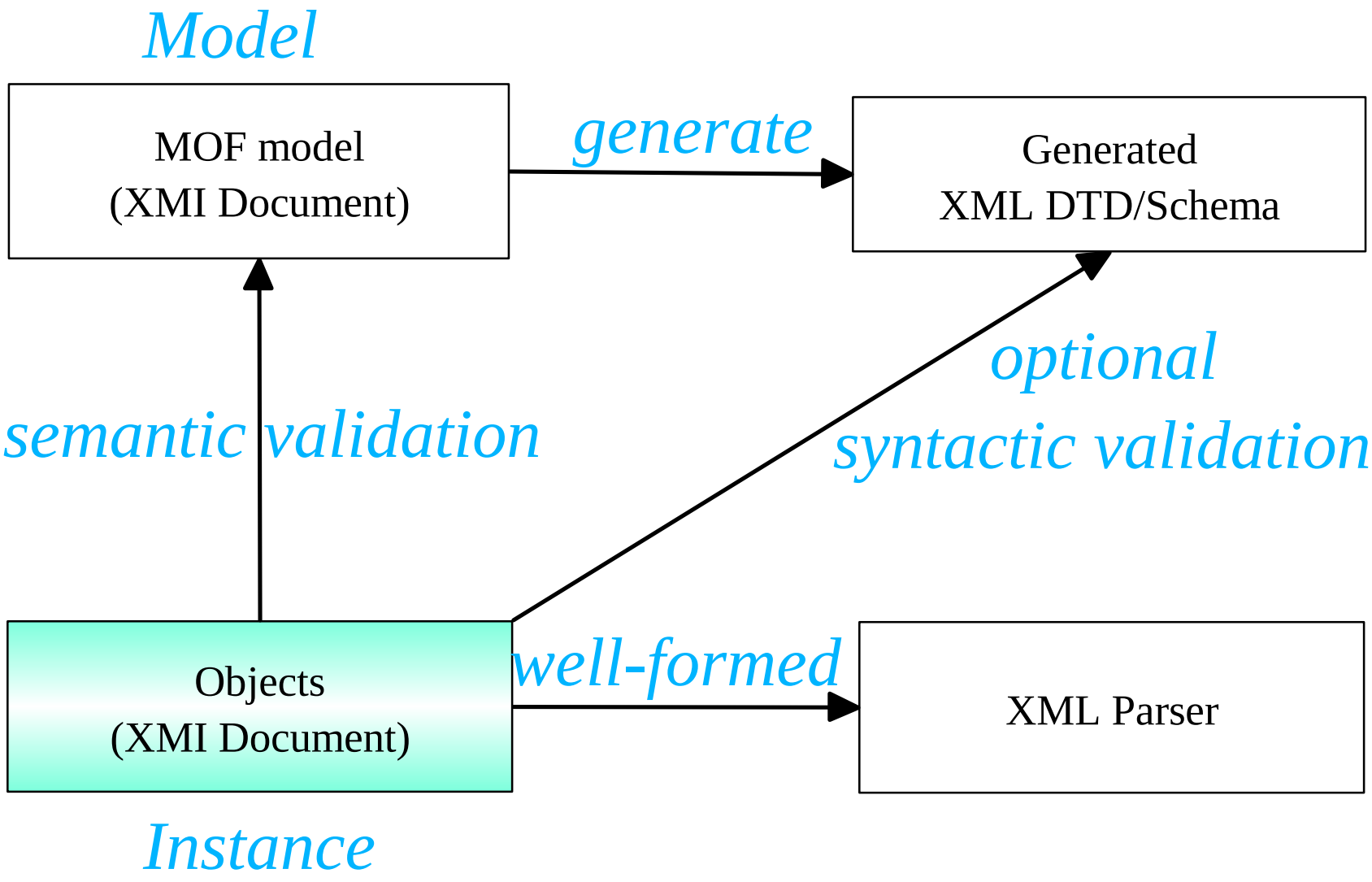


XMI Object Serialization





Validation in XMI



Validation

- Well-formed

  No: *15>LS<<y99>/4*

  Yes: *<Car />*

XML Parser

- Syntactic

  <!element Car (Door+)>

  No: *<Car />*

  Yes: *<Car><Door/></Car>*

XML DTD/Schema

- Semantic

  No: *Cow jumped over the moon*

  Yes: *Rocket flew to the moon*

  Yes: *<Car><Door/><Door/><Door/><Door/></Car>*

MOF model

XMI Object Mapping

MOF Concept	XML Concept
Class	XML element, odd levels
Object typed attribute	XML element, even levels
Containment	XML element, even levels
Reference	XML attribute IDREF
Basic attribute (int, String)	XML attribute CDATA

XMI Mapping Example

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

Class definition

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

XMI instance

XMI Mapping Example

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

- XML element, odd levels
- XML element, even levels
- XML attribute IDREF
- XML attribute CDATA

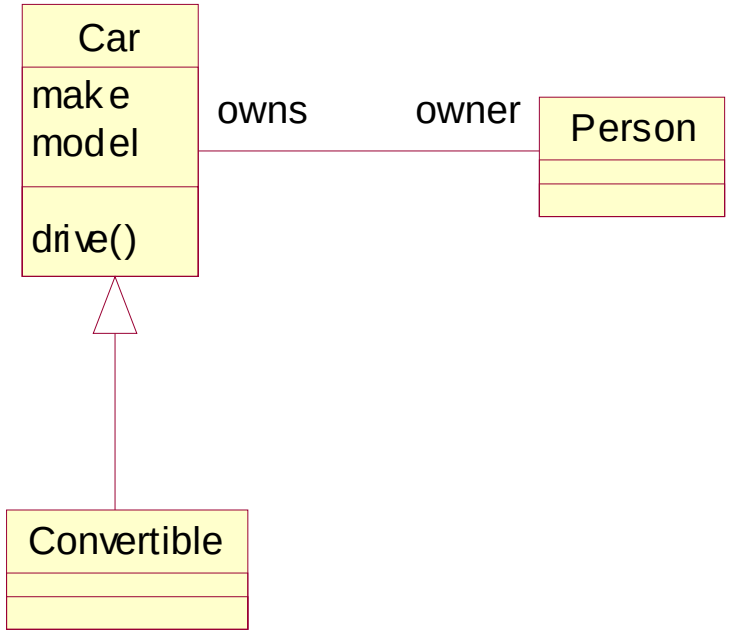
```
<Car color="Blue" customer="Joe">
    <Car.standardEngine>
        <Engine .....  />
    <Car.standardEngine>
    <Car.optionalEngine>
        <Engine .....  />
    </Car.optionalEngine>
</Car>
```

XMI Architecture

- Elements match information model
- Standardized element structure
  - Sub-elements, contents
  - Identity, Object ids
  - Linking
  - Extensions
  - Navigation, associations
- Standard transfer structure with header
- Differences (add, delete, replace)

XMI - example UML model

```
<Class name="Car">
    <Class.ownedElements>
        <Attribute name="make"/>
        <Attribute name="model"/>
        <Operation name="drive"/>
    </Class.ownedElements>
</Class>
```



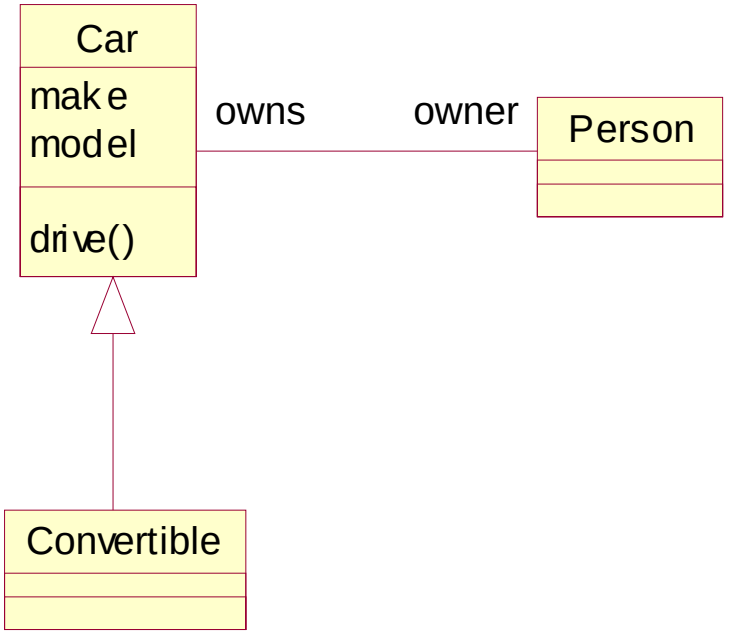
XMI - example of a Car

Car.xml

```
<Car make="Ford" model="Mustang"/>
```

Car.dtd

```
<!element Car>
<!attlist Car
    make cdata #implied
    model cdata #implied
>
```



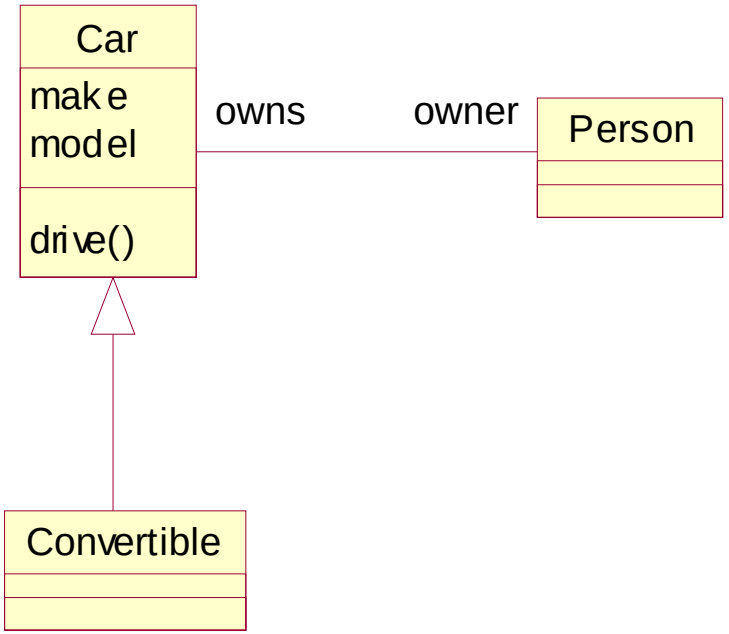
XMI - example, cont

Car.xml

```
<Car xmi.id="4ABC123" make="Ford" model="Mustang" owner="John Doe"/>
<Person xmi.id="John Doe" owns="4ABC123"/>
```

Car.dtd

```
<!element Car>
<!attlist Car
    xmi.id id #implied
    make cdata #implied
    model cdata #implied
    owner idref #implied>
<!element Person>
<!attlist Person
    xmi.id id #implied
    owns idref #implied>
```



XMI Example Document

header

content

metadata and version

```
<XMI xmi.version="1.1" xmlns:uml="org.omg/uml1.3">
  <XMI.header>
    <XMI.documentation>
      An example of an auto.
    </XMI.documentation>
    <XMI.metamodel name="UML" version="1.3"
      href="uml1.3.xmi" />
    <XMI.model name="Cars" version="1.0" />
  </XMI.header>
  <XMI.content>
    <Class name="Car">
      <Class.ownedElements>
        <Attribute name="make"/>
        <Attribute name="model"/>
        <Operation name="drive"/>
      </Class.ownedElements>
    </Class>
  </XMI.content>
</XMI>
```

XMI Identity and Linking

- Identity
  - id — unique within this document
  - uuid — unique across all documents
- Linking
  - idref — link to object within this document
  - href+uuid — link to object in any document

newCars.xml

```
<Car xmi:uuid="4JFK196" color="blue" year="2000"/>
```

carUser.xml

```
<Car xmi:uuid="4JFK196" href="http://cars.org/newCars.xml">
```

XMI extensions

- XMI.extension element
- Helps bridge the gap between systems when performing round-trip interchange
- Example: Two systems, each with their own id for an object

```
<Car xmi:uuid="4JFK196">
   <XMI.extension extender="NorwegDMV" extenderId="G166F97" />
</Car>
```

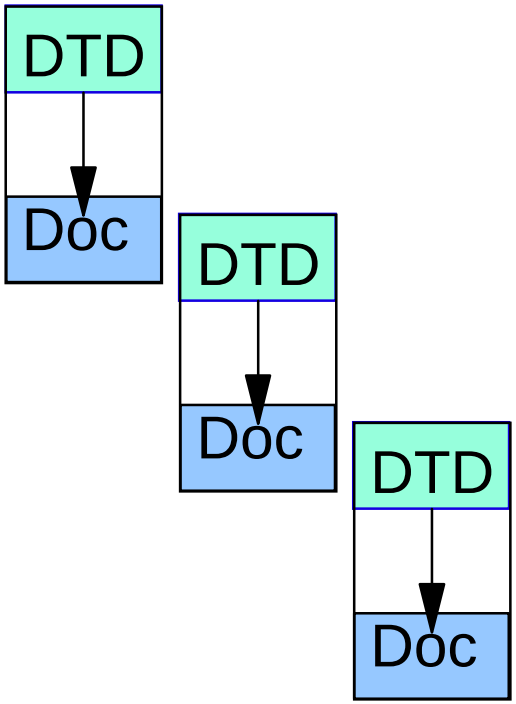
Vehicle id in US

Vehicle id in Norway

XMI Differences

- Interchange incremental differences
- Express differences in terms of another document
- Reduce quantity of information to transfer
- Canonical differences:
  - Add
  - Delete
  - Replace

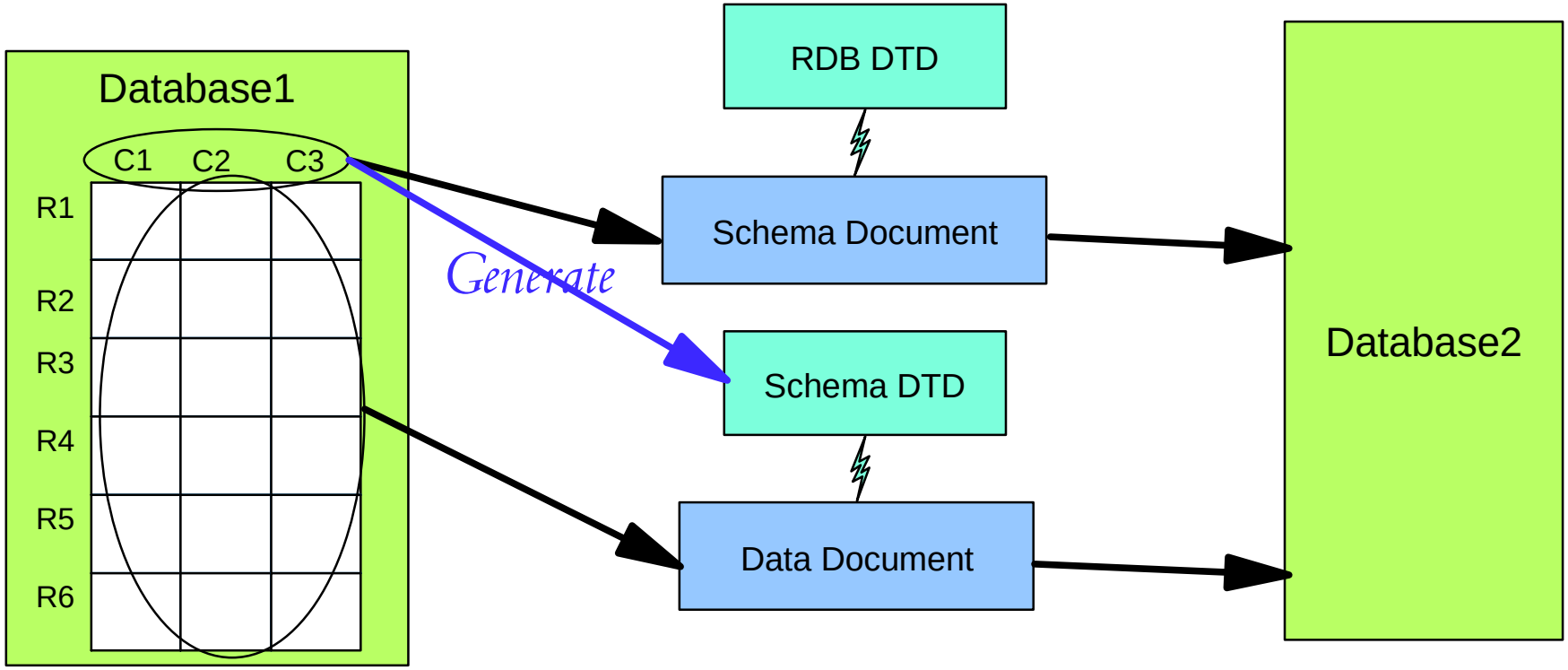
XMI Metadata and DTDs



MOF	Java VM		
UML	Java	C++, IDL	Relational Database
Model	.java, .class	.c, .h, .idl, .obj, .dll	Schema (cols)
Instances	Instances	Instances	Data (rows)

DTD Generation Example

Exchange DB schema (columns) and data (rows)



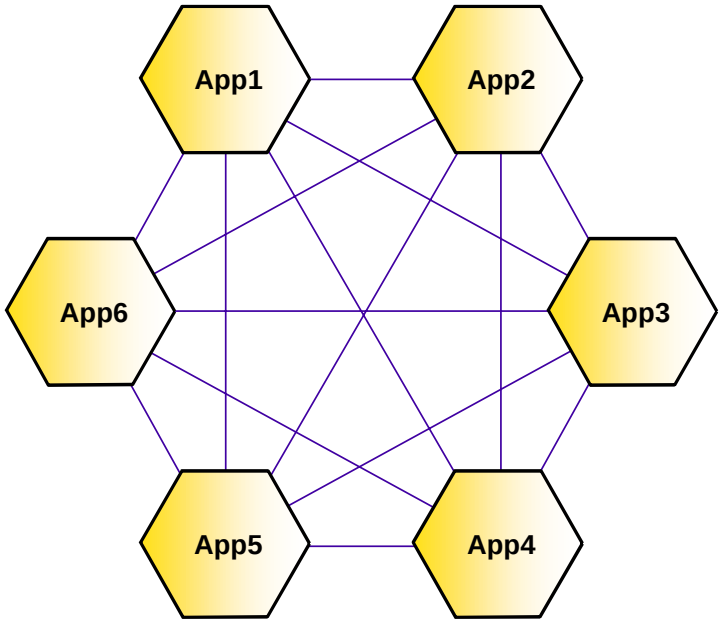
XMI Summary

- XML Metadata Interchange
  - Establish an industry standard specification for a stream-based model format
  - Provide a generic format that can be used to transfer a wide variety of models
- Status
  - OMG standard - March 23, 1999
  - XMI 1.1 - February 1, 2000
  - XMI 1.2 in progress
  - XMI production of XML Schema in progress
- Value
  - Generates DTDs (& Schemas soon) for interchange
  - Generates XML documents using the DTDs
  - Leverages available XML technology
  - Uniform architecture for naming, linking, extensions, round-trip
  - Applicable to UML, databases, messaging, repositories, tool interchange, CCM Components, Java, EJBs
  - Will import DTDs and Schemas

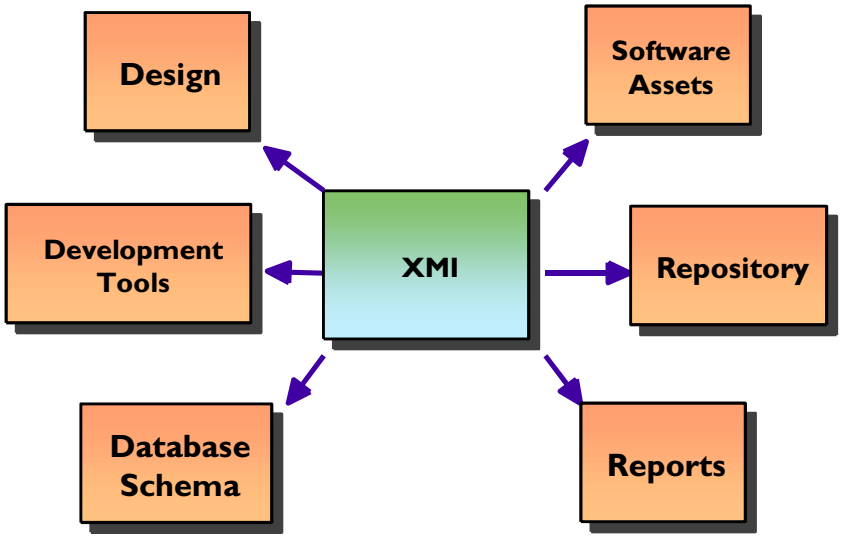
XMI Topics

- XML
- XMI 1.1
- XMI Applications
  - Tools
  - Repositories
  - Messaging
- XMI and XML Schema

Integrating Tools



- 6 vendors (N)
- Write 30 bridges (N*N-N)
- Increased by product releases

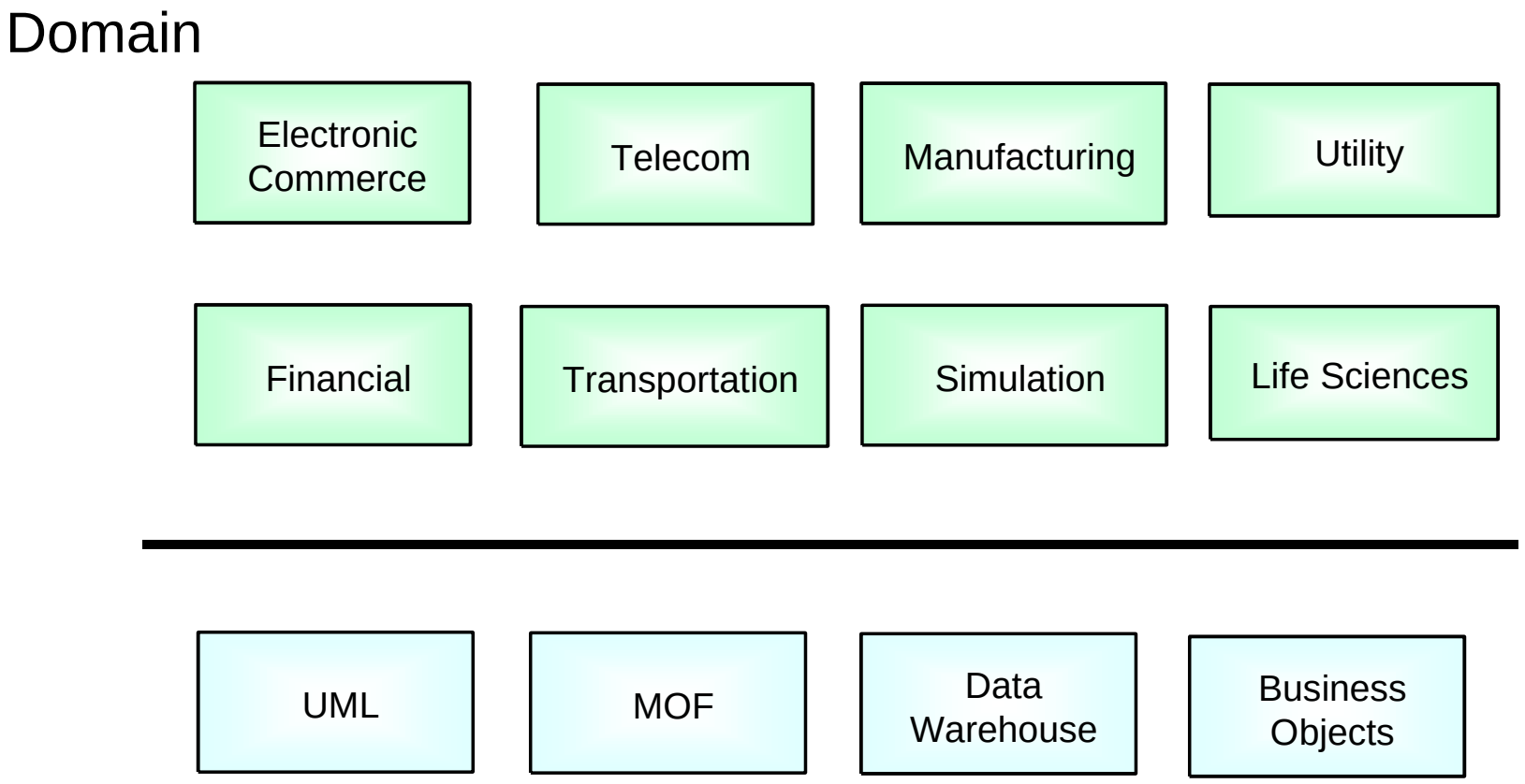


- 6 vendors (N)
- Write 6 bridges (N)

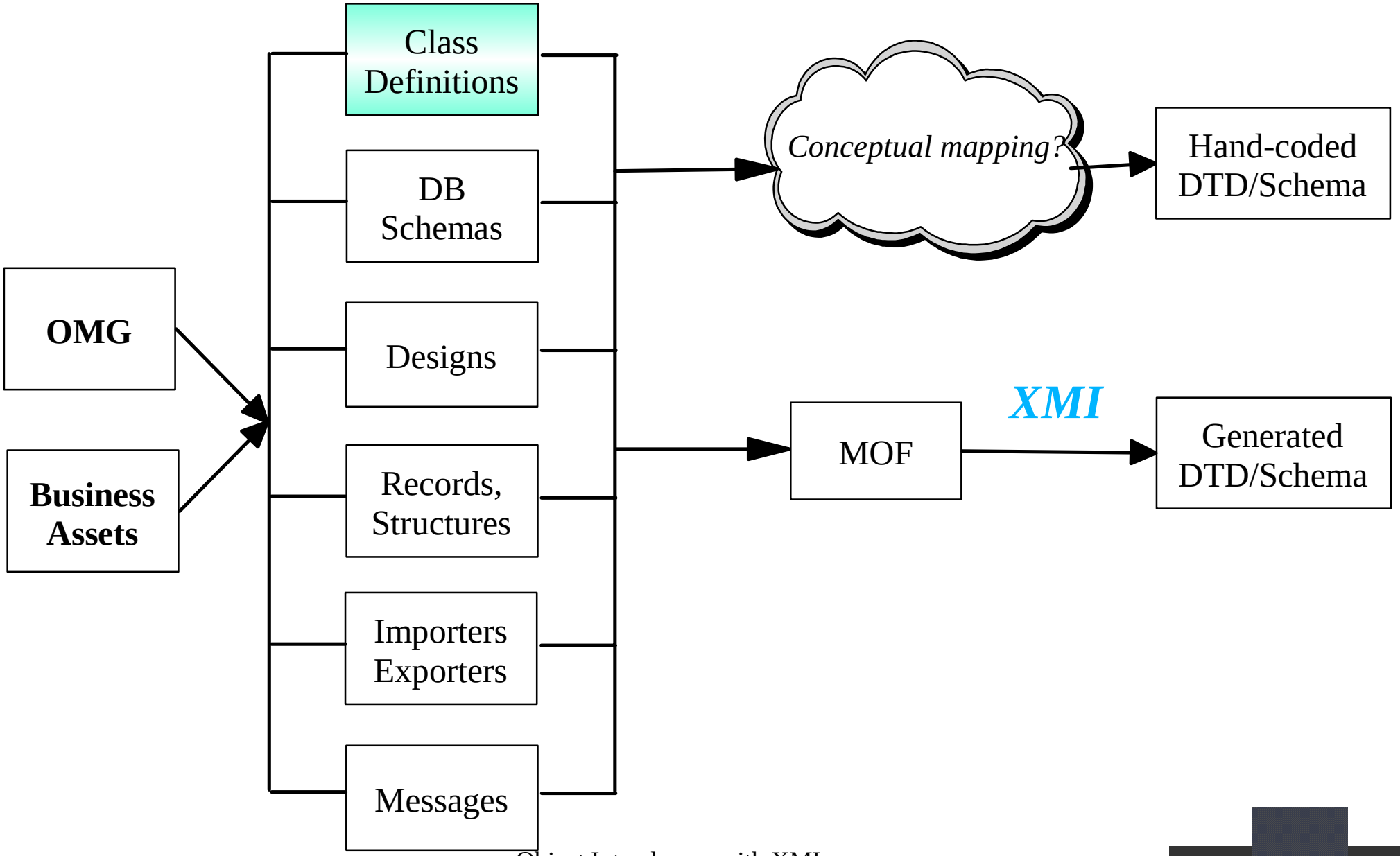
XMI Open Scenarios

- Interchange of information for...
  - Object Oriented Analysis and Design (UML)
  - Data Warehouse (CWM)
- Publish design metadata on the web
  - Leverage XML/HTML infrastructure that already exists
- Open new integration paths between applications
  - Design tool X
  - Language environment Y
  - Database Z

XMI and the OMG

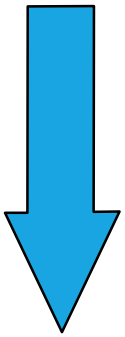


Where do DTDs come from?



Federating Tools: Goals

- Consistent role-relevant views of software assets
- Shared definitions of data, code or components, with effortless interchange among tools



Virtual "global repository"

Federating Tools: Standards

- Deliver assets ⇒ WebDAV

  +

- Define their meaning ⇒ XMI

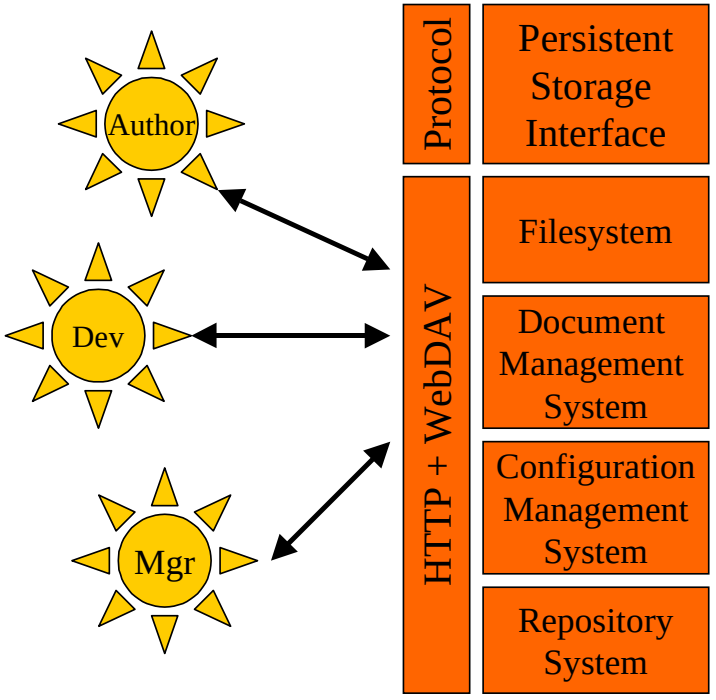
  ⇓

- Virtual "Global Repository"

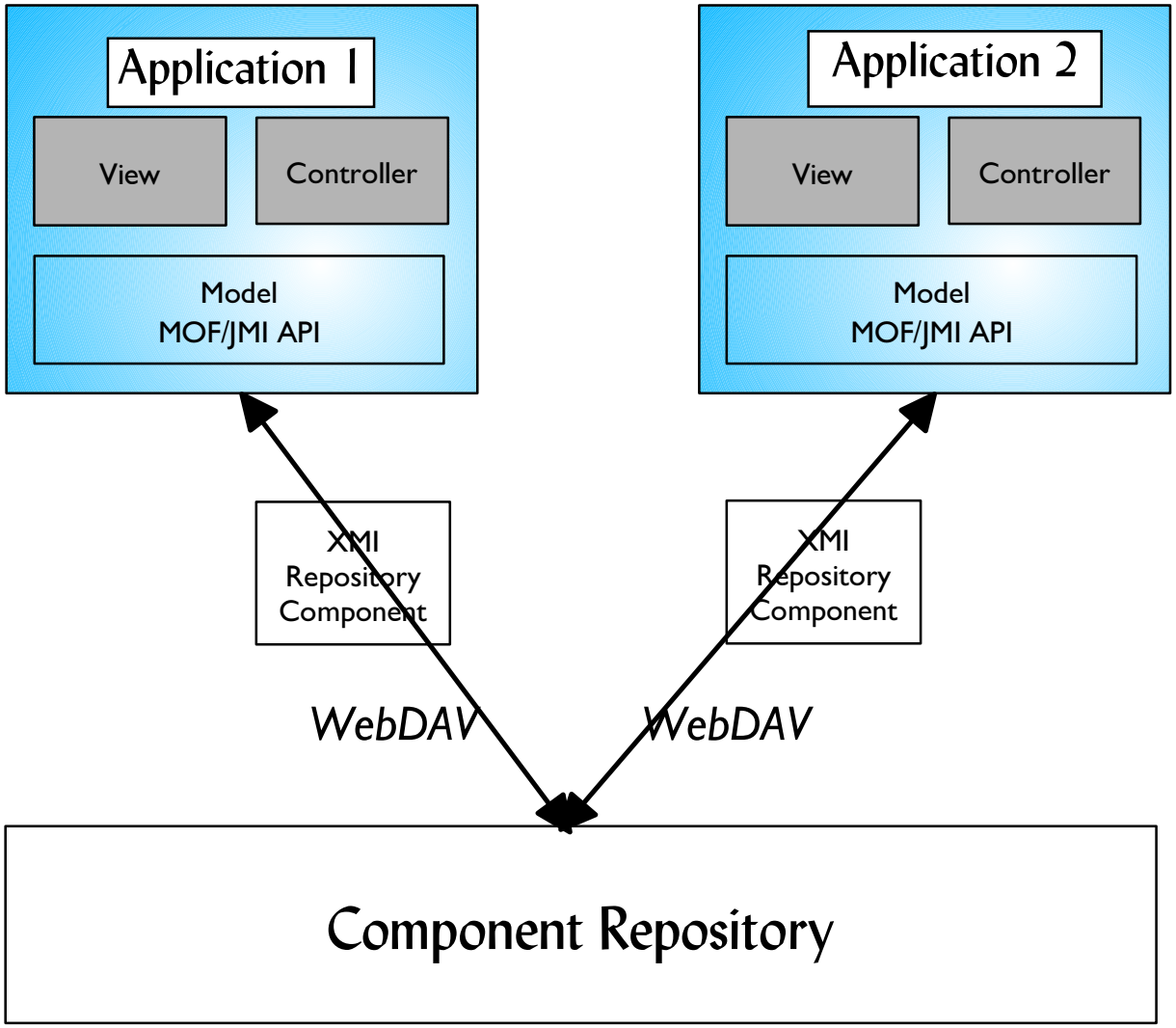
Federating Tools with WebDAV:

Distributed **A**uthoring & **V**ersioning

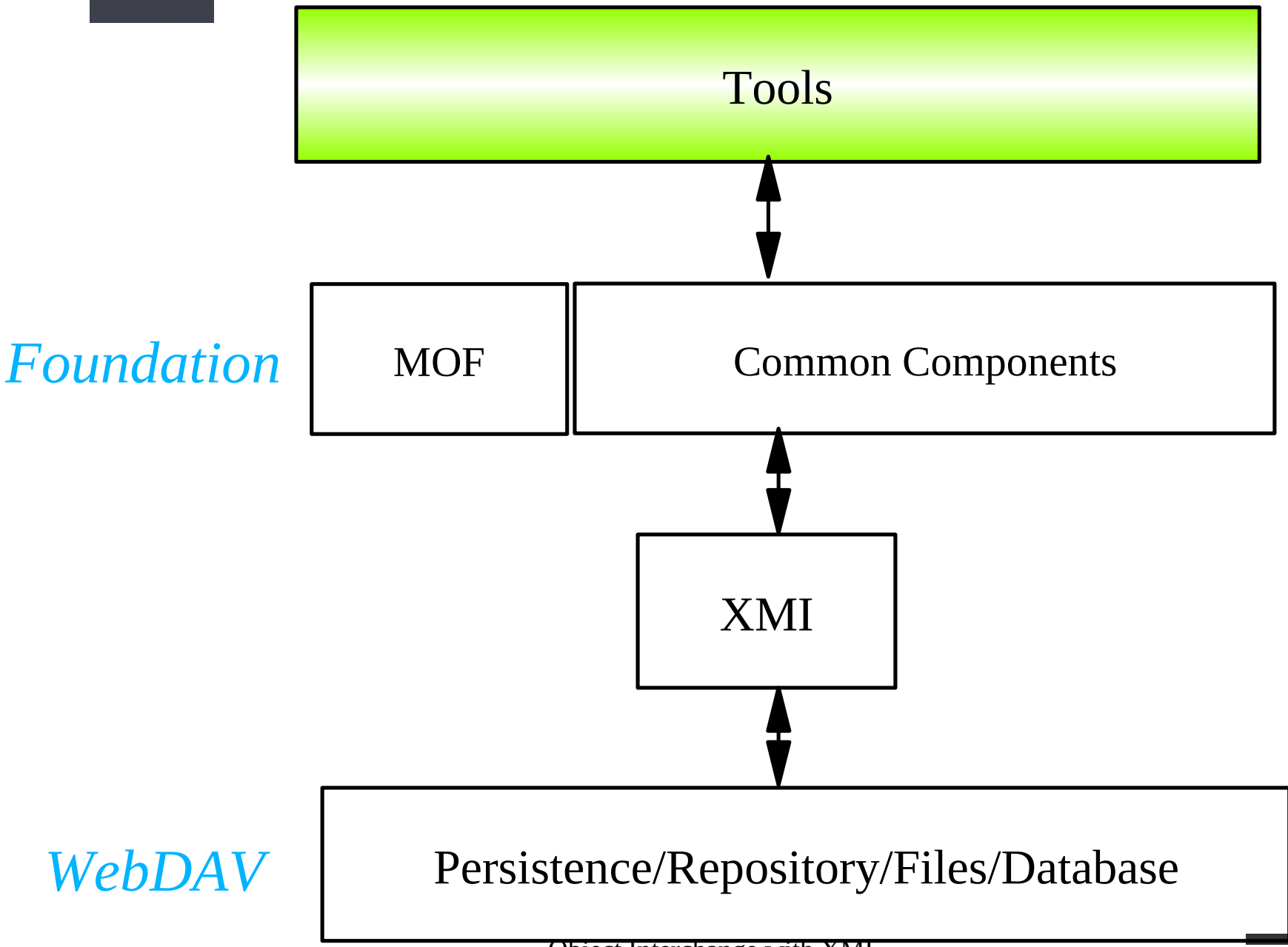
- Capabilities
  - Overwrite prevention
  - Properties
  - Collections
  - Name space management
  - Version management
  - Access control
- Protocol method requests
  - get put
  - propfind proppatch mkcol index addref delref delete copy move lock unlock patch
  - References are URIs



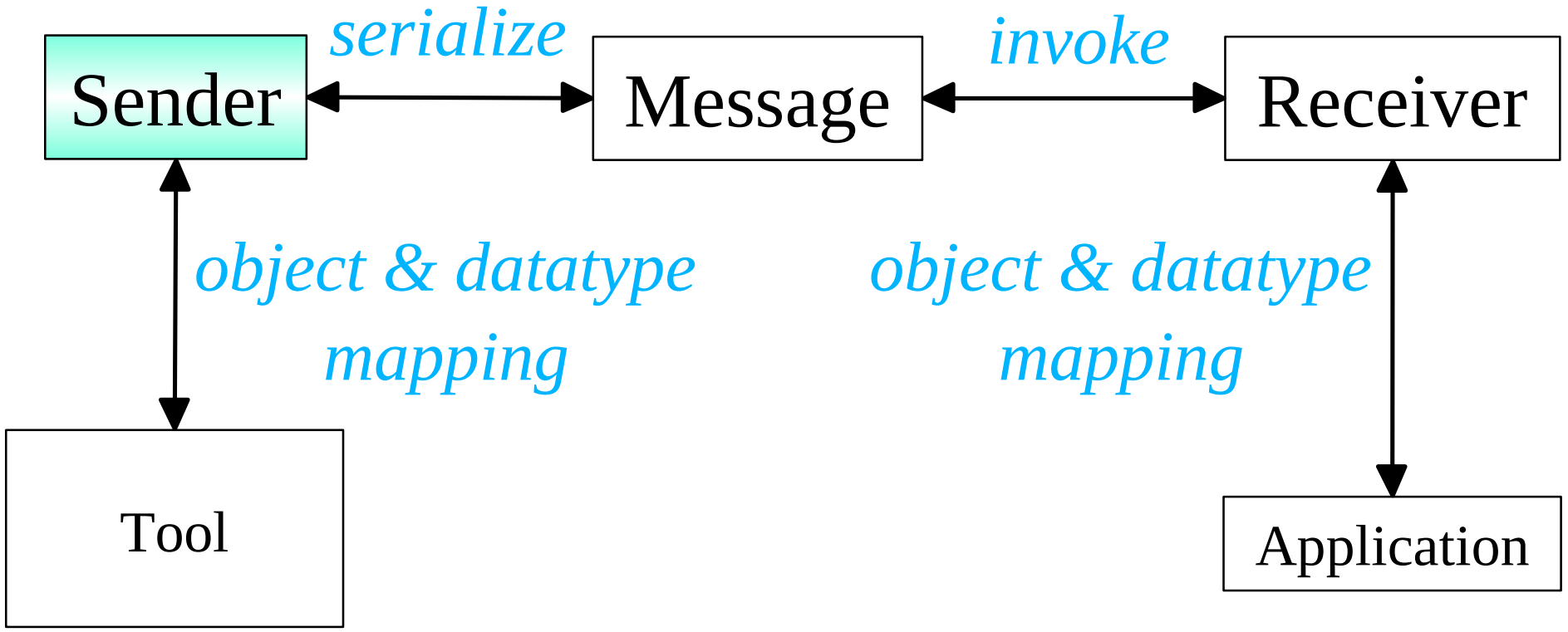
Application object integration



Tools Architecture



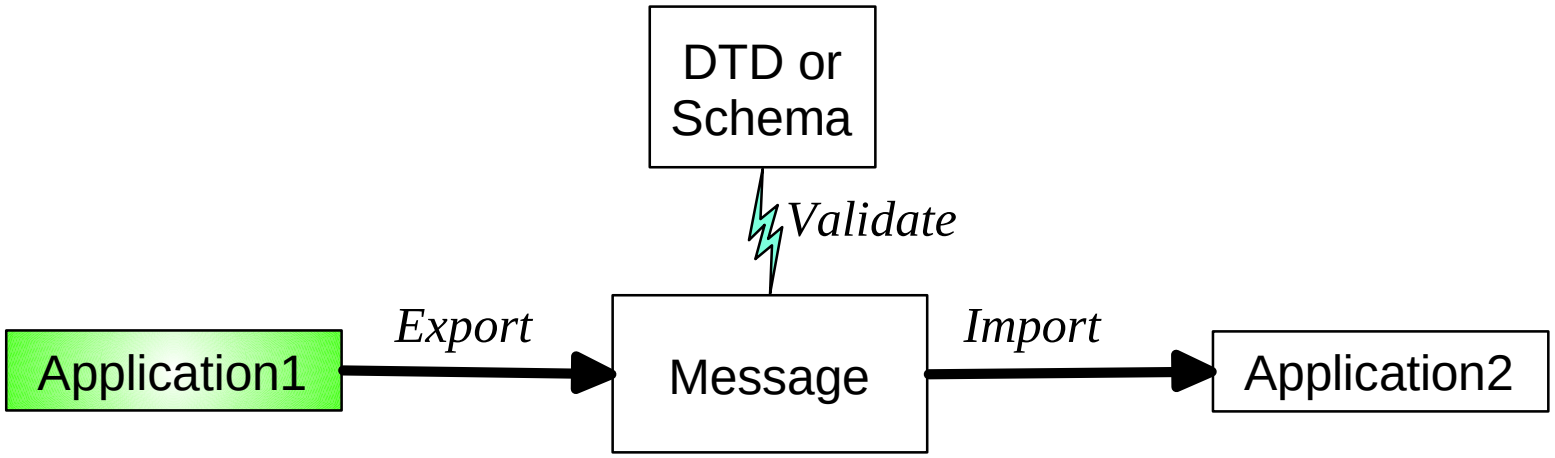
XML/XMI Messaging



SOAP4J uses XMI for object messaging

XMI Messaging

1. Export: XMI message written to file or stream
2. Delivery: Passed to import software
3. Import: Parse XMI message




```
<Order number="123456">
    <contents>
        <Car make="Ford" model="Mustang"/>
    </contents>
</Order>
```

XMI in Messaging

- The fundamental infrastructure necessary to support generic XML-based object messaging requires:
  1. an object model, = MOF
  2. a set of data types, = MOF Language Models
  3. interface specifications, = MOF Application Models
  4. an XML serialization format. = XMI

XMI companies

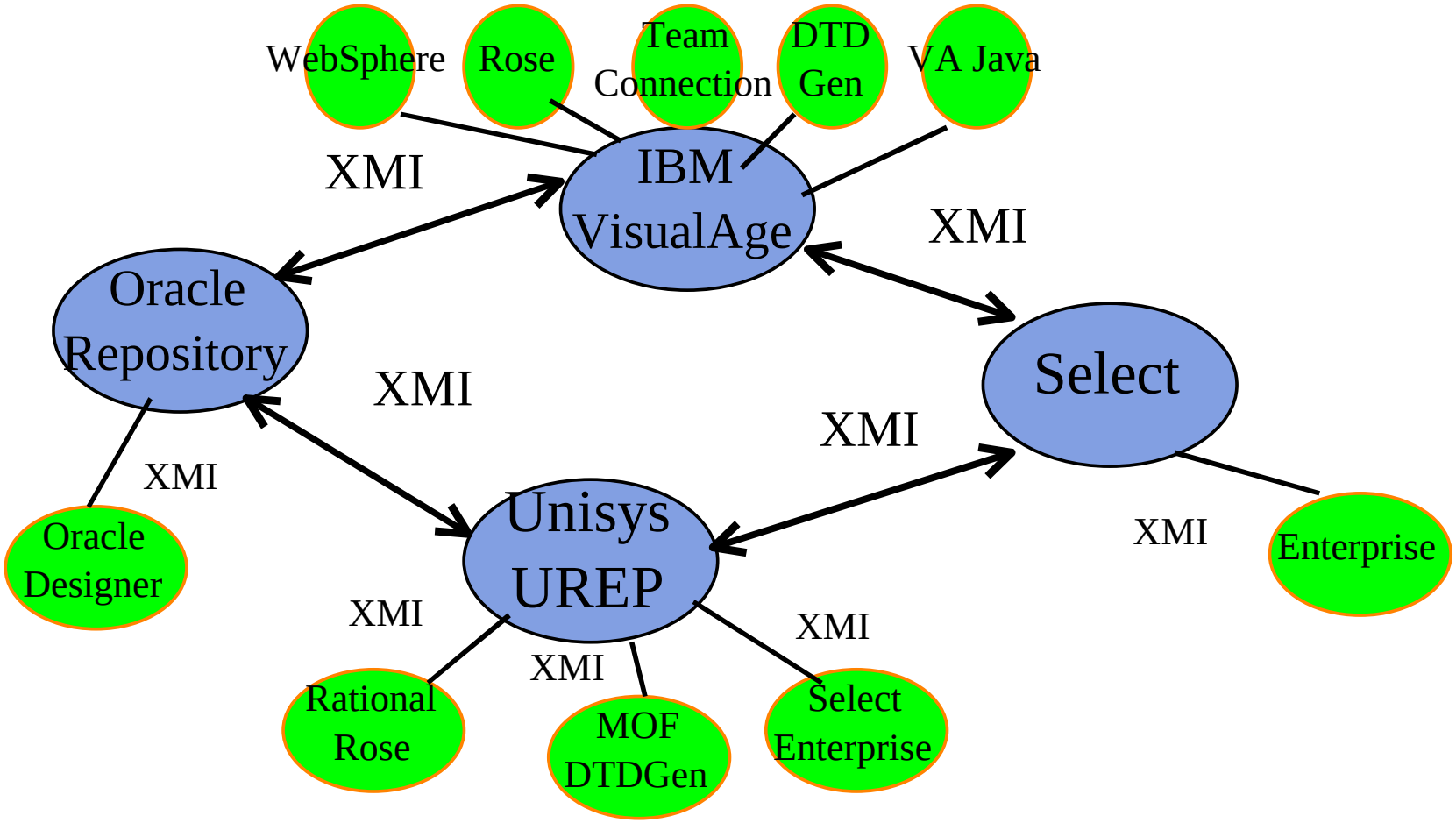
- IBM
- Unisys
- Oracle
- Rational
- Inline
- Hyperion
- Dimension EDI
- Genesis
- Novosoft
- Softeam
- Ontogenics
- NCR
- UBS
- TogetherSoft
- ObjectsByDesign
- Argo at UCI
- Objecteering
- Aonix

XMI used by

- WebSphere Enterprise
- VisualAge for Java
- San Francisco project
- Application Framework for e-Business
- UML, MOF, CCM XMI DTDs
- CWM, Java, EJB XMI DTDs
- XMI Toolkit (AlphaWorks, VADD)
- Unisys UREP, Integrate+
- Oracle Designer 2000
- More on the way...
  - IMS, manufacturing, insurance, MQ

XMI Proof of Concept

(OMG demo - November 1998)



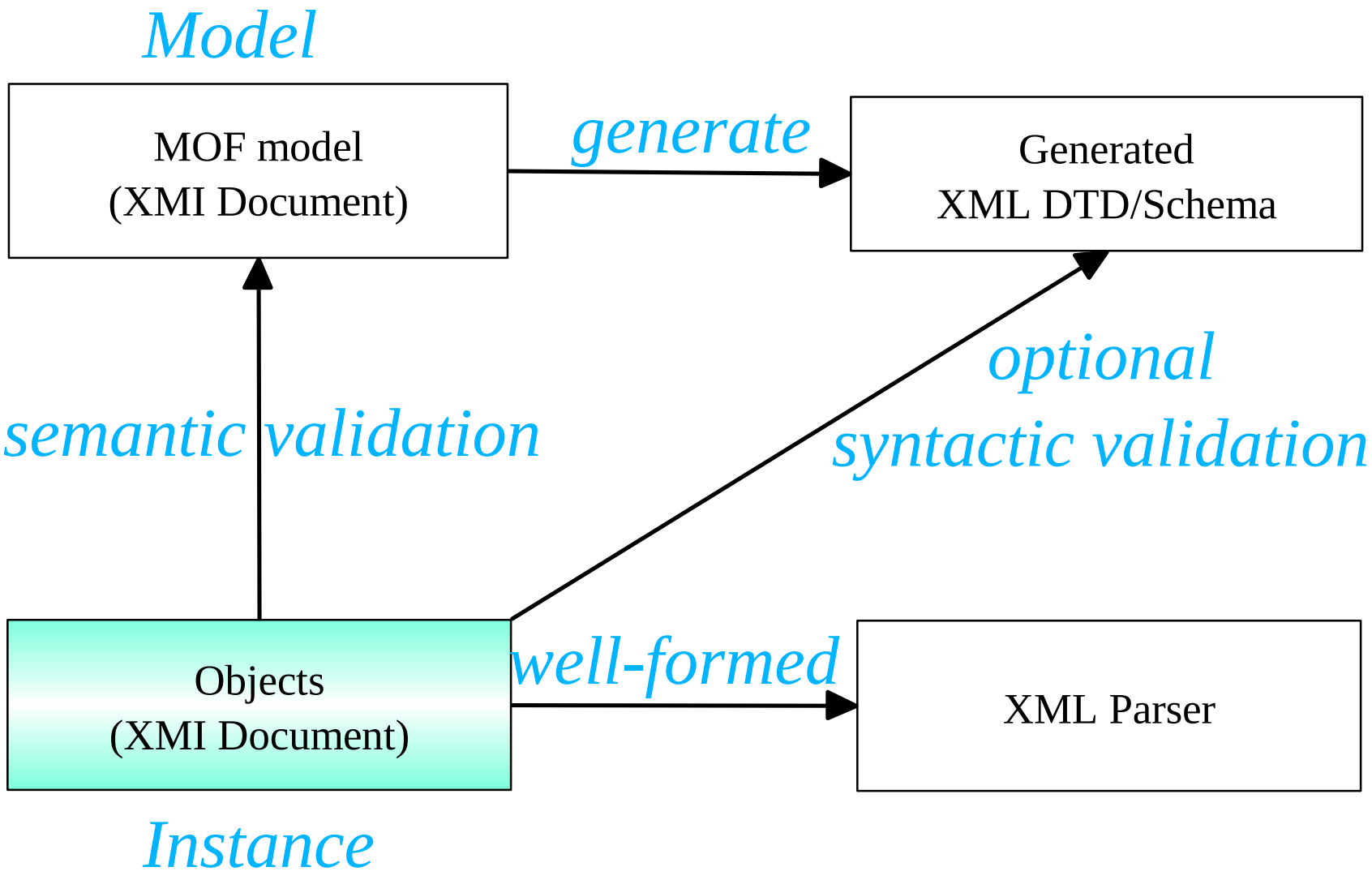
XMI Topics

- XML
- XMI 1.1
- XMI Applications
  - Tools
  - Repositories
  - Messaging
- XMI and XML Schema

XMI Production of XML Schema

- Initial submission
- Generate XML Schemas to match XMI 1.1
  - XMI 1.1 documents identical
  - Backwards compatible
  - Use either DTDs or Schemas
- Generate XMLSchemas for XMI 2.0
  - Uses new features
  - Not backwards compatible with DTDs
- Import XML Schemas and DTDs

Validation in XMI



XML Schema is ...

- A working draft of the W3C since 1998
- Syntactic validation XML documents with more expressive power than DTDs
- Defines nested data structures
- Uses XML syntax
- Uses XML Namespaces
  - (DTDs could also support XML Namespaces)
- Supports single extension points
- Supports a simple set of data types

XML Schema is not ...

- Simple
- Fast
- Concise
  - Twice the size of DTDs (ComplexTypes & Element declarations)
- Generally extensible
  - Multiple extension points not supported
- For semantic validation
- For defining objects

UML vs DTD/XML Schema

- UML
  - High level object-oriented concepts
  - Designed for human comprehensibility
  - Information structure plus dynamics
  - Example concept:
    - "What relationships and scenarios do these business objects participate in?"
- DTD/XML Schema
  - Designed for XML document validation
  - Example concept:
    - "This XML element contains that elemement zero or more times."
  - Less information than databases and software need for robust interchange
  - Captures only a portion of the static information

Standardizing Schemas without models is insufficient

- XML is primarily a serialization technology
- Schemas cannot capture semantics of metamodels
- Schema mapping rules from metamodel must be consistent across applications
- Version DTD/Schemas with metamodel
- Architecture / consistency across DTD/Schemas
  - new features can be rapidly incorporated
  - common look and feel
  - linking, ids, differences, extensions, ...
  - leverage XML advances coherently

XML Validation Compared

- DTD
  - Declares elements and attributes
  - Declares element containment
  - Specify both order and multiplicity
- Schema
  - Declares elements and attributes
  - Declares element containment
  - *Limited: Specify order and/or multiplicity*
  - *Limited: Substitute one element for another*
  - *Limited: Extend previously declared elements*

- Syntactic, not semantic validation
- Must keep in synch (generate) with classes
- Use Classes, not Schemas for Object definitions

XMI / XML Schema challenges

- Multiple extension defined by MOF/UML needed for multiple inheritance
- Substitutability and extension concepts are intertwined
- Requiring base content first interferes with mutple extensions
- Extended groups cannot mix with base content
- Suggest: Mapping types to elements leads to extra complexity

XMI Object Interchange

Questions?

sbrodsky@us.ibm.com

For more info:

- www.omg.org
- alphaworks.ibm.com