XML Schema & MPEG DDL

Outline

- Basic Tools: MPEG-7, XML Schema, DDL
  - Why use MPEG-7 for MMDBMS ?
  - MPEG-7 DDL bases on XML Schema, but defines MPEG-7 specific extensions

    DDL is: “...a language that allows the creation of new Description Schemes and, possibly, Descriptors. It also allows the extension and modification of existing Description Schemes.”

MPEG-7 Overview

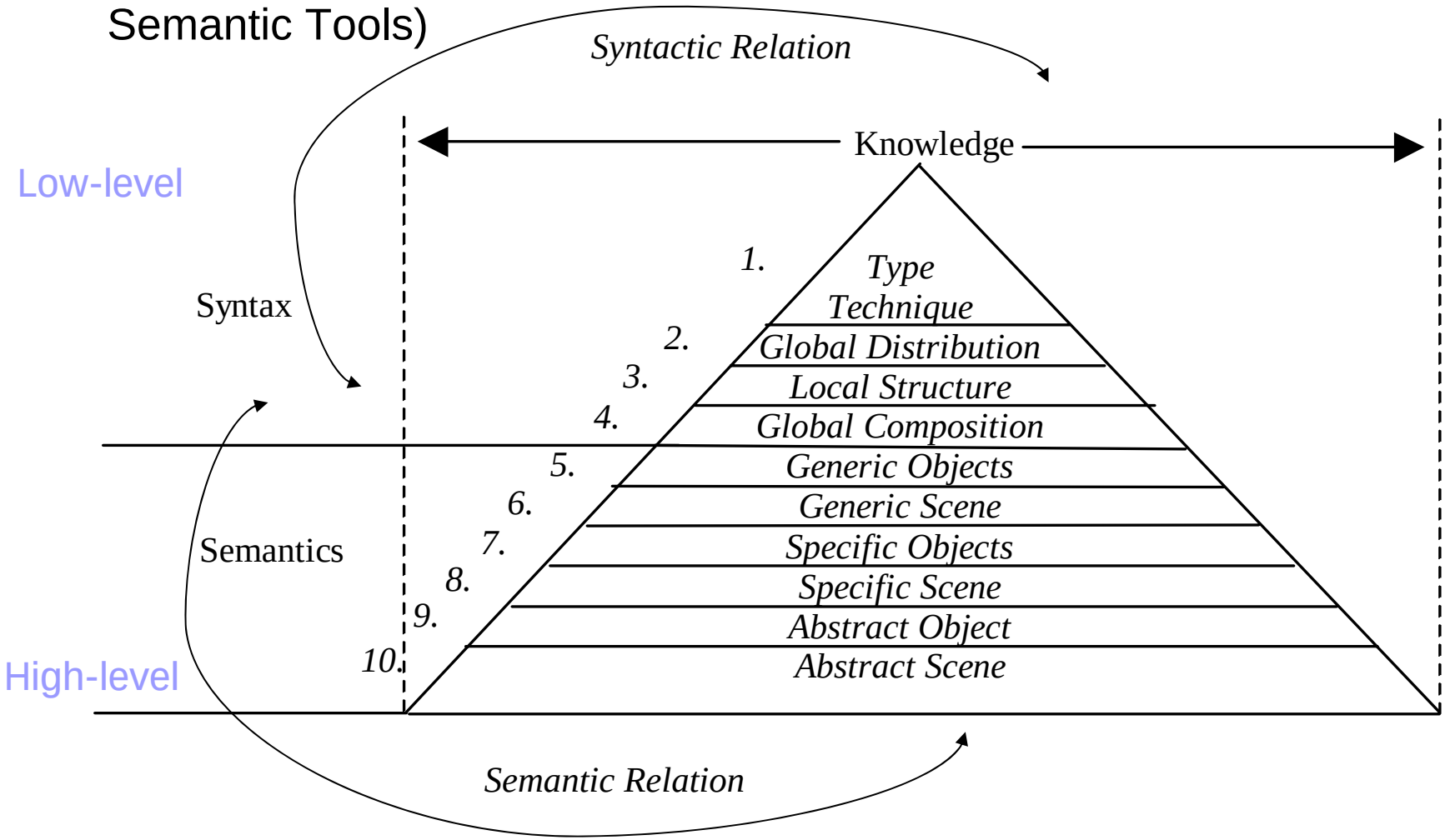
- Brief Overview of MPEG-7 and why it is important for MMDBMS
- Consider mainly Visual and MDS (Multimedia Description Scheme), thus principally the semantics, structural, summary and production-oriented parts.

MPEG-7

- Gives means for a rather complete indexing framework, with respect to the indexing pyramid.
- Considers meta-data life-cycle.
- Interoperability (XML, …)
- Application Push (http://www.mpeg-industry.com)

From low-level to high-level Meta-Data (Aspect 1)

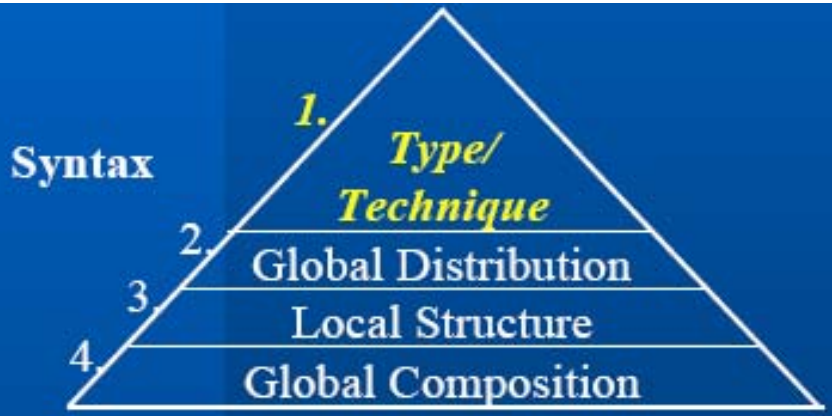
- Multi-level (Visual) Indexing Pyramid [Benitez 2000] (MDS Semantic Tools)



- Why is this important?
  - Guide indexing process
  - Build better indexing systems
- Key points
  - Pyramid represents full range of visual attributes
  - Strong impact on MPEG-7

- The syntactic levels hold attributes that describe the way in which the content is organized, but not their meanings.
  - In images, for example, type could be "color image.“;
  - Global distribution holds attributes that are global to the image (e.g., color histogram;
  - Local structure deals with local components (e.g., lines and circles), and global composition relates to the way in which those local components are arranged in the image (e.g., symmetry).
- The semantic levels, however, deal with the meaning of the elements.
  Objects can be described at three levels:
  - generic, representing everyday objects (e.g., person);
  - Specific, representing individually named objects (e.g., Roger Moore);
  - abstract, representing emotions (e.g., power).

  In a similar way, a scene can be described at these three levels.



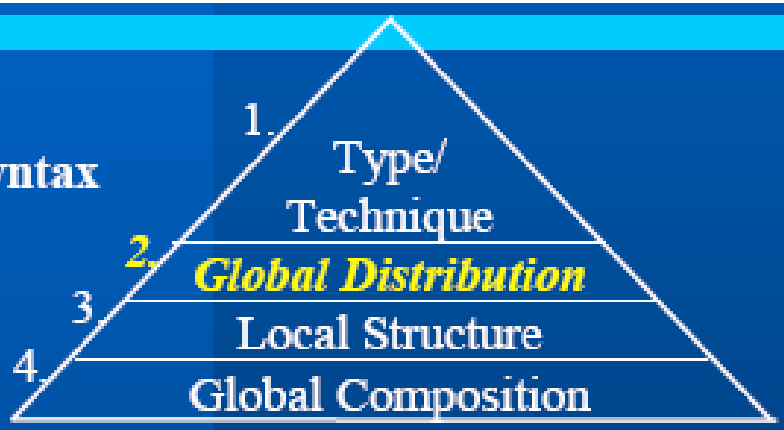
- Type/technique used during production
- No knowledge of visual content, just general visual characteristics

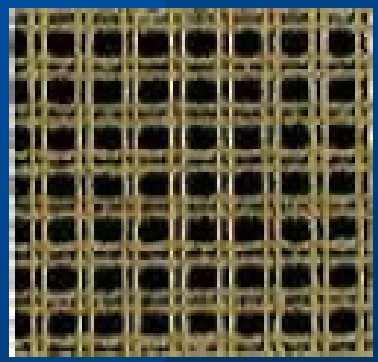
Examples:

- Color or b/w photograph
- Water color or oil painting

Oil painting

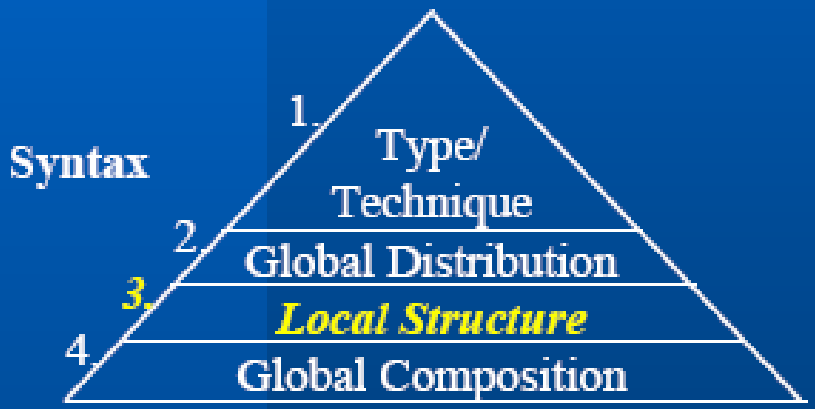
Color photograph



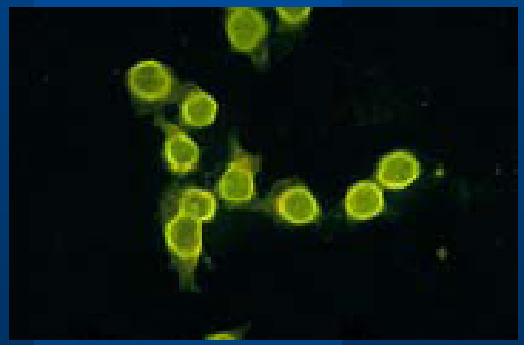


Similar texture, color histogram

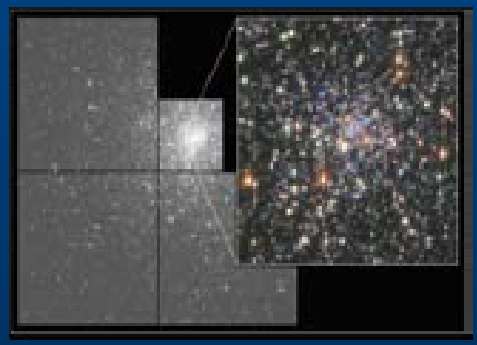
- Distribution of Low-level features only
- Examples:
  - Color distribution
    - Dominant, histogram
  - Global texture
    - Coarseness, contrast
  - Global shape
    - Aspect ratio
  - Global motion/deformation
    - Speed, acceleration



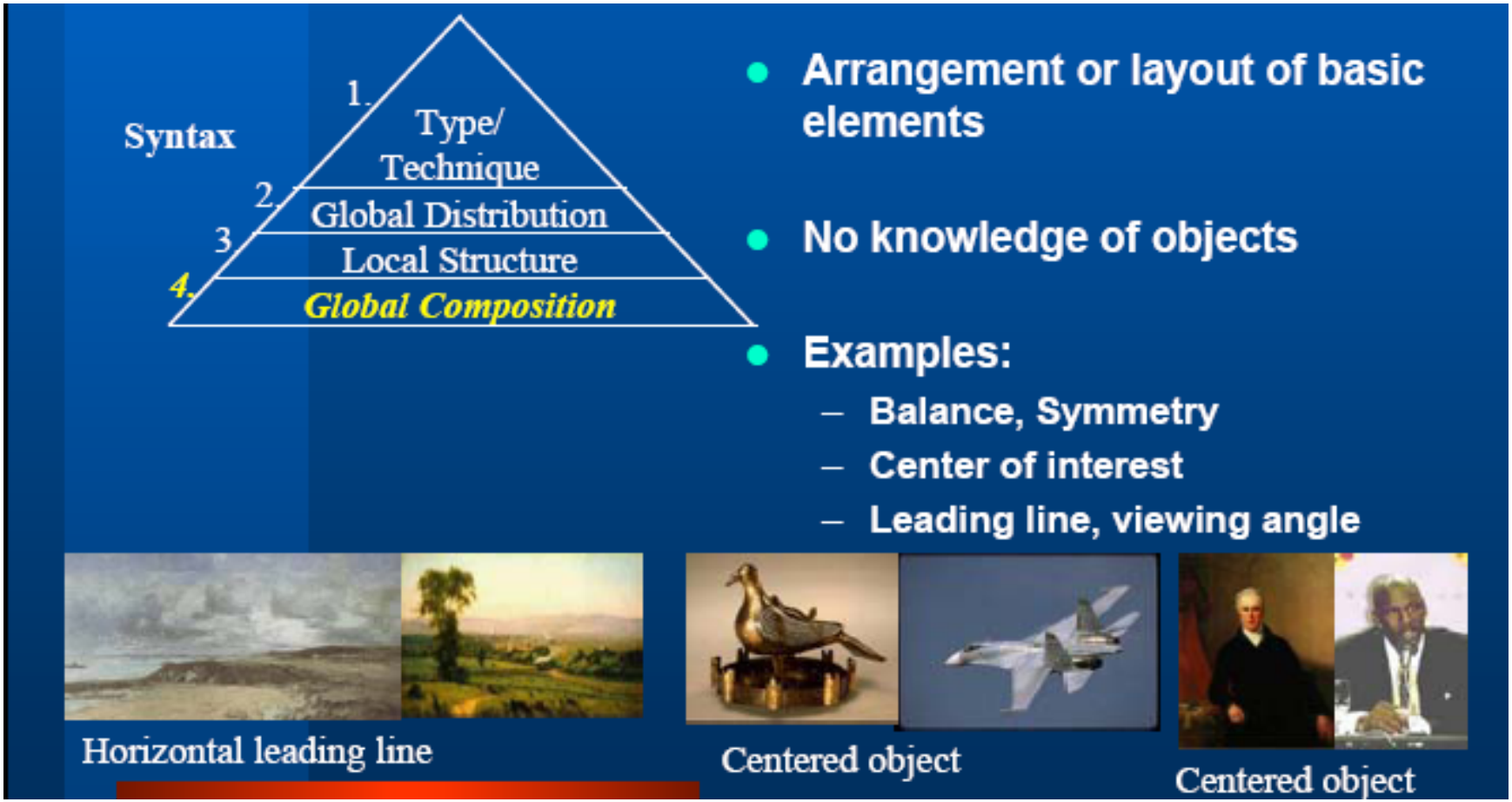
- Characterization and extraction of basic visual elements
- Examples:
  - Dots, lines, tone, circles, squares
  - Local color
  - Binary shape mask
  - Local motion/deformation

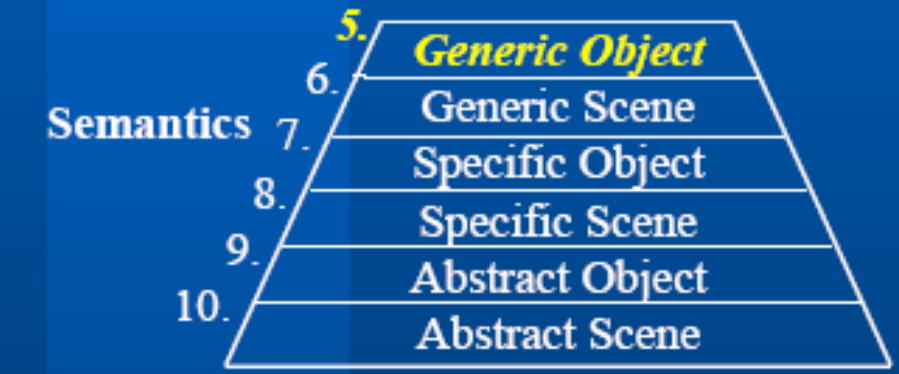


Blood cells = circles



Stars = dots


Syntax
1. Type/ Technique
2. Global Distribution
3 Local Structure
4. Global Composition
Arrangement or layout of basic elements
No knowledge of objects
Examples:
Balance, Symmetry
Center of interest
Leading line, viewing angle
Horizontal leading line
Centered object
Centered object

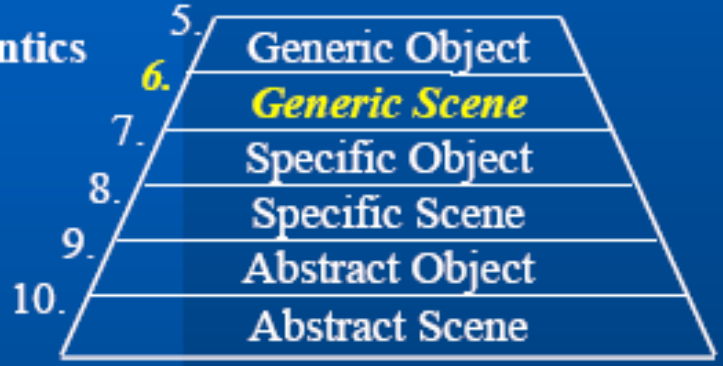


- General (every day) knowledge about objects
- Examples:
  - Common nouns
    - Person
    - Chair
    - Desk

Airplane

Persons, flag

What the image is "of"

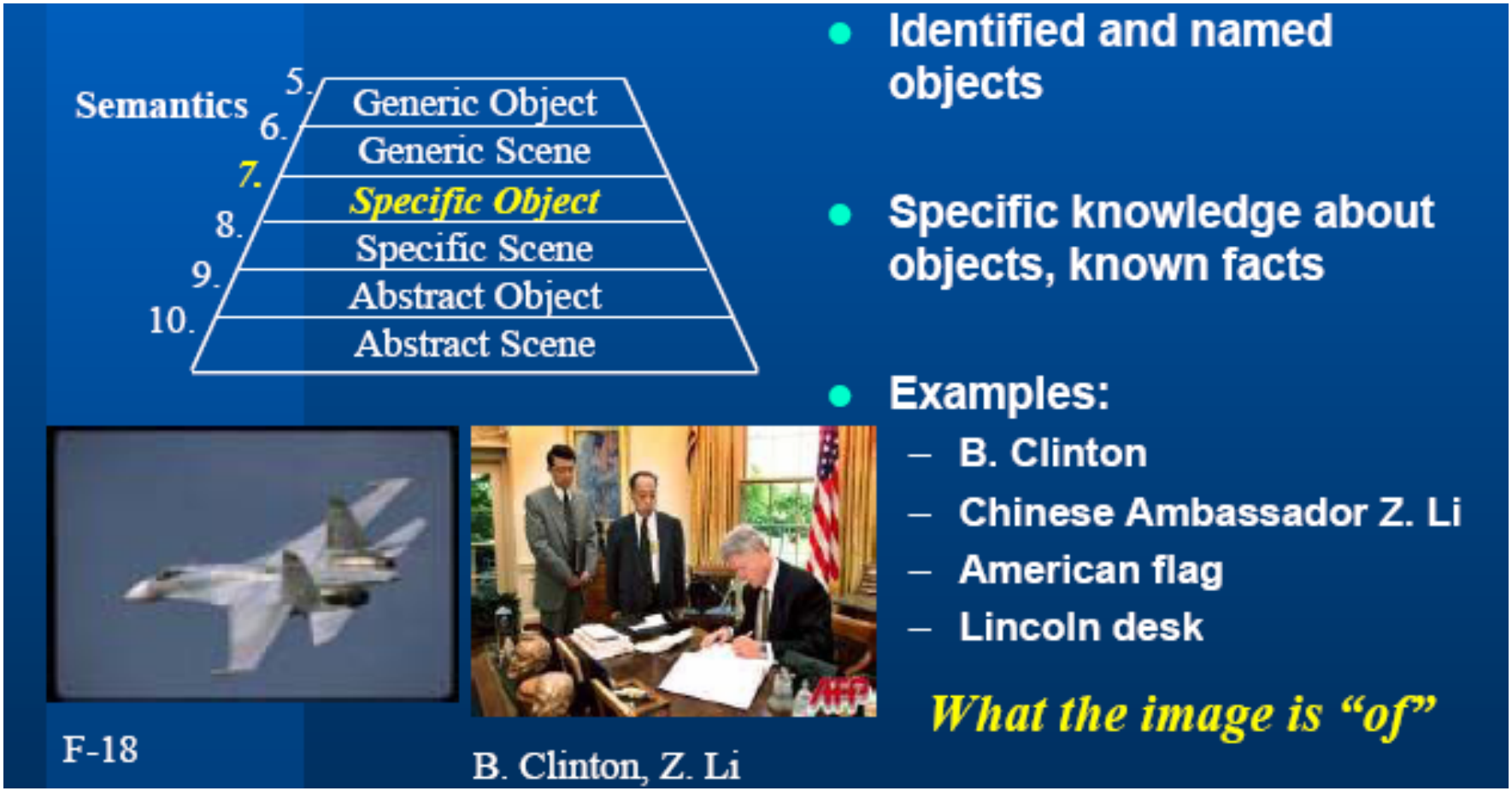


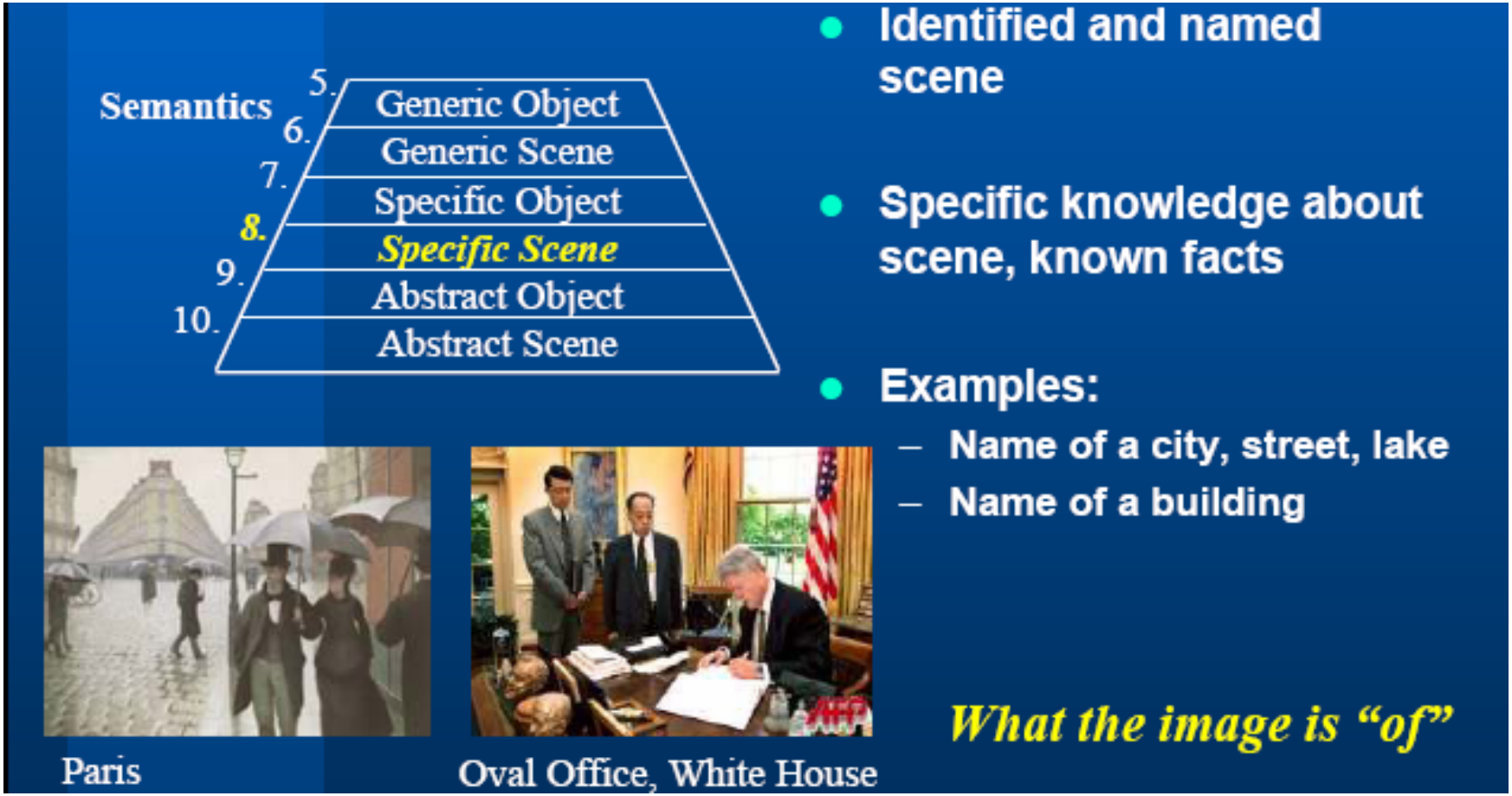
- General knowledge about scene
- Examples:
  - City, Landscape
  - Indoor, Outdoor
  - Daytime, Nighttime

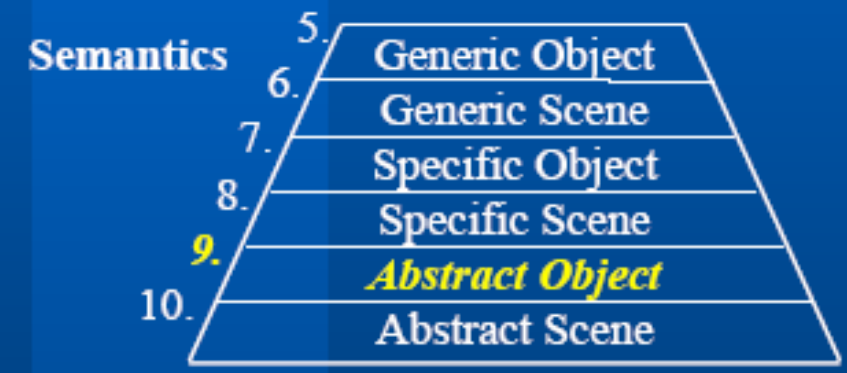
Outdoors, city, street

Indoors, office

What the image is "of"


Semantics
5. Generic Object
6. Generic Scene
7. Specific Object
8. Specific Scene
9. Abstract Object
10. Abstract Scene
F-18
B. Clinton, Z. Li
Identified and named objects
Specific knowledge about objects, known facts
Examples:
B. Clinton
Chinese Ambassador Z. Li
American flag
Lincoln desk
What the image is "of"


Semantics
5. Generic Object
6. Generic Scene
7. Specific Object
8. Specific Scene
9. Abstract Object
10. Abstract Scene
Identified and named scene
Specific knowledge about scene, known facts
Examples:
Name of a city, street, lake
Name of a building
What the image is "of"
Paris
Oval Office, White House

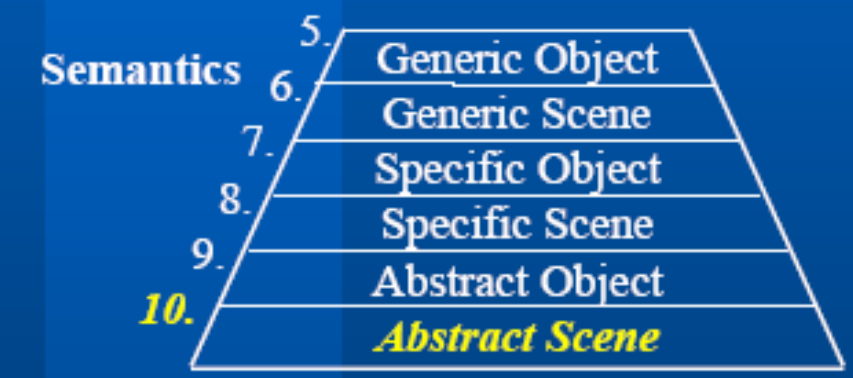


- Interpretation of an object
- Subjective or based on specific personal knowledge
- Examples:
  - Political power
  - Sympathy

About baseball (or basketball?)

Political Gesture

What the image is "about"



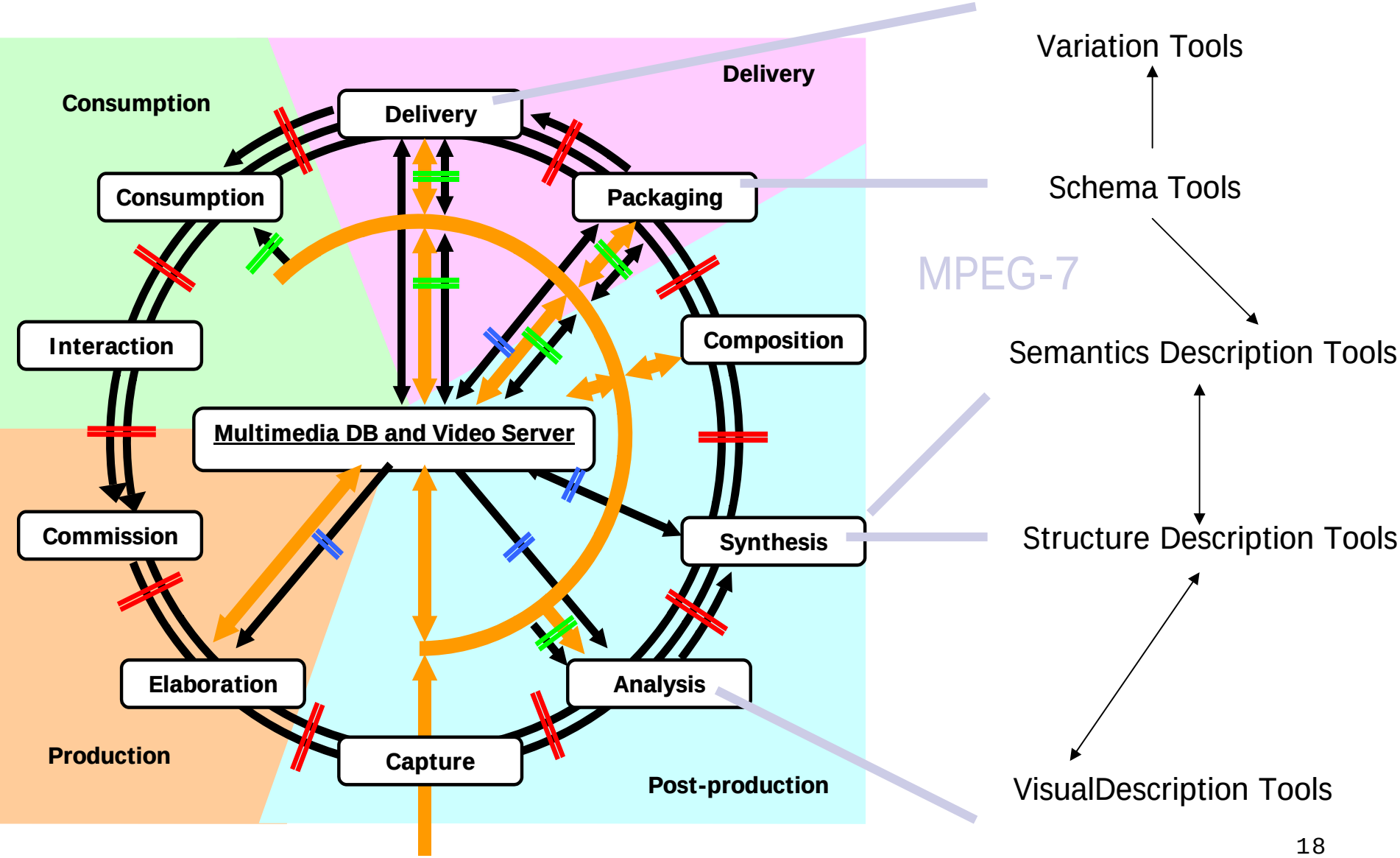
- Subjective interpretation of a scene
- Examples:
  - International politics
  - War
  - Apology

Peacefulness

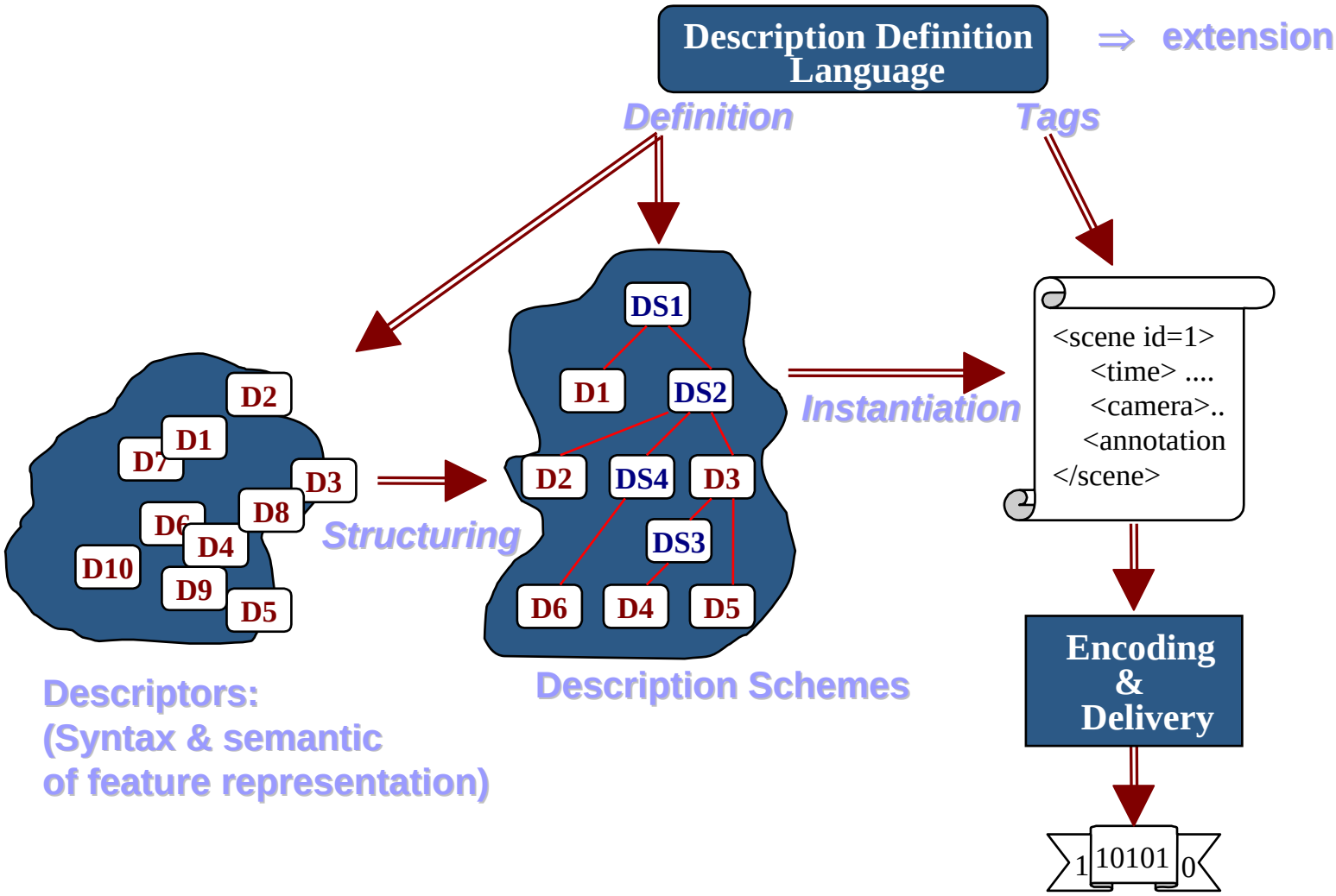
US Government

What the image is "about"

Multimedia Meta-Data Lifecycle (Aspect 2)



MPEG-7 Network (what it glues together)



Mpeg-7 Visual (1)

- Basic structures (5 D's)
  - Grid layout
  - Time series
  - Multiple view
  - Spatial 2D co-ordinates
  - temporal interpolation
- Color (7 D's)
  - Color Space & Color Quantization
  - Scalable Color: HSV color space & Haar transformation
  - Dominant Color
  - Color Layout, Color structure
  - Group-of-Frames/Group-of-Pictures color

Mpeg-7 Visual (2)

- Texture (3 D's)
  - Homogenous: directionality, coarseness and regularity of patterns
  - Non-Homogenous (Edge Histogram)
- Shape (3 D's)
  - Contour-based: Curvature Scale-Space (CCS)
  - Region-based: Angular Radial Transformation
  - 3D
- Motion (4 D's)
  - Motion Activity: intensity, direction, spatial distribution
  - Camera Motion
  - Motion Trajectory
- Localization (2 D's)
  - Region locator
  - Spatial-temporal locator
- Face recognition (1 D)

MPEG-7 Data Definition Language DDL

- MPEG-7 History: 1998-2000 CfP for DDL (Data Definition Language) together with XML Schema development.
- What are the elements of DDL:
  - XML-Schema:
    - Namespaces, element and attribute declerations, and type definitions.
    - Attribute and model group declarations, identity-constraint definitions and notation declarations.
    - Helpers: annotations, model groups, particles, wildcards.
  - XML-specific Schema:
    - Array and Matrix Data Type, Build-in Derived Data Types (e.g., basicTimePoint)

Schema Components

- Simple type definitions
- Complex type definitions
- Element declarations

} Primary components

- Attribute Group definitions
- Identity constraint definitions
- Model Group definitions
- Notation declarations

} Secondary components

- Annotations
- Attribute declarations
- Model groups
- Particles
- Wildcards

} Helper components

New Features

- Separate simple and complex types
- Derived types by extension or restriction
- Can prevent certain types of derivation and subclass substitutions
- Can define elements with null content
- Can create equivalent elements
- Can specify element content as being unique (keys on content) and uniqueness within a region
- Wildcards

Schema Wrapper

```
<xs:schema
    xmlns:xs="http://www.w3.org/1999/XMLSchema"
    targetNamespace="http://www.mpeg.org/mpeg-7"
    version="1.1">
….
</xs:schema>
```

xmlns:xs - use the 'xs' prefix to reference elements defined in a schema from another namespace

targetNamespace - all the elements and types defined in this schema come from this namespace. Use this URI to import or include these definitions in other schemas

- Example of MPEG7 MDS schema wrapper :

```
<schema targetNamespace="urn:mpeg:mpeg7:schema:2001"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
```

Main Schema Components

Definitions of:

- Complex types = sub-elements + attributes
- Simple types = no sub-elements, constraints on strings (datatypes)

Declarations of:

- elements (of simple and complex types)
- attributes (simple types), attribute groups

Declaration vs Definition

- *declarations* - things that will be used in an XML instance document; enable appearance of elems and attrs, with specific names and types
  - element declarations
  - attribute declarations
- *definitions* - things that are just used in the schema document; a definition creates a new type
  - simple type definitions
  - complex type definitions
  - attribute group, model group definitions

Simple Type Definitions

Can have: built-in, pre-declared or anonymous simple type defns.

```
<attribute name="State1" type="string"/>

<simpleType name="US-State" base="string">
    <enumeration value="AK"/>
    <enumeration value="AL"/>
    <enumeration value="AR"/>
……
</simpleType>
<attribute name="State2" type="US-State"/>
```

Example of MPEG-7 simple type

```
<!-- Definition of countryCode datatype -->
<simpleType name="countryCode">
    <restriction base="string">
        <whiteSpace value="collapse"/>
        <pattern value="[a-zA-Z]{2}"/>
    </restriction>
</simpleType>
```

Anonymous Simple Type Defn.

```
<attribute name="myAttribute" default="42">
    <simpleType base="integer">
        <minExclusive value="0"/>
    </simpleType>
</attribute>
```

- If you define many types that are referenced only once and contain very few constraints, a type can be more succinctly defined as an anonymous type which saves the overhead of having to be named and explicitly referenced

Complex Type Definitions

```
<complexType name="personName">
    <element name="title" type="string"/>
    <element name="forename" type="string"/>
    <element name="surname" type="string"/>
    <attribute name="age" type="integer"/>
</complexType>
<element name="producer" type="personName"/>
```

Anonymous Complex Type Defn.

```
<element name="personName">
    <complexType>
        <element name="title"/>
        <element name="forename"/>
        <element name="surname"/>
        <attribute name="age" type="integer"/>
   </complexType>
</element>
```

Example of MPEG-7 complex type

```
<!-- Definition of KeywordAnnotation datatype -->
<complexType name="KeywordAnnotationType">
    <sequence>
        <element name="Keyword" maxOccurs="unbounded">
            <complexType>
              <simpleContent>
                <extension base="mpeg7:TextualType">
                    <attribute name="type" use="optional" default="main">
                        <simpleType>
                          <restriction base="NMTOKEN">
                            <enumeration value="main"/>
                            <enumeration value="secondary"/>
                            <enumeration value="other"/>
                          </restriction>
                        </simpleType>
                    </attribute>
                </extension>
              </simpleContent>
            </complexType>
        </element>
    </sequence>
    <attribute ref="xml:lang" use="optional"/>
</complexType>
```

Example of MPEG-7 anonymous type

```
<element name="Affiliation" minOccurs="0" maxOccurs="unbounded">
        <complexType>
                <choice>
                        <element name="Organization"
                                type="mpeg7:OrganizationType"/>
                        <element name="OrganizationRef"
                                type="mpeg7:ReferenceType"/>
                        <element name="PersonGroup"
                                type="mpeg7:PersonGroupType"/>
                        <element name="PersonGroupRef"
                                type="mpeg7:ReferenceType"/>
                </choice>
        </complexType>
</element>
```

Element Declarations

```
<element name="myglobalelt1" type="mySimpleType"/>
<element name="myglobalelt2" type="myComplexType"/>
<element name="myglobalelt3">
    <complexType>
        <element name="mylocalelt" type="otherType"/>
        <element ref="myglobalelt2"/>
        <attribute name="mylocalattr" type="date"/>
    </complexType>
</element>
```

note: can't have named element refs

Element Declarations

```
<element name="myglobalelt1" type="mySimpleType"/>
<element name="myglobalelt2" type="myComplexType"/>

<element name="myglobalelt3">
    <complexType>
        <sequence>
            <element name="mylocalelt1" type="otherType1"/>
            <element name="mylocalelt2" type="otherType2"/>
        <sequence>
    </complexType>
</element>
```

Anonymous Type

References

- Sometimes it is preferable to reference an existing element rather than declare a new one:

<element ref="Country" minOccurs="1"/>

- Country is defined elsewhere, in general is defined globally (under schema rather than as part of complex type defn.) The content of this element Country has to be consistent with the element's type.

Attribute Declarations

```
<attribute name="lang" type="language" use="optional"
    default="en-uk"/>

<element name="Annotation"/>
    <complexType>
        <attribute ref="lang"/>
    </complexType>
</element>
```

Example XML:

```
<Annotation lang="en-us">
```

Constraints on Element Content

content =

- textOnly - only character data
- mixed - character data appears alongside subelements
- elementOnly - only subelements
- empty - no content (only attributes)
- any

```
<element name="price">
    <complexType content="empty">
        <attribute name="currency" type="string"/>
        <attribute name="value" type="decimal"/>
    </complexType>
</element>
<price currency="AUD" value="256.76"/>
```

Schemas Secondary Components

- Content Model Constraints
  - Empty
    - No child elements only attributes
  - Mixed
    - character data appears between element and child
  - elementOnly
    - default type: elements and attributes only
  - textOnly
    - used when deriving complexType from simpleType the new type contains only character data

Schemas: Secondary Components

- Mixed Content Model Example

```
<element name="salutation">
   <complexType content="mixed">
            <element name="name" type="string' />
    </complexType>
</element>

<salutation> Dear Mr. <name> Robert Smith
   </name>,<salutation>
```

Null Elements

- use the nullable attribute to indicate element content is null
- only applies to elements not attributes

```
<element name="endTime" type="time" nullable="true"/>

xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
…..
  <endTime xsi:null="true"></endTime>
```

Abstract Elements

- An abstract element is a kind of template/placeholder.
- If an element is abstract then it cannot appear in the XML instance document

  <element name=”videodoc" type=”doc" abstract="true"/>
- However, elements that are equivalent to the abstract type may appear in its place.

  <element name=”myvideo1” equivClass=“videodoc”/>

Abstract Types

- Types can also be defined as abstract
- Instances of elements of abstract type must use *xsi:type* to indicate a derived type that is not abstract.

```
<complexType name='Vehicle' abstract='true'/>
<complexType name='Car' base='Vehicle'/>
<complexType name='Plane' base='Vehicle'/>

 <element name='transport' type='Vehicle'/>

    <transport xsi:type ="Car"/>
```

Group Element

- The *group* element enables you to group together element declarations.
  - Groups can be incorporated by reference in complexType defns.
- the *group* element is only for grouping together element declarations, no attribute declarations allowed.
- Compositors - *all*|*sequence*|*choice*

Group Example

```
<group name="myModelGroup">
    <sequence>
        <element name="firstThing" type="type1"/>
        <element name="secondThing" type="type2"/>
    </sequence>
</group>

<complexType name="newType">
   <choice>
       <group ref="myModelGroup"/>
       <element ref="anotherThing"/>
  </choice>

</complexType>
```

Group Example

- Unnamed groups

```
<complexType name="myGroup1">
    <sequence>
        <element name="firstThing" type="type1"/>
        <element name="secondThing" type="type2"/>
    </sequence>
</complexType>

<complexType name="myGroup2">
   <choice>
      <element name ="myfirstThing type="type1"/>
      <element name="anotherThing type="type2"/>
  </choice>
</complexType>
```

Attribute Groups

```
<attributeGroup name="myAttrGroup">
        <attribute name="myD1" type="string"/>
        <attribute name="myD2" type="integer"/>
        <attribute name="myD3" type="date"/>
</attributeGroup>
<complexType name="myDS">
      <element name="myelement" type="myType"/>
      <attributeGroup ref="myAttrGroup"/>
</complexType>
```

note: attribute declarations always come last after the element declarations

Derived Types

Subclassing of type definitions

- derive by *extension*: extend the parent type with more elements
- derive by *restriction*: constrain the parent type by constraining some of the elements to have a more restricted range of values, or a more restricted number of occurrences.
- derive by *reproduction* - copy of the parent type

Deriving Types by Extension

- extend a base type's content by adding elements and/or attributes

```
<complexType name="newType" base="baseType"
                                        derivedBy="extension"/>
    <element name="newelt" type="string"/>
    <attribute name="newattr" type="integer"/>
</complexType>
```

Deriving Types by Extension

- extend a base type's content by adding elements and/or attributes

```
<complexType name="newType" >
   <complexContent>
     <extension base="integer">
          <sequence>
        <element name="newelt" type="string"/>
        <attribute name="newattr" type="integer"/>
          </sequence>
      </extension>
  </complexContent>
</complexType>
```

Deriving Types by Extension

```
<complexType name="RelationType">
<simpleContent>
    <extension base="string">
        <attribute name="source" type="IDREF"/>
        <attribute name="target" type="IDREF"/>
    </extension>
</simpleContent>
</complexType>
```

Complex type is derived by extending a simple Type

```
<element name="Relation" type="RelationType">
```

Example XML:

```
<Relation source="edge1" target="edge2"> morph </Relation>
```

Example of MPEG-7 derived complex type

```
<!-- Definition of Creator datatype -->
<complexType name="CreatorType">
    <complexContent>
        <extension base="mpeg7:MediaAgentType">
            <sequence>
                <element name="Character"
    type="mpeg7:PersonNameType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="Instrument"
    type="mpeg7:CreationToolType" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<!-- Definition of MediaAgent datatype  -->
<complexType name="MediaAgentType">
    …
```

Creator is an extension of MediaAgent, with Character and Instrument

Deriving Types by Restriction

- Narrows the ranges or reduces alternatives of elements and/or attributes

```
<complexType name="myType">
      <element name="elt1" type="someType" minOccurs="0"/>
      <attribute name="attr1" type="string"/>
</complexType>

<complexType name="restrictedType" base="myType"
                                          derivedBy="restriction">
      <element name="elt1" type="someType" minOccurs="1"/>
      <attribute name="attr1" type="string" fixed="sausage"/>
</complexType>
```

Deriving Types by Restriction

- Narrows the ranges or reduces alternatives of elements and/or attributes

```
<simpleType name="US-State"
        <restriction base="string">
                        <enumeration value="AK"/>
                        <enumeration value="AL"/>
                        <enumeration value="AR"/>
        </restriction>
</simpleType>
```

Deriving Types by Restriction

- Simple Type Definitions

```
<simpleType name="6bitInteger" base="nonNegativeInteger">
    <minInclusive value="0" />
    <maxIncusive value="63" />
 </simpleType>

<simpleType name="DirectionType" base="string">
    <enumeration value="unidirectional" />
    <enumeration value="bi-directional" />
 <simpleType>
```

Preventing Type Derivations

- final= #all|extension|restriction|reproduction

  Prohibits derivations of type ‘final’

```
<complexType name=“myType” final=“restriction”>
      <element name=“elt1” type=“string”/>
      <element name=“elt2” type=“date”/>
….
</complexType>
```

Preventing Derived Type Substitution

- block = #all|extension|restriction|reproduction

  Prohibits replacements by any ‘block’ derived types in instances

```
<complexType name=“myType” block=“restriction”>
    <element name=“elt1” type=“string”/>
    <element name=“elt2” type=“date”/>
….
</complexType>
```

xsi:type

- In an instance document, we may need to indicate the derived type being used
- Suppose that there are several types derived (by extension) from VideoDoc, and some of the extended element declarations are optional. In the instance document, if xsi:type was not used, it could get very difficult for an XML Parser to determine which derived type is being used.

Schema:

```
<complexType name=”VideoDoc">
   <element name=“Title”…./>
   <element name=“Producer”…./>
   <element name=“Date”…./>
</complexType>
<complexType name=”NewsDoc" base=”VideoDoc" derivedBy="extension">
   <element name=“Broadcaster”…. maxOccurs=“0”/>
   <element name=“Time”….   maxOccurs=“0”/>
</type>
<element name=”VideoCatalogue">
   <complexType>
      <element name="CatalogueEntry" minOccurs="0" maxOccurs="*" type=”VideoDoc”/>
   </complexType>
</element>
```

This permits VideoDoc elements, as well as types derived from VideoDoc to be used as a child of VideoCatalogue, e.g., NewsDoc

Instance doc:

```
<CatalogueEntry xsi:type=”NewsDoc">
        <Title>”CNN 6 oclock News” </Title>
        <Producerr>David James</Author>
        <Date>1999</Date>
        <Broadcaster>CNN</Channel>
</CatalogueEntry>
```

Equivalence Classes

- Create an *exemplar* element and then create other elements which state that they are equivalent to the exemplar.
- The exemplar must be a global element
- Equivalent elements must be the same type or subtypes of the exemplar
- Often create equivalences to abstract elements to enable instantiation

Equivalence Example1

Schema:

```
<element name=”myD1" type="string"/>
<element name=”myD2" equivClass=”myD1" type="string"/>
<element name=”myD3">
    <complexType>
        <element ref=”myD1"/>
    </complexTtype>
</element>
```

Instance doc:

```
<myD3>
    <myD1>transcript</myD1>
</myD3>
```

Alternative Instance doc:

```
<myD3>
    <myD2>transcript</myD2>
</myD3>
```

Equivalence Example 2

```
<element name="myExemplar" type="myType"
                                        abstract="true"/>
<element name="element1" equivClass="myExemplar"/>
<element name="element2" equivClass="myExemplar"/>
<complexType name="newType">
    <element ref="myExemplar"/>
</complexType>
```

element1 and element2 are declared to be equivalent to the abstract element myExemplar.
Since myExemplar is abstract, instances of newType can only contain its equivalent classes - element1 or element2.

Schemas: Primary Components

- Abstract Elements and Types
  - Cannot appear in instantiations
  - Member of equivalence class must appear in instance document

<element> name="abstractElement" type="string' abstract="true" />

  - When the type is also abstract, must use xsi:type to indicate a derived type that is not abstract.

<complexType name="Vehicle" abstract="true" />
<complexType name="Car" base="Vehicle" />
<complexType name="Plane" base="Vehicle" />
<element name="transport" type="Vehicle"/>
<transport xsi:type="Car" />

Uniqueness

XML Schemas can specify:

- any attribute or element to be unique
- combinations of attribute and element values to be unique
- the range of the document over which something is unique

Defining Uniqueness

Need to specify

- a selector which is an Xpath expression to a parent element
- one or more fields - Xpath expressions to subelements or attributes within the selector

```
<unique name=”newsUnique">
        <selector>VideoCatalogue/NewsCatalogue/NewsDoc</selector>
        <field>Title</field>
        <field>@Category</field>
</unique>
```

The combination of the contents of Title plus the valueof the attribute Category must be unique throughout an XML instance document. The Title element and Category attribute lie within the XPath expression shown in the selector element.

unique vs key

- `key`: an element/attribute (or combination thereof) which is defined to be a key must
  - always be present (minOccurs must be greater than zero)
  - be non-nullable (i.e., nullable="false")
  - be unique
- `key` implies `unique`, but `unique` does not imply `key`

Defining a Key

Need to specify

- a selector which is an Xpath expression to a parent element
- fields - Xpath expressions to subelements or attributes

```
<key name=”newsKey">
      <selector>VideoCatalogue/NewsCatalogue/NewsDoc</selector>
      <field>Title</field>
      <field>@Category</field>
</key>
```

The combination of the contents of Title plus the value of the attribute Category is to be unique throughout an XML instance document. The Title element and Category attribute lie within the XPath expression shown in the selector element.

Defining a Reference to a Key

```
<keyref name="newsRef" refer="newsKey">
        <selector>Library/CheckoutRegister/newsItem</selector>
        <field>@titleRef</field>
        <field>@categoryRef</field>
<keyref>
```

The values in the two attributes, titleRef and categoryRef, combined, are a reference to a key defined by newsKey. The two attributes are located at the XPath expression found in the selector element.

i.e. for each value of newsRef (pair of titleRef and categoryRef values) there must be a newsKey with the same value

Note: if there are 2 fields in the key, then there must be 2 fields in the keyref, if there are 3 fields in the key, then there must be 3 fields in the keyref, etc. Further, the fields in the keyref must match in type and position to the key.

Specifying scope of uniqueness in XML Schemas?

•The key/keyref/unique elements may be placed anywhere in your schema.

•Where you place them determines the scope of the uniqueness.

•If placed at the top-level (direct child of the schema element), we are stating that in an instance document the uniqueness is with respect to the entire document.

•If we were to place the key/keyref elements as a child of an element then the uniqueness will have a scope of just the parent element. Thus, over the entire instance document there may be repeats, but within any instances of the parent element it will be unique.

Wildcards - “any”

- The *any* element allows any well-formed XML
- The *anyAttribute* element allows any attribute
- They are used to make extensible documents: we can add whatever element or attribute we want !
- *namespace* can be ##any|##other|##local|list

Schema

```
<element name=”open_element">
    <complexType>
        <any namespace=“##other”/>
        <anyAttribute namespace=“##local”/>
    </complexType>
</element>
```

Instance

```
<open_element size=“10000”>
    <comment>This could be anything
    </comment>
</open_element>
```

Notation declarations

- Specify a name, public id and a system id

Schema

```
<notation name="jpeg" public="image/jpeg" system="viewer.exe"/>
<element name="picture">
    ….<attribute name="pictype" type="NOTATION"/>
</element>
```

Instance

```
<picture pictype="jpeg">…..</picture>
```

Annotation

annotation element

- *documentation* subelement - schema description and copyright info
- *appinfo* subelement - information for tools, stylesheets, other applications

```
<annotation>
    <documentation>
        MPEG-7 description scheme for DSTC lectures
        Copyright DSTC Pty Ltd
    </documentation>
</annotation>
```

SchemaLocation

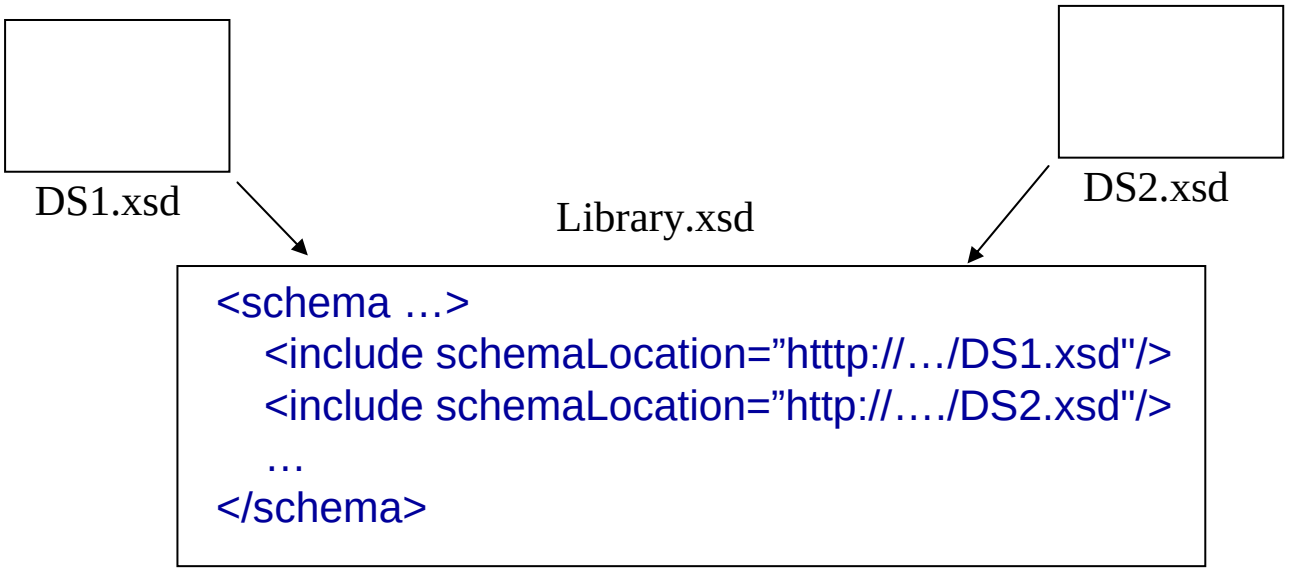
- Use in an instance document to provide parser with location of schema

xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
xsi:schemaLocation="http://…./MPEG7DS.xsd"

- *include schemaLocation* to merge the contents of other schemas into a schema
- *import namespace schemaLocation* to reference elements in a schema in a different namespace

include Element

- The include element allows you to include schema definitions from other schema
  - They must all have the same namespace
  - The net effect of include is as though you had typed all the definitions in directly into the containing schema



import Element

- The import element allows you to reference elements in another namespace

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

Include and import in MPEG-7

```
<!-- ISO/IEC 15938 Information Technology - Multimedia Content Description -->
<!-- Interface MPEG-7 Description Example developed by MPEG MDS Sub-group, -->
<!-- ################################################################### -->
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:mpeg7="urn:mpeg:mpeg7:sch
targetNamespace="urn:mpeg:mpeg7:schema:2001" elementFormDefault="qualified" attribut
  <annotation>
    <documentation>
      This document contains main wrapper of MPEG-7 schema
    </documentation>
  </annotation>
  <!-- ################################################################ -->
  <!--  import xml components                                           -->
  <!-- ################################################################ -->
  <import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="./xml-1
  <!-- ################################################################ -->
  <!--  include MPEG-7 specific extensions for DDL(ISO/IEC 15938-2)-->
  <!-- ################################################################ -->
  <include schemaLocation="./ddl-2001.xsd"/>
  <!-- ################################################################ -->
  <!--  include MPEG-7 Visual tools (ISO/IEC 15938-3)                   -->
  <!-- ################################################################ -->
  <include schemaLocation="./visual-2001.xsd"/>
  <!-- ################################################################ -->
  <!--  include MPEG-7 Audio tools (ISO/IEC 15938-4)                    -->
  <!-- ################################################################ -->
  <include schemaLocation="./audio-2001.xsd"/>
  <!-- ################################################################ -->
  <!--  include MPEG-7 MDS tools (ISO/IEC 15938-5)                      -->
  <!-- ################################################################ -->
  <include schemaLocation="./mds-2001.xsd"/>
</schema>
```

Built-in Datatypes

Note: 'T' is the date/time separator
INF = infinity
NAN = not-a-number

- Primitive Datatypes
  - string → "Hello World"
  - boolean → {true, false}
  - float → 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
  - double → 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
  - decimal → 7.08
  - timeDuration → 1Y2M3DT10H30M12.3S
  - recurringDuration → same format as timeDuration
  - binary → 010010111001
  - uriReference → http://www.somewhere.org
  - QNAME → book:part
  - ID → Token, must be unique
  - IDREF → Token which matches an ID
  - ENTITY
  - NOTATION

Built-in Derived Datatypes

- Derived datatypes
  - language
  - NMTOKEN
  - NMTOKENS
  - Name
  - NCName
  - IDREFS
  - ENTITIES
  - integer
  - non-negative-integer
  - positive-integer
  - non-positive-integer
  - negative-integer
  - date
  - time

- Subtype of primitive datatype
  - any valid xml:lang value, e.g., EN, FR, ...
  - "house"
  - "house barn yard"
  - "hello-there"
  - part (no namespace qualifier)
  - List of IDREF

  - 456
  - zero to infinity
  - one to infinity
  - negative infinity to zero
  - negative infinity to negative one
  - 1999-05-21
  - 13:20:00.000

Creating your own Datatypes

- You can create new datatypes from existing datatypes (called basetype) by specifying values for one or more of the optional *facets*
- Example. The string primitive datatype has nine optional facets - pattern, enumeration, length, maxlength, maxInclusive, maxExclusive, minlength, minInclusive and minExclusive.

Example

```
<simpleType name="TelephoneNumber" base="string">
        <length value="8"/>
        <pattern value="\d{3}-\d{4}"/>
</simpleType>
```

This creates a new datatype called 'TelephoneNumber'. Elements of this type can hold string values, but the string length must be exactly 8 characters long and the string must follow the pattern: ddd-dddd, where 'd' represents a 'digit'.

Facets of the Integer Datatype

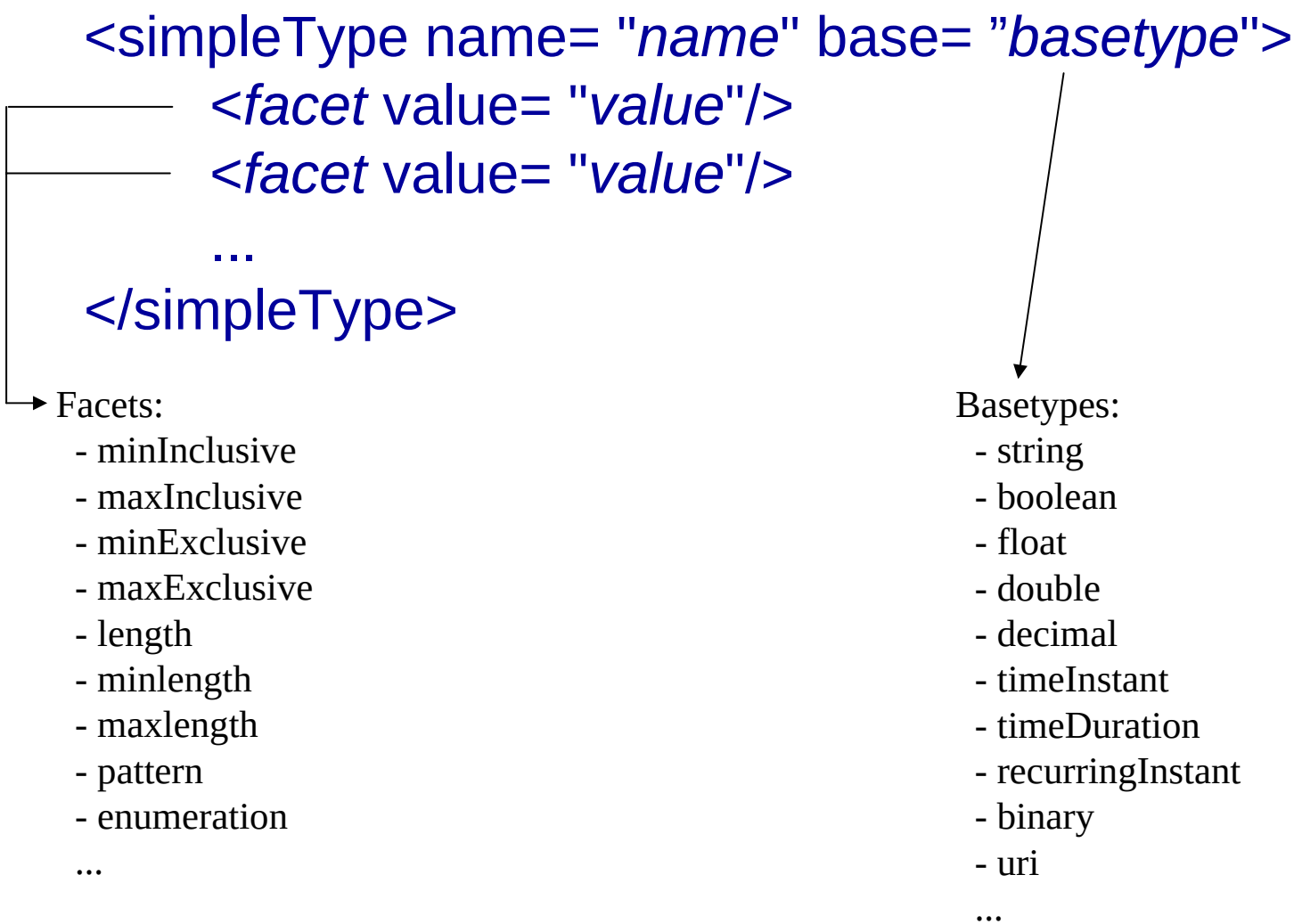
- Facets:
  - maxInclusive
  - maxExclusive
  - minInclusive
  - minExclusive

Example

```
<simpleType name= ”mySimpleType" base="integer">
        <minInclusive value="-1246"/>
        <maxInclusive value="29028"/>
</simpleType>
```

This creates a new datatype called ’mySimpleType'.
Elements of this type can hold an integer.
However, the integer must have a value
between -1246 and 29028, inclusive.

General Form of Datatype/Facet Usage



Regular Expressions

- The string datatype has a pattern facet. The value of a pattern facet is a regular expression.

Regular Expression	Example
- Chapter \d	- Chapter 1
- a*b	- b, ab, aab, aaab, …
- [xyz]b	- xb, yb, zb
- a?b	- b, ab
- a+b	- ab, aab, aaab, …
- [a-c]x	- ax, bx, cx

List datatype

- Finite-length sequence of an atomic datatype
- always derived
- set of literals separated by whitespace

```
<simpleType name="listOfInteger" base='integer' derivedBy='list'>
    <length value="5"/>
</simpleType>

<someElement xsi:type='listOfInteger'>
  1 2 3 4 5
</someElement>
```

MPEG-7 specific extensions (ISO/MPEG N4288)

- The following features have been added to the XML Schema Language specification in order to satisfy specific MPEG-7 requirements
  - Array and Matrix datatypes;
  - Built-in datatypes for basicTimePoint and basicDuration.
- Relies on mpeg7:dim (normative)

```
<!-- definition of mpeg7:dim attribute -->
<attribute name="dim">
      <simpleType>
            <restriction base="mpeg7:listOfPositiveIntegerForDim">
                  <minLength value="1"/>
            </restriction>
      </simpleType>
</attribute>
```

MPEG-7 specific extensions: Matrix

- Arrays and Matrices
  - Restrict the size of arrays and matrices
    - pre-defined or attribute bound
  - mpeg7:dimension facet
    - composed using list
    - specify the dimensions -parser generates array
  - 1D Arrays
    - uses mpeg7:dimension or length
    - default is arbitrary

MPEG-7 specific extensions: Matrix

```
<simpleType name="IntegerMatrix2D">
   <list itemType="integer">
      <annotation><appinfo>
         <mpeg7:dimension value="unbounded unbounded"/>
      </appinfo></annotation>
   </list>
</simpleType>

<simpleType name="IntegerMatrix3x4">
   <restriction base="mpeg7:IntegerMatrix2D">
      <annotation><appinfo>
         <mpeg7:dimension value="3 4"/>
       </appinfo></annotation>
    </restriction>
</simpleType>

<element name="IntegerMatrix3x4" type="mpeg7:IntegerMatrix3x4"/>
```

Schema

```
<IntegerMatrix3x4>
   8  9  4  5
   6  7  8  2
   2  1  3  5
</IntegerMatrix3x4>
```

Instance

Matrices and Arrays

- Another Example

```
<simpleType name="ListOfInteger" base="integer"  derivedBy="list" />

<complexType name="NDimIntegerArray" base="listOfInteger"
                              derivedby="extension">
      <attibute ref="mpeg7:dim" />
</complexType>

<element name="IntegerMatrix" type="NDimIntegerArray" />

<IntegerMatrix mpeg7:dim="2 4"> 1 2 3 4 5 6 7 8 </IntegerMatrix>
```

  - Note: matrix dimensions are specified in the instantiation !

Datatype Extensions

- Built-in Datatypes
  - unsignedInt1 (0-1)
  - unsignedInt3 (0-7)
  - unsignedInt5 (0-31)
  - unsignedInt6 (0-63)
  - unsignedInt7 (0-127)
  - unsignedInt8 (unsignedByte)
- Built-in Enumerated Datatypes
  - MimeType - IANA List of Mime Types
  - CountryCode - ISO3166
  - RegionCode - ISO3166
  - CurrencyCode - ISO4217
  - Character Set Code - IANA list of Character Sets

MPEG-7 specific extensions: basicTimePoint

```
<!-- ################################################ -->
<!--    Definition the basicTimePoint Datatype                -->
<!-- ################################################ -->

<simpleType name="basicTimePointType">
    <restriction base="string">
        <pattern value="(\-?\d+(\-\d{2}(\-\d{2})?)?)?(T\d{2}(:\d{2}(:\d{2}
                                (:\d+(\.\d{2})?)?)?)?)?(F\d+)?((\-
|\+)\d{2}:\d{2})?"/>
    </restriction>
</simpleType>
```

Examples are: YYYY-MM-DDThh:mm:ss:nnn.ffFNNN±hh:mm
For instance a timeInstant of T13:20:01.235 would be expressed using the basicTimePoint datatype by T13:20:01:235F1000.

MPEG-7 specific extensions: basicDurationPoint

```
<!-- ################################################ -->
<!--   Definition the basicTimePoint Datatype              -->
<!-- ################################################ -->

<simpleType name="basicTimePointType">
   <restriction base="string">
      <pattern value="(\-?P(\d+D)?(T(\d+H)?(\d+M)?
(\d+S)?(\d+N)?(\d{2}f)?)?(\d+F)?((\-|\+)\d{2}:\d{2}Z)?"/>
   </restriction>
</simpleType>
```

For instance a duration of 10 days would be expressed using the basicDuration datatype by P10T.

MPEG-7 Documents

- All MPEG-7 documents should have the following header information:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
       xmlns:xml="http://www.w3.org/XML/1998/namespace"
       xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 Mpeg7-2001.xsd">

<!-- Your MPEG-7 Metadata -->

</Mpeg7>
```

Credits

- Dr. Harald Kosch
- MPEG-7 XML Schema tutorial
- Steve Dennis
- Alejandro Jaimes
- Jane Hunter