

**PUBLIC
DELIVERABLE
ESPRIT 28798
AVIR**



**Deliverable # 6
Report on Description Scheme
Specifications**

AVIR

Audio Visual Indexing and Retrieval
for non-IT-expert users

1	INTRODUCTION	3
1.1	MAIN MPEG-7 CONCEPTS	3
1.2	MPEG-7 TERMINOLOGY	3
1.3	THE XML CONTEXT	4
1.3.1	XML overview.....	4
1.3.2	XML schemas overview	5
2	MAIN ORIENTATIONS OF MPEG-7	6
2.1	MPEG-7 MEETING ANALYSIS	6
2.1.1	Lancaster meeting recommendations	6
2.1.2	Seoul Output concerning the DDL.....	6
2.1.3	Vancouver Output concerning the DDL.....	7
2.2	THE CURRENT DDL REQUIREMENTS (W2859).....	7
2.3	IMPORTANT DDL ISSUES.....	8
2.3.1	Description Schemes, Descriptors and compound datatypes	8
2.3.2	Links and relations	8
3	AVIR DESCRIPTION SCHEMES DEFINITION LANGUAGE.....	9
3.1	AVIR DESCRIPTION SCHEMES DEFINITION LANGUAGE OVERVIEW	9
3.2	DDL SPECIFICATION	9
3.2.1	Prolog.....	11
3.2.2	Schema definition	11
3.2.3	Datatypes Definition.....	12
3.2.4	Definition of Attributes	14
3.2.5	Descriptors Definition	15
3.2.6	Writing a descriptor : a case study.....	15
3.2.7	Description Schemes Definition.....	18
3.2.8	Links and relations	20
3.3	EXTENSIONS OF THE CURRENT MPEG-7 DDL	20
3.3.1	A Tool for Mathematical expressions	20
3.3.2	Definition of relations.....	22
3.3.3	Semantic features in the DDL.....	23
3.4	AVIR DESCRIPTIONS SPECIFICATION.....	26
3.4.1	Descriptions consideration.....	26
3.4.2	Coding descriptions to preserve semantic	26
3.4.3	Locators in descriptions	31
4	MPEG-7 AND AVIR DESCRIPTION SCHEMES.....	34
4.1	AVIR MPEG-7 PROPOSALS ANALYSIS	34
4.2	FORMALIZATION OF AVIR DS	35
4.2.1	The TOCAI DS.....	35
4.2.2	Definition of 'AudioDS'.....	36
4.2.3	The VDX/ADX format.....	38
5	THE USE OF DS IN AVIR ARCHITECTURE	41
5.1	THE PARSING OF DESCRIPTION SCHEMES AND DESCRIPTIONS	42
5.1.1	Parsing Description Schemes Definitions	43
5.1.2	Parsing description using description schemes.....	44
5.2	AVIR DATABASES, DESCRIPTIONS AND DESCRIPTION SCHEMES	46
6	CONCLUSIONS.....	48
7	REFERENCES	49
8	ANNEX 1 – DOCUMENT TYPE DEFINITION OF THE MPEG-7 DDL	50

9 ANNEX 2 - DATATYPE MASKS..... 52

10 ANNEX 3 - REGULAR EXPRESSIONS 54

1 Introduction

The goal of this deliverable is to present the AVIR language for expressing metadata definition. This language tries to be as much conformant as possible to MPEG-7 requirements and current discussions. Unfortunately, the standard is far from being defined and we can only rely on what has already been done by the MPEG-7 committee. We managed to define a language which fits most of the MPEG-7 requirements and which is sufficiently stable to be used within the AVIR project. Description schemes written with it will be easily translated in the official MPEG-7 language when it will be stable.

In this chapter we give a simple overview of the field of audiovisual metadata. We then go into details by first presenting the recent steps of MPEG-7 activities. We introduce then AVIR Description Schemes Definition Language (DDL) and use it to formalize some description schemes used within the AVIR consortium.

1.1 Main MPEG-7 concepts

The main tools used to implement MPEG-7 descriptions are the Description Scheme Definition Language (DDL), Description Schemes (DSs), and Descriptors (Ds). Descriptors bind a feature to a set of attribute-value pairs. Description Schemes are models of the multimedia objects and of the universes that they represent. They specify the types of the descriptors that can be used in a given description, and the relationships between these descriptors or between other Description Schemes. Description Schemes can be defined by MPEG or by other organizations. MPEG agrees that new Description Schemes can be created when needed by any organization, and that resulting descriptions will be understandable by MPEG-7 decoders.

1.2 MPEG-7 Terminology

In order to clarify the terminology used in this document we briefly present the basic MPEG-7 definitions.

Data

Data is audiovisual information that will be described using MPEG-7, regardless of storage, coding, display, transmission, medium, or technology. This definition is intended to be sufficiently broad to encompass graphics, still images, video, film, music, speech, sounds, text and any other relevant AV medium. Examples for MPEG-7 data are: a MPEG-4 stream, a video tape, a CD containing music, sound or speech, a picture printed on paper, and an interactive multimedia installation on the web.

Feature

A Feature is an abstract concept characteristic of the data which signifies something to somebody. Some examples are: colour of an image, pitch of a speech segment, rhythm of an audio segment, camera motion in a video, style of a video, the title of a movie, the actors in a movie etc.

Descriptor

A Descriptor (D) associates a label for a Feature with one or many data type. An example might be: Color: string. The data type may be composite, meaning that it may be formed by concatenating multiple instances of data types. An example of this would be: RGB-Color: [int, int, int]. For a Descriptor to be functional in the standard, it must precisely define the semantics of the Feature, the associated data type, legal values, and an interpretation of the Descriptor Values (see the following point below).

It is possible to have several Descriptors representing a single Feature, i.e. to address different relevant requirements. Examples for multiple Descriptors for a single Feature are: enumerated lists, color moments and histograms for representing color.

Description Scheme

A Description Scheme (DS) specifies the structure and semantics of the relationships between its components. These components may be both Descriptors and other Description Schemes. The distinction between a Description Scheme and a Descriptor is that a Descriptor contains only basic data types or other Descriptors, as provided by the DDL, and does not refer to another (sub)DS.

Description

A Description is a DS (structure) and the set of Descriptor Values (instantiations) that describe the Data. Depending on the completeness of the set of Descriptor Values, the DS may be fully or partially instantiated. Whether or not the DS is actually present in the Description depends on technical solutions still to be provided.

Coded Description

A coded description is a Description that has been encoded to fulfil relevant requirements such as compression efficiency, error resilience, random access, etc. The coded descriptions are not presented in this document.

Description Definition Language

The Description Definition Language (DDL) will allow the creation of new Description Schemes and Descriptors and the extension and modification of existing Description Schemes.

1.3 The XML context

Basically, XML is a standard method for putting structured data in a text file. A video description is highly structured, and for that reason XML has been considered as one of the best candidate for the MPEG-7 Description scheme Definition Language (DDL).

1.3.1 XML overview

The Extensible Markup Language (XML) is a subset of SGML which goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. Since it was originally designed for encoding human-readable documents, it quickly attracted attention from groups interested in electronic commerce, interchange of data from relational and object-oriented databases, and other non-document applications.

An XML defines document structure and embed it directly within the document by the use of markups. A markup is composed of two kind of tags which encapsulate data : open tags and close tags. In a word XML is a sort of HTML where tags can be defined by the users.

Here is a simple example of an XML document :

Simple example of an XML document :

```
<letter>
  <header>
    <name>Mr John Smith</name>
    <address>
      <street>15 rue Lacedpede</street>
      <city>Paris</city>
    </address>
  </header>

  <text>Dear Mr Doe, .....</text>
</letter>
```

Note :

<letter> and </letter> are respectively the open tag and the end tag of the entire document. They specify that the included data describes or is a letter.

The XML 1.0 specification defines the concepts of well-formedness and validity. XML Well-formedness requires that document tags are correctly nested. XML validity requires that a document follow the constraints expressed in its document type definition (DTD), which provides the rough equivalent of a context-free grammar for a document type.

Here is an example of a basic DTD file :

```
Simple example of a DTD file :

<!DOCTYPE letter[
  <!ELEMENT letter (header, text)>
  <!ELEMENT header (name,address)>
  <!ELEMENT address (street, city)>

  <!ELEMENT name #PCDATA>
  <!ELEMENT street #PCDATA>
  <!ELEMENT city #PCDATA>
  <!ELEMENT text #PCDATA>
]>
```

The presented DTD defines :

- Nesting rules. ex : a letter is composed of an header and a text.
- Basic datatypes. ex : a name is a PCDATA (parsed character data)

Two kind of items can be defined in a DTD : elements and attributes. Basically elements are tags (name between angle brackets), while attributes are parameters of elements. In the following example two elements are defined : “book” and “novel”. The “Book” element has two attributes “title” and “type”, and the “novel” element has only one attribute “num” :

```
Example :

<book title="Sherlock Holmes" type="compilation">
  <novel num="1"> ... </chapter>
  <novel num="2"> ... </chapter>
</book>
```

At present, rules about what kinds of information can appear in an XML document can be expressed only in the form of XML document type definitions (DTDs). DTDs use a special format to define the rules for using XML markup for different kinds of documents, but in practice there are some common rules that cannot be expressed at all in DTD form. For example, mpeg-7 applications must communicate detailed information about the legal values of particular fields in the data being exchanged.

1.3.2 XML schemas overview

The main recommendation of the MPEG-7 AHG was to use an XML schema language for the DDL. The purpose of a schema is to define and describe a class of XML documents by using particular constructs to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content, attributes and their values. Schema constructs may also provide implicit information such as default values. Schemas document their own meaning, usage, and function. Schema can be seen as an extended DTD.

Even more important for many purposes is that XML schemas are themselves XML documents. Using XML as the document format for schemas, instead of using a special-purpose form as DTDs do, will allow users and developers of XML schemas to use standard XML tools, the same ones they use for other structured information, instead of having to shift to specialized tools for work on schemas. Any existing XML processor can read an XML schema; interchange will be easy.

Any application of XML can use the Schema formalism to express syntactic, structural and value constraints applicable to its document instances. The Schema formalism will allow a useful level of constraint checking to be described and validated for a wide spectrum of XML applications. For applications which require other, arbitrary or complicated constraints, the application must perform its own additional validations.

2 Main orientations of MPEG-7

2.1 MPEG-7 meeting analysis

2.1.1 Lancaster meeting recommendations

The Lancaster AHG agreed with the fact that XML-like languages (XML-schema or RDF) are the best candidates for the DDL. These languages have the ability to be easily parsable, well known and used, and particularly exhibit many needed features : object orientation, data types, etc. The Lancaster AHG recommendations defines that the DDL language class should be a syntactic language with semantic capabilities. The Lancaster proposal of DSTC's (p547) has been chosen as a good starting point for the DDL. Most features of one of the AVIR partner's proposal have also been held (p625 & m4586).

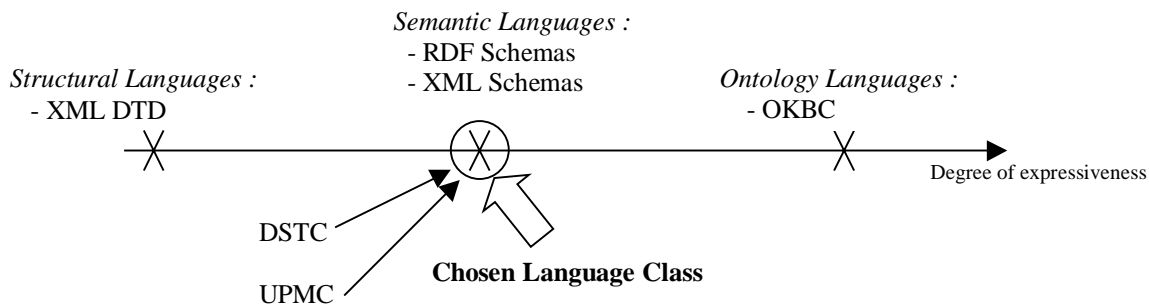


Figure 1 – Languages expressiveness

Lancaster discussions led to improve DDL requirements with the following recommendations :

- Description should be human readable
- Cardinality of relations between DS should be constrained
- Ability to specify constraints on attribute values of related elements (e.g. starttime<endtime)
- Ability to link descriptions, description parts and content
- Spatial datatype and coordinate space support
- Linking to Ontologies should be possible
- Real time support (generating or parsing it is not clear yet)
- Mapping of Ds and DSs to MPEG-4 BIF
- A mechanism or interface for linking from the DDL to remote procedural code

2.1.2 Seoul Output concerning the DDL

The DDL group and the Seoul WG11 meeting decided to keep in line with the Lancaster orientations on the most general aspects, but to design a full grammar written in BNF, in order to facilitate the implementation of a validating parser. A first BNF was written by the DDL group in three days with an important participation of UPMC. It is mainly based on the two following proposals :

- A revised proposal by University of Queensland, which includes an example DS with datatypes, constraints and coordinate spaces.
- UPMC contribution (AVIR project), which includes mechanisms for datatypes, constraints expression, coordinate spaces and links to ontologies and procedural code.

The proposal also made use of an existing grammar of XML, provided by the Diceman project. This grammar was then modified to include the specific object orientations of the semantic languages class.

A DDL Ad-Hoc Group was set at the end of the Seoul meeting. Its goals are the implementation of a version 1 DDL grammar in BNF, and the implementation of a validating parser by the Vancouver meeting. This parser can

be based on an existing XML parser, which can possibly be extended by its authors. The parser should preferably be made available in C or C++, to integrate in the XM (eXperimentation Model).

2.1.3 Vancouver Output concerning the DDL

During a DDL ad hoc group intermeeting which took place in Darmstadt, it has been decided that XML schema should be deeply analyzed and considered as a possible DDL. However due to some copyright reasons (W3C is the owner of the XML schema specification), XML schema use is delayed and the DDL group investigates their own DDL grammar. The evolution of XML Schema is monitored and the DDL groups should be ready to switch to an XML schema solution in the case of a consensus between W3C and MPEG.

The DDL group agreed on producing a new grammar by the beginning of September, and on providing a parser for this grammar before the Melbourne meeting. The DDL grammar described in this document is based on the latest produced grammar (7th September). UPMC is one of the two teams in charge of the parser development.

2.2 The current DDL Requirements (W2859)

The DDL requirements document has been updated given the Lancaster proposals and discussions :

1. **Compositional capabilities:** The DDL shall allow new DSs and Ds to be created and existing DSs to be modified or extended.
2. **Unique identification:** The DDL shall allow a unique identification of Ds and DSs. An example for unique identification is a namespace, which allows to qualify uniquely the element names and relationships and make these names recognizable to avoid name collisions on elements that have the same name but are defined in different vocabularies.
3. **Primitive data types:** The DDL shall provide a set of primitive data types, e.g. text, integer, real, date, time/time index, version, etc.
4. **Composite data types:** The DDL shall be able to describe composite data types such as histograms, graphs, rgb values, enumerated types etc.
5. **Multiple media types:** The DDL shall provide a mechanism to relate Ds to data of multiple media types of inherent structure, particularly audio, video, audio-visual presentations, the interface to textual description, and any combinations of these.
6. **Various types of DS instantiations:** The DDL should allow various types of DS instantiation: full, partial, full-mandatory, partial-mandatory.
7. **Relationships within a DS and between DSs:** The DDL shall be able to express spatial, temporal, structural, and conceptual relationships between the elements of a DS, and between DSs.
8. **Relationship between description and data:** The DDL shall supply a rich model for links and references between one or more descriptions and the data that it describes.
9. **Link to ontologies:** the DDL shall supply a linking mechanism between a description and several ontologies.
10. **Platform independence:** The DDL shall be platform and application independent.
11. **Grammar:** The DDL shall follow an unambiguous and easily parsed grammar.
12. **Validation of constraints - i.e. a parser shall be capable of validating the following:**
 - a. Values of properties
 - b. Structures
 - c. Related classes
 - d. Values of properties of related classes
13. **Intellectual Property Management:** The DDL shall provide a mechanism for the expression of Intellectual Property Management and Protection (IPMP) for description schemes and descriptors.
14. **Human readability:** The DDL shall allow that Ds and DSs can be read by humans.
15. **Real time support:** The DDL shall desirably provide features to support real time applications (database output like electronic program guides)

2.3 **Important DDL issues**

2.3.1 Description Schemes, Descriptors and compound datatypes

A very important issue which is not yet fully solved, is to determine what is the difference between a Description Schemes, a Descriptor and a compound datatype. The question is the following: "Should a descriptor be a structured element?". The descriptor "goal" is to map a name to a feature description, and many features are intrinsically structured. Then a descriptor can be structured. So from a syntactic point of view, what is the difference between a Description Scheme (by definition structured) and these "structured descriptor". Some suggestion have been made on the mpeg-7 DDL mailing list :

"to distinguish a D from a DS, we just ask: is it something about the representation of a feature, or is it about the structure of the description?" He pointed out that the structure of D and that of DS would be exploited by an application differently. Hence, "the structure of the Ds should be optimized for the feature representation and the structure of the DS should be optimized for the functionality it gives to the application".

"if a description entity is composed by various Ds (associated to features) it is a DS even if it can represent a feature"

"Every DS is structurally a D. And because nothing prevent us to have basic data types in DS, I guess, every D is also structurally a DS."

2.3.2 Links and relations

Links mechanisms are provided to localize distant resources, or other description elements. Relations are proposed to link elements that can be associated through some semantic. In the current version of the DDL, it is possible to embed description schemes (like XML element). In this case, what is the relation between a "contained" DS and its "container" DS ? As it is ambiguous, should we allow "contained" description scheme ?

3 AVIR Description schemes Definition Language

3.1 AVIR Description schemes Definition Language overview

The current version of the AVIR DDL is an XML schema which has been specifically designed for MPEG-7. It includes many of the features defined in the MPEG-7 requirements document : complex datatypes, constraints on feature values, coordinate spaces and links. It is based on the latest version of the DDL provided by Jane Hunter and Franck Nack and the milestone 3.

Description Schemes and Descriptors – The DDL allows to define Description Schemes. Description Schemes contains embedded description schemes and descriptors. Description schemes are in relation with other description schemes and can be linked to a set of content data.

Links – Links are used to define reference to the described multimedia object (the movie segment, the visual object, the sound, etc...).

Relations – Relations are used to define “semantic” relation between Description Schemes.

Locators – Locators are used to precisely define which parts of the description are in relations. Locators are used in description for expressing relations and in most of the semantic features of the DDL (constraints, coordinate spaces, etc...).

Datatypes – A set of basic datatypes is provided. New datatypes can be defined by "over constraining" basic datatypes. Basic datatypes can also be aggregated to create compound datatypes. Datatypes are subject to many discussion within the MPEG-7 DDL AdHoc Group. The main problem is to allow or not the definition of complex datatypes. Indeed datatypes with complex structure hide semantics. Our datatypes proposal are based on the XML schemas W3C datatypes.

Constraints and coordinate spaces - Mathematical expressions can be associated with DDL elements (descriptors, description schemes and links). They allow to define integrity constraints on descriptors and possible coordinate spaces translations. For that purpose we decided to use a subset of MathML. MathML is a w3c standard for exchanging mathematical expression on the web expressed in XML. Using MathML reduce description schemes legibility but their analysis is partially handled by XML parser. The use of MathML is still subject of debates.

MPEG-7 descriptions will have many different features and many associated mechanisms for expressing their semantics. The purpose of this deliverable is to present the current state of MPEG-7 and AVIR partners expectations. It offers a good starting point for understanding Description Definition Language main features which are still subject of many discussions. The standard will certainly change but those features are expected to remain the same. Furthermore, some of the required features are not yet developed. We propose then some early solutions knowing that slight changes may occur.

3.2 DDL specification

The basic AVIR-DDL features and syntax are described in this chapter. Before reading it be sure to be familiar with the XML notions of “element” and “attributes” (cf. 1.3.1).

The syntax is XML-like. A BNF grammar of the current language is given in annex. The presentation is made with a set of simple examples of DS. Consistent descriptions are presented. The main elements are precisely defined by Element Boxes like this.

Notation :

<i>Element name</i>	<i>Definition</i>	<i>Container</i>	<i>Contained Elements</i>	
The Element Name	<i>The definition of the element.</i>	<i>Its possible Containers</i>	<i>The contained elements. Expressions are composed of :</i> - <i>element names</i> - <i>' ' (means 'or')</i> - <i>'*' (means 'zero or more')</i> - <i>'+' (means 'one or more')</i> - <i>'?' (means 'zero or one')</i> - <i>',' (used to separate required elements)</i> - <i>'(' and ')'</i> (used to prioritization)	
<i>Attribute name</i>	<i>Definition</i>	<i>Attribute type</i>	<i>Occurrence</i>	<i>Default value</i>
<i>attribute</i>	<i>The definition of the attribute</i>	<i>The XML type of the attribute can be one of the following :</i> • <i>CDATA</i> • <i>ID</i> • <i>IDREF</i> • <i>ENTITY</i> • <i>ENTITIES</i> • <i>NMTOKEN</i> • <i>NMTOKENS</i> • <i>NOTATION</i> • <i>ENUMERATED</i>	<i>Specifies if the attribute is required or not.</i>	<i>Specifies a default value when the attribute is not present.</i>

Example:

The following boxes :

Element name	Definition	Container	Contained Elements	
Book	An element to define what is a book	none	author,chapter+	
Attribute name	Definition	Attribute type	Occurrence	Default value
title	The title of the book	CDATA	required	

Element name	Definition	Container	Contained Elements	
Author	The name of an author	Book		
Attribute name	Definition	Attribute type	Occurrence	Default value

Element name	Definition	Container	Contained Elements	
Chapter	A chapter itself	Book		
Attribute name	Definition	Attribute type	Occurrence	Default value
num	The number of the chapter	CDATA	required	

validates the following XML file :

```
<book title="Madame Bovary">
  <author>Stendhal</author>
  <Chapter num="1">
    .....
  </Chapter>
  <Chapter num="2">
    .....
  </Chapter>
</book>
```

The proposed grammar is consistent with the MPEG-7 DDL provided the 7th September by members of the DDL ad-hoc Group.

3.2.1 Prolog

MPEG-7 description schemes definition files should begin with a declaration which specifies the version of XML being used. This tag is called the document prolog.

```
Example

<?xml version='1.0' ?>
```

3.2.2 Schema definition

The main document is included between "Schema" tags.

This tag contains the version of the schema and some namespaces attributes. The namespace facility enables the inclusion of multiple namespaces. It enables the same feature to have different descriptors which correspond to different domains or description schemes. The ability to mix vocabularies allows video authors or others to deliver richer domain-specific content descriptions by increasing the accessibility and re-usability of video content on the Web. The proposed MPEG-7 schema can also be included in other schemas or descriptions using this same facility. The definition is similar to XML namespaces defined in [8].

Element name	Definition	Container	Contained Elements	
Schema	The definition of the schema		(DSType DType AttrGroup Datatype)*	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the schema	CDATA	required	
version	The version of the schema	CDATA	required	
xmlns	Namespaces used in this schema	URI	optional	

```
Example :

<?xml version = "1.0"?>

<Schema
  xmlns: dc="http://purl.org/metadata/dublin_core_dstd#"
  xmlns: sync="http://cselt.it/mpeg/mpeg7ds/sync#"
  xmlns: dcq="http://purl.org/metadata/dublin_core_qualifiers#"
  xmlns: smil="http://www.w3c.org/TR/WD-smil#"

  <!-- ... Definitions ... -->

</Schema>
```

3.2.3 Datatypes Definition

Descriptors are data typed elements. A basic set of datatypes called “primitive datatypes” is defined. Other datatypes can be created by constraining basic datatype values. The primitive set of datatypes is the following :

- **NMToken** : a “name” token
- **boolean** : a true/false value
- **real** : a real number
- **number** : an integer number
- **binary** : a sequence of bits
- **dateTime** : a date
- **URI** : an Universal Resource Identifier
- **language** : a language name
- **string** : a Sequence of characters

The main element:

Element name	Definition	Container	Contained Elements	
Datatype	Embeds the definition of this new datatype.	Schema	BaseType, Length MaxLength MinLength MinInclusive MaxInclusive MinExclusive MaxExclusive Precision Scale Enumeration LexicalRepresentation	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of this new datatype.	CDATA	required	

The reference datatype:

Element name	Definition	Container	Contained Elements	
BaseType	The associated primitive datatype.	Datatype		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of this new datatype.	Enumerated : - NMToken - boolean - real - number - binary - dateTime - URI - language - string	required	

The “constraints” elements can be classified in three categories.

The elements of the first category are used to constraint length of strings :

- **MinLength, MaxLength** : This elements are used to constrain length of bits or character strings (binary or string).
- **Length** : This element constraints the length of strings.

```

Examples :
<Datatype name="french_word">
  <BaseType name="string"/>
  <MinLength>1</MinLength>
  <MaxLength>25</MaxLength> <!-- anticonstitutionnellement -->
</Datatype>
    
```

The second category aims at constraining values of numbers :

- **MinInclusive, MaxInclusive, MinExclusive, MaxExclusive** : This elements are used to constrain value of a number (integer or real).
- **Enumeration** : The value should be chosen among a set of possible values.
- **Precision** : The total number of decimal digits.
- **Scale** :The number of decimal digits after the decimal point.

```
Examples :

<Datatype name="age">
  <BaseType name="integer"/>
  <MinInclusive>0</MinInclusive>
  <MaxInclusive>120</MaxInclusive>
</Datatype>

<Datatype name="color_name">
  <BaseType name="string"/>
  <Enumeration>
    <literal>red</literal>
    <literal>green</literal>
    <literal>blue</literal>
    <literal>yellow</literal>
    <literal>black</literal>
    <literal>white</literal>
  </Enumeration>
</Datatype>

<Datatype name="french_postalcode">
  <BaseType name="string"/>
  <Length>5</Length/>
</DatatypeDef>
```

The third category aims at constraining shapes of strings :

- **LexicalRepresentation** : Allows to specify the format (or shape) of strings. Format are described using masks or regular expression. Masks and regular expression grammars are defined in annex.

```
Examples :

<Datatype name="better_french_postalcode">
  <BaseType name="Number">
    <!-- means 5 numeric characters -->
    <LexicalRepresentation>#####</LexicalRepresentation>
  </Datatype>

<Datatype id="creditcardnumber_mask">
  <BaseType name="string">
    <LexicalRepresentation>
      <Lexical>#####[.]#####[.]#####[.]#####</Lexical>
    </LexicalRepresentation>
  </Datatype>

<Datatype id="creditcardnumber_regexp">
  <BaseType name="string">
    <LexicalRepresentation>
      <Lexical>\d{4}(\.\d{4}){3}</Lexical>
    </LexicalRepresentation>
  </Datatype>

<Datatype id="myDate">
  <BaseType name="dateTime">
    <LexicalRepresentation>
      <Lexical>CCYY-DD</Lexical>
      <Lexical>CCYY-MM-DD</Lexical>
    </LexicalRepresentation>
  </Datatype>
```

Referencing a datatype

The language need to be able to reference a datatype. This can be done with the “DatatypeRef” tag.

Element name	Definition	Container	Contained Elements	
DatatypeRef	Reference of a datatype	DType, AttrDecl		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the referenced datatype	CDATA	required	

3.2.4 Definition of Attributes

Group of attributes (in the XML sense of the word) can be defined. They are used to parameterized Description Scheme and Descriptor instances. Note that the attributes are datatyped. This datatyping is done using the DatatypeRef tag.

Element name	Definition	Container	Contained Elements	
AttrGroup	Group of attributes definition.	Schema	AttrDecl+	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the group	CDATA	required	

Element name	Definition	Container	Contained Elements	
AttrDecl	Attribute definition.	AttrGroup, DType, DSType	DatatypeRef, default	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the attribute	CDATA	required	
required		enumerated : - yes - no	optional	

Element name	Definition	Container	Contained Elements	
default	Default value of an attribute.	AttrDecl		
Attribute name	Definition	Attribute type	Occurrence	Default value

Example :

```
<AttrGroup name="BasicParams">
  <AttrDecl name="id" required="yes">
    <DatatypeRef name="String"/>
  </AttrDecl>
  <AttrDecl name="num" required="no">
    <DatatypeRef name="integer"/>
    <default>0</default>
  </AttrDecl>
</DType>
```

Referencing a group of attributes

A group of attributes can be referenced within descriptors and description schemes definition.

Element name	Definition	Container	Contained Elements	
AttrGroupRef	Reference to a group of attributes	DSType, DType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the referenced datatype	CDATA	required	

3.2.5 Descriptors Definition

A descriptor is an association between a feature and the way to represent it. For instance descriptors for colors, shapes or sound brilliance can be defined. In this version of the DDL, descriptors can be composed of other descriptors in order to defines “structured” or “compound” descriptors. Descriptors can have attributes (XML attributes).

Element name	Definition	Container	Contained Elements	
Dtype	Descriptor definition	Schema	(AttrDecl AttrGroupRef)* , SubDOF? , (DatatypeRef DTypeRef sequence choice all)+	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	the name of the descriptor	CDATA	required	

Inheritance

In order to improve the reuse of preexisting Descriptors, an inheritance mechanism is available.

Element name	Definition	Container	Contained Elements	
SubDOF	For inheritance	DType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The identifier of the parent description scheme	CDATA	required	

Referencing a Descriptor Definition

A descriptor can be referenced within descriptors and description schemes definition.

Element name	Definition	Container	Contained Elements	
DtypeRef	Reference to a Descriptor	DSType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the referenced descriptor	CDATA	required	
minOccur	The min number of allowed occurrence of the ‘name’ DS within the current DS. The value should be number.	CDATA	optional	
maxOccur	The max number of allowed occurrence of the ‘name’ DS within the current DS. The value of this attribute should be a number.	CDATA	optional	
minOccurPar	The min number of allowed occurrence of the ‘name’ DS within the current DS. The value of this attribute is a parameter name. The parameter name is the name of an attribute.	CDATA	optional	
maxOccurPar	The min number of allowed occurrence of the ‘name’ DS within the current DS. The value of this attribute is a parameter name. The parameter name is the name of an attribute.	CDATA	optional	

3.2.6 Writing a descriptor : a case study

In this chapter we will show many way to define a very simple descriptor for colors. The requirements are the following : the color should be composed of three components (red, green, blue), each of the component value should be between 0 and 256.

The first example shows a string based representation of the color.

```
Descriptor definition :  
  
<Datatype name="colorStr">  
  <BaseType name="String"/>  
  <LexicalRepresentation>###;###;###<LexicalRepresentation>  
</Datatype>  
  
<DType name="Color">  
  <DatatypeRef name="color_string"/>  
</DType>  
  
Example of valid descriptions :  
  
<!-- which one is the red component ? -->  
<color>120;200;99<color>  
<!-- The value are not checked -->  
<color>600;800;999<color> <!-- possible -->
```

This solution is simple and clear. Unfortunately it has two main drawbacks. First, the value of the component are not checked against their possible value (between 0 and 256). Second, it is not clear which component is the red, the green or the blue one.

The second example uses a compound descriptor, composed of three "colorComponent" descriptors.

```
Descriptor definition :  
  
<Datatype name="int256">  
  <BaseType name="integer"/>  
  <MinInclusive>0</MinInclusive>  
  <MaxExclusive>256</MaxExclusive>  
</Datatype>  
  
<DType name="colorComponent">  
  <DatatypeRef name="int256"/>  
</DType>  
  
<DType name="color_Dtype">  
  <DTypeRef name="colorComponent">  
  <DTypeRef name="colorComponent">  
  <DTypeRef name="colorComponent">  
</DType>  
  
Example of a valid description :  
  
<!-- which one is the red component ? -->  
<color_DType>  
  <colorComponent>200</colorComponent>  
  <colorComponent>120</colorComponent>  
  <colorComponent>30</colorComponent>  
</color_DType>
```

The same result can be obtained by using the "minOccur" and "maxOccur" attributes :

```
Same example :  
  
...  
  
<DType name="color_Dtype">  
  <DTypeRef name="colorComponent" minOccur="3" maxOccur="3"/>  
</DType>  
  
...
```

The third example uses attributes to disambiguate the "colorComponent".

```
Descriptor definition :

<Datatype name="int256">
  <BaseType name="integer" />
  <MinInclusive>0</MinInclusive>
  <MaxExclusive>256</MaxExclusive>
</Datatype>

<Datatype name="possibleComponents">
  <BaseType name="string" />
  <Enumeration>
    <literal>red</literal>
    <literal>green</literal>
    <literal>blue</literal>
  </Enumeration>
</Datatype>

<DType name="colorComponent">
  <AttrDecl name="compName" required="yes">
    <DatatypeRef name="possibleComponents" />
  </AttrDecl>
  <DatatypeRef name="int256" />
</DType>

<DType name="color_Dtype">
  <DTypeRef name="colorComponent">
  <DTypeRef name="colorComponent">
  <DTypeRef name="colorComponent">
</DType>

Example of valid descriptions :

<color_DType>
  <colorComponent compName="red">200</colorComponent>
  <colorComponent compName="blue">30</colorComponent>
  <colorComponent compName="green">120</colorComponent>
</color_DType>

<!-- Three red components -->
<color_DType>
  <colorComponent compName="red">200</colorComponent>
  <colorComponent compName="red">30</colorComponent>
  <colorComponent compName="red">120</colorComponent>
</color_DType>
```

In this example the value of each component are checked. There is still a drawback, there is no way to express that we need one of each component.

The fourth example uses the inheritance mechanism of Descriptor to define red, green and blue components from a "colorComponent" Descriptor.

```
Descriptor definition :

<Datatype name="int256">
  <BaseType name="integer" />
  <MinInclusive>0</MinInclusive>
  <MaxExclusive>256</MaxExclusive>
</Datatype>

<DType name="colorComponent">
  <DatatypeRef name="int256" />
</DType>

<DType name="red">
  <SubDOf name="colorComponent" />
</DType>

<DType name="green">
  <SubDOf name="colorComponent" />
</DType>
```

```

<DType name="blue">
  <SubDOf name="colorComponent" />
</DType>

<DType name="color">
  <DTypeRef name="red" />
  <DTypeRef name="green" />
  <DTypeRef name="blue" />
</DType>

Example of a valid description :

<color>
  <red>120</red>
  <green>120</green>
  <blue>120</blue>
</color/>
    
```

This last example is verbose but semantically correct.

3.2.7 Description Schemes Definition

Description schemes are composed of reference to descriptor definitions (the "DTypeRef" tag), reference to description scheme definitions (the "DSTypeRef" tag), links (the "Link" tag) and relations (the "Relation" tag). These elements can be grouped within containers (the "sequence", "all" and "choice" tags). Finally, some consistency constraints can apply on description schemes elements.

Inheritance

In order to improve the reuse of preexisting Description Schemes, an inheritance mechanism is available. The inheritance is a transitive relation between DS. A inherited DS has all the descriptors, relations and nested description schemes of its parents.

Element name	Definition	Container	Contained Elements	
DSType	For defining description schemes	Schema	(AttrDecl AttrGroup), SubDSOf?, (DSTypeRef DTypeRef all sequence choice)*	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The identifier of the description scheme	CDATA	required	
abstract	An attribute that specifies whether a DS can be instantiated or not	enumerated : - yes - no	optional	yes

Element name	Definition	Container	Contained Elements	
SubDSOf	For inheritance	DSType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The identifier of the parent description scheme	CDATA	required	

Let's see an example of DS definitions :

```

Example :

<!-- a very simple datatype for timecode -->
<Datatype name="timecode_s">
  <BaseType name="real" />
  <precision>2</precision>
</Datatype>
    
```

```

<!-- descriptors -->
<DType name="startTC">
  <DatatypeRef name="timecode_s"/>
</DType>

<DType name="endTC">
  <DatatypeRef name="timecode_s"/>
</DType>

<!-- a MovieGranule is a slice of the video -->
<DSType name="MovieGranule">
  <DtypeRef name="startTC"/>
  <DtypeRef name="endTC"/>
</DSType>

<!-- a shot is a movie granule -->
<DSType name="Shot">
  <SubDSOf name="MovieGranule"/>

  <!-- place here a definition of your shot description -->

</DSType>
    
```

You can include description schemes within a description scheme :

```

Example :

<!-- definition of an object -->
<DSType name= "object">
  <DtypeRef name="color"/>
</DSType>

<!-- a shot is a movie granule which contains objects -->
<DSType name="Shot">
  <SubDSOf name="MovieGranule"/>

  <!-- object DS is embedded in the Shot DS -->
  <DSTypeRef name="object" minOccurs="0" maxOccurs="*" />

</DSType>
    
```

Referencing a Descriptor Definition

A descriptor can be referenced within descriptors and description schemes definition.

Element name	Definition	Container	Contained Elements	
DSTypeRef	Reference to a Description Scheme	DSType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The name of the referenced description scheme	CDATA	required	
minOccur	The min number of allowed occurrence of the 'name' DS within the current DS. The value should be number.	CDATA	optional	
maxOccur	The max number of allowed occurrence of the 'name' DS within the current DS. The value of this attribute should be a number.	CDATA	optional	
minOccurPar	The min number of allowed occurrence of the 'name' DS within the current DS. The value of this attribute is a parameter name. The parameter name is the name of an attribute.	CDATA	optional	
maxOccurPar	The min number of allowed occurrence of	CDATA	optional	

	the 'name' DS within the current DS. The value of this attribute is a parameter name. The parameter name is the name of an attribute.			
--	---	--	--	--

3.2.8 Links and relations

The current version of the DDL doesn't propose a precise enough definition of links and relations. We suggest to use relations defined in chapter 3.3.2.

3.3 Extensions of the current MPEG-7 DDL

The following chapter defines mechanisms not yet defined in MPEG7 current DDL grammar.

3.3.1 A Tool for Mathematical expressions

Many DDL features (constraints, coordinate space translation) require mathematical expressions. In order to write those mathematical expressions we propose to use a subset of MathML [10]. This allows to use a DTD parser for the expression syntax validation.

We use a subset of the content category of MathML markups. MathML content markup consists of about 75 elements accepting roughly a dozen attributes. The majority of these elements are empty elements corresponding to a wide variety of operators, relations and named functions. Examples of this sort include *partialdiff*, *leq* and *tan*. Others such as matrix and set are used to encode various mathematical data types, and a third, important category of content elements such as *apply* are used to make new mathematical objects from others. We selected 22 MathML tags, which seems to be sufficient for our purpose.

The *apply* element is perhaps the single most important content element. It is used to apply a function to a collection of arguments. The positions of the child schemata is again significant, with the first child denoting the function to be applied, and the remaining children denoting the arguments of the function, with order preserved. Note that the *apply* construct always uses prefix notation, like the programming language LISP. In particular, even binary operations like subtraction are marked up by applying a prefix subtraction operator to two arguments. For example, $a - b$ would be marked up as :

```

Example : (a - b)

<apply>
  <minus/>
  <ci>a</ci>
  <ci>b</ci>
</apply>
  
```

The *reln* element is used much like the *apply* element, except that it is used with relations instead of operators and functions :

```

Example : x^2 + 4*x + 4 = 0

<reln>
  <eq/>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <apply>
      <times/>
      <cn>4</cn>
      <ci>x</ci>
    </apply>
    <cn>4</cn>
  </apply>
  <cn>0</cn>
</reln>
  
```

</reln>

Writing MathML expressions

Content expression trees are built up from basic mathematical objects. At the lowest level, "leaf nodes," are encapsulated in non-empty elements that define their type. Numbers and symbols are marked by the token elements `cn` and `ci`. (Ex : `<cn>12345</cn>` and `<ci>x</ci>`, represent the mathematical number "12345" and the variable "x").

The most fundamental way of building up a mathematical expression in MathML content markup is the *apply* construct. An *apply* element typically applies an operator to its arguments. It corresponds to a complete mathematical expression. Roughly speaking, this means a piece of mathematics which could be surrounded by parentheses or "logical brackets" without changing its meaning. For example, $(x + y)$ might be encoded as

```
<apply><plus/> <ci> x </ci> <ci> y </ci> </apply>
```

The opening and closing tags of *apply* specify exactly the scope of any operator or function. The most typical way of using *apply* is simple and recursive. Symbolically, the content model can be described as:

```
<apply> op a b </apply>
```

Another construction closely related to the use of the *apply* with operators and arguments involves the *reln* element. The *reln* element is used to denote that a mathematical relation holds between its arguments, as opposed to applying an operator. Thus, the MathML markup for the expression $x < y$ is given by:

```
<reln>
  <lt/>
  <ci> x </ci>
  <ci> y </ci>
</reln>
```

The authorized subset of MathML tag is given in the following table :

Basic constructs tags		
Tokens	ci, cn	Token element used to represent numbers and variables
Mathematical expression	apply	
Mathematical relation	reln	
Arithmetic tags		
Unary arithmetic	abs	
Binary arithmetic	divide, minus	
N-ary arithmetic	plus, times, max, min	
Logical tags		
Unary logical	not	
N-ary logical	and, or, xor	
Relation tags		
Binary relation	neq	
N-ary relation	eq, leq, lt, geq, gt	
Conditional expression		
Condition	condition	To express conditional expression (typically used within constraints).

Parsing MathML expressions

MathML is an application of XML, or Extensible Markup Language, and as such, its syntax is governed by the rules of XML syntax, and its grammar is in part specified by a DTD, or Document Type Definition. MathML also specifies some syntax and grammar rules in addition to the general rules it inherits as an XML application. These rules allow MathML to encode a great deal more information than would ordinarily be possible with pure XML, without introducing many more elements, and using a substantially more complex DTD. Of course, one drawback to using MathML specific rules is that they are invisible to generic XML processors and validators.

There are basically two kinds of additional MathML grammar and syntax rules. One kind involves placing additional criteria on attribute values. For example, it is not possible in pure XML to require that an attribute value be a positive integer. The second kind of rule specifies more detailed restrictions on the child elements (for example on ordering) than are given in the DTD. For example, it is not possible in XML to specify that the first child be interpreted one way, and the second in another. We do not take into account those specific MathML rules, since they don't have practical use in our purpose.

3.3.2 Definition of relations

Relations aims at linking parts of descriptions.

The definition of a relation type is made using the "RelationType". A RelationType can extend an existing RelationType by overconstraining its range and domain. Consistency constraints can be expressed on source and target values.

Element name	Definition	Container	Contained Elements	
RelationType	Definition of a relation type	Schema	SubRelationOf?,domain?, range?	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	name of the relation type	CDATA	required	
direction	the direction of the link : uni, bi	enumerated: - uni - bi	optional	uni
InverseRelation	the name of the inverse relation	IDREF	optional	

Element name	Definition	Container	Contained Elements	
SubRelationOf	For inheritance	RelationType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The identifier of the extended RelationType	CDATA	required	

Element name	Definition	Container	Contained Elements	
Source	The source of a relation type.	RelationType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The type of the source element. This type should be the name of a DS because only DS can contains relations.	CDATA	required	

Element name	Definition	Container	Contained Elements	
Target	The target of a relation type.	RelationType		
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The type of the relation target. This type should be a DS.	CDATA	required	

Referencing a relation

Element name	Definition	Container	Contained Elements	
RelationRef	Reference to a relation type	DSType		
Attribute name	Definition	Attribute type	Occurrence	Default value

name	the identifier of the relation	ID	required	
occurs	the occurrence of the relation.	enumerated: - optional - required	optional	required

3.3.3 Semantic features in the DDL

Constraints

Constraints are conditions that the description should satisfies in order to be consistent. These conditions are expressed using the MathML subset proposed in 3.3.1. Using constraints improves the quality of descriptions and offers a mechanism needed for most of the MPEG-7 applications.

Constraints can be put within DType, DStype or RelationType. We need new definitions of DType, DStype and RelationType:

Element name	Definition	Container	Contained Elements	
Dtype	Descriptor definition	Schema	(AttrDecl AttrGroupRef)* , SubDOF? , (DatatypeRef DTypeRef sequence choice all)+, constraint*	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	the name of the descriptor	CDATA	required	

Element name	Definition	Container	Contained Elements	
DStype	For defining description schemes	Schema	(AttrDecl AttrGroup), SubDSOf?, (DStypeRef DTypeRef all sequence choice)*, constraint*	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	The identifier of the description scheme	CDATA	required	
abstract	An attribute that specifies whether a DS can be instantiated or not	enumerated : - yes - no	optional	yes

Element name	Definition	Container	Contained Elements	
RelationType	Definition of a relation type	Schema	SubRelationOf?,domain?, range?, constraint*	
Attribute name	Definition	Attribute type	Occurrence	Default value
name	name of the relation type	CDATA	required	
direction	the direction of the link : uni, bi	enumerated: - uni - bi	optional	uni
InverseRelation	the name of the inverse relation	IDREF	optional	

Element name	Definition	Container	Contained Elements	
constraint	Consistency constraints.	RelationType, DStype	condition (from MathML valid subset)	
Attribute name	Definition	Attribute type	Occurrence	Default value
NONE				

The notion of locators

In the case of constraints on occurring in Description Scheme or Descriptor definition, the descriptor values should be referenced. For that purpose we introduce new tags called Locator tags. They define a way (called a "reaching path") to reach the desired value i.e. the value which is used in the constraint expression. The notion of reaching path is developed in chapter 3.4.3.

Definition :

- The tag "Locator" embeds the reaching path.
- The tag "Dvalue" is used for reaching the value of a descriptor.

In the case of a RelationType constraints we propose two Locator tags to determine which of the source descriptors or the target descriptors should be used : source and target

Example :

```
<Datatype name="timecode_s">
  <BaseType name="real"/>
</Datatype>

<DType name="TC">
  <DatatypeRef name="timecode_s"/>
</DType>

<DType name="startTC">
  <BaseType name="TC"/>
</DType>

<DType name="endTC">
  <BaseType name="TC"/>
</DType>

<!-- a related event is a link which associates a MovieGranule and an event -->
<RelationType name="related_event">
  <source name="MovieGranule">
    <target name="event">
</RelationType>

<!-- an event is something happening at a certain time -->
<DSType name="event">
  <DTypeRef name="eventtype"/>
  <DTypeRef name="TC"/>
</DSType>

<!-- a MovieGranule is a slice of the video -->
<DSType name="MovieGranule">
  <DTypeRef id="startTC"/>
  <DTypeRef id="endTC"/>
  <RelationRef name="consequence"/>

  <constraint>
    <condition> <!-- startTC < endTC -->
      <reln>
        <lt/> <!-- lower than -->
        <ci>
          <Locator>
            <Dvalue name="startTC"/>
          </Locator>
        </ci>
        <ci>
          <Locator>
            <Dvalue name="endTC"/>
          </Locator>
        </ci>
      </reln>
    </condition>
  </constraint>
</DSDef>
```

```
<!-- a consequence is an event that occurs after the end of the movie granule -->
<RelationType id="consequence">
  <SubRelationOf name="related_event"/>
  <constraint>
    <condition> <!-- source.startTC < target.TC -->
      <reln>
        <lt/>
        <ci>
          <Locator>
            <Source/> <!-- MovieGranule -->
            <DValue name="endTC"/>
          </Locator>
        </ci>
        <ci>
          <Locator>
            <Target/> <!-- event -->
            <DValue name="TC"/>
          </Locator>
        </ci>
      </reln>
    </condition>
  </constraint>
</ RelationType >
```

Coordinate spaces

As stated at the Lancaster meeting, the DDL should provide possibilities to define new data types. We propose to add a conversion formalism between different units. In the enumeration of the different possible tokens, a referent is defined, and for each token, the parameters of the conversion are defined.

```
Example :

<Datatype name="DistanceUnit">
  <BaseType name="nmtoken"/>
  <Enumeration>
    <literal>miles</literal>
    <literal>kilometer</literal>
  </Enumeration>
</Datatype>

<Datatype name="DistanceType">
  <BaseType name="real"/>
</Datatype>

<DType name="Distance">
  <DTypeRef name="Distance"/>
  <DTypeRef name="DistanceUnit">
    <conversion name="DistanceUnit">
      <reference>miles</reference>
      <!-- conversion depends on the DistanceUnit value -->
      <!-- conversion will change the numericValue and the DistanceUnit value -->
      <from unit="kilometer">
        <reln>
          <times/>
          <cn>1.8</cn>
        </reln>
        <ci>
          <Locator>
            <DValue name="DistanceType"/>
          </Locator>
        </ci>
      </from>
      <to unit="kilometer">
        <reln>
          <times/>
          <cn>0.56</cn>
        </reln>
        <ci>
          <Locator>
            <DValue name="DistanceType"/>
          </Locator>
        </ci>
      </to>
    </conversion>
  </DTypeRef>
</DType>
```

```
</to>  
</conversion>  
</DSDef>
```

This feature needs to be further analyzed.

Procedural code

It is possible to reference procedural code information within description schemes. Typical use of procedural code is computation of similarity measures since it can be a complex operation. Procedural code is also useful at the indexing phase. It can give the extracting method, specify which procedural code should be used for extracting the descriptor if needed.

We proposed a way to reference a code using a particular tag named *ProceduralCode*. For each description, a procedural code can be referenced using a *proceduralCodeDescriptor*. We define here several properties but the list is not exhaustive (it can eventually be completed by a DS):

- *signature*: allows authentication of code
- *location* : URL (where, you can find the procedural code)
- *parameter* : the sequence of parameters

This feature needs to be further analyzed.

Ontology interfaces

A description is independent of any model of ontological representation. But, we want to be able to translate a description in order to be used by an ontology. In that purpose, the parameters of the translation is defined in the DS. In the definition of a relation or a property, we indicate the type of language used, the name of the predicate and the variables of the predicate.

This feature needs to be further analyzed.

3.4 AVIR descriptions specification

3.4.1 Descriptions consideration

MPEG7 description are also based on XML. The description DTD is directly generated from the "Schema" of the document. Basically, a description is composed of a set of tags included within the main "Description" tag. The possible elements are the DS and D names. The description can be seen has a tree of nested description schemes instances where the leaves are instances of descriptors. Some named relations relates parts of the descriptions. Some other links refers to external entities like raw data (image, sound, objects in videos, etc..), ontologies or procedural code.

The syntactic validation of description (the right nesting of elements) is handled by an XML parser. Constraints, links, datatypes and inheritance mechanisms are handled by a semantic parser based on the syntactic parser.

3.4.2 Coding descriptions to preserve semantic

In this chapter we propose to focus on the structure of the MPEG-7 descriptions and particularly on the way to express relations between description objects. We aim at disambiguating the "child/parent" relation between nested XML elements. Indeed, this implicit relation (the "nesting") can express many different kind of relation like "contains", "is-characterized-by", etc. For that purpose we propose a description structure which explicitly shows the relation between nested elements within descriptions. It is very similar to what Resource Description Framework (RDF) [11] and DSTC proposal does though our current proposal is not based on RDF. The description legibility and interpretation will be eased by such description structure.

It should be noticed that these considerations will impact the Description schemes Definition Language. Our contribution can be implemented as a specialization of XML schemas, among several other solutions, such as extending XML schemas.

Main issue addressed

As stated in the MPEG-7 DDL requirements documents [1] [2], the DDL should be able to express relations between description schemes. The first draft of the Generic Audio Visual Description Scheme demonstrates this need with, for instance, the event/object relation graph [12]. We propose to integrate the relations directly **in** the language instead of describing it **with** the language as proposed by the DS AHG group.

Indeed, XML schemas or XML DTD are too generic. It is not necessary to express relations between related objects (nested elements). This can lead to semantic ambiguity like in the following example:

Example :

```
<book>
  <title>...</title>
  <author>...</author>
  <chapter>...</chapter>
</book>
```

This description structure is allowed but it doesn't have a clear semantic since chapters are parts of the book whereas author is an attribute of the book. We have two nested elements (chapter and author) but the relationship they have with the book are different. It is not clearly defined in the syntax. We should have written a description where the relation 'contains' is explicitly written between the 'book' and the 'chapter' elements.

Example :

```
<book>
  <title>...</title>
  <author>...</author>
  <contains>
    <chapter>...</chapter>
  </contains>
</book>
```

Basic rules

We propose to define an XML Schemas application that permits such disambiguation by giving a unique way of expressing relation between objects. We define the nesting rules of the basic elements appearing in descriptions : Objects (i.e. Description Scheme instances), Descriptors and Relations. The description language should respect the following rules :

- 1- An object is composed of descriptors
- 2- An object can be in relation with one or many other objects

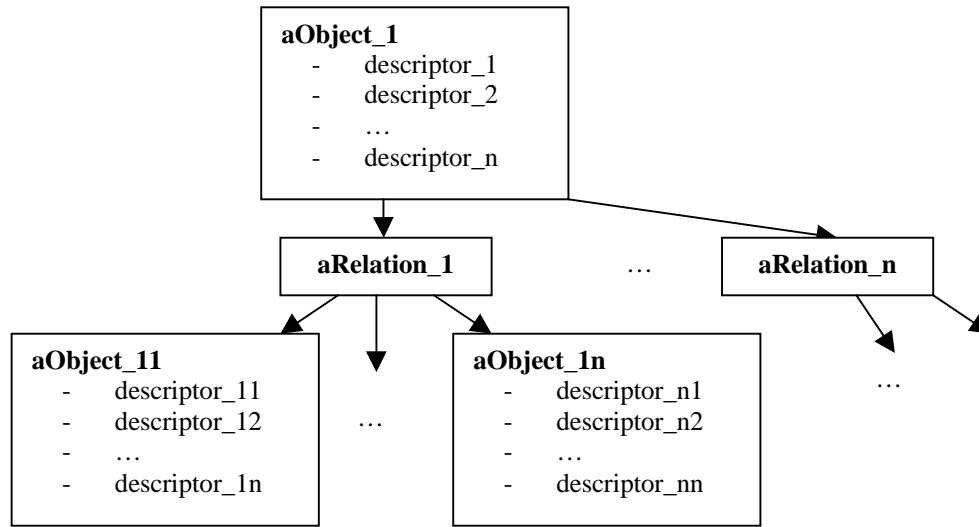
We focus on the way to represent relation between description objects within the description. Either the relation and the object definitions are defined in a description scheme which is expressed with the DDL. The following example shows formally the structure of relation between description objects :

Example :

```
<aObject1>
  <aDescriptor_1>...</aDescriptor_1>
  ...
  <aDescriptor_n>...</aDescriptor_n>
  <aRelation_1>
    <aObject_11>...</aObject_11>
    ...
    <aObject_1n>...<aObjectc_1n>
  </aRelation_1>
  ...
  <aRelation_n>
    ...
  </aRelation_n>
```

```
</aObject>
```

This architecture can be understood with the following tree :



It can be translated in natural language :

anObject_1 has a characteristic *descriptor_1*.
(instead of *descriptor_1* is a son of *anObject_1*).

anObject_1 is in relation *aRelation_1* with *anObject_11*.
(instead of *anObject_11* is a son of *Relation_1* which is a son of *anObject_2*).

Let's focus on a real life example :

```
Example:  
  
<movie id="m1">  
  <author>Mr Smith</author>  
  <date>12:2:1998</date>  
  <contains> <!-- m1 and sequences are in relation : the "contains" relation -->  
    <sequence id="s1">  
      <startTimeCode>12:3:4:5</startTimeCode>  
    </sequence>  
    <sequence id="s2">  
      <startTimeCode>12:6:9:5</startTimeCode>  
    </sequence>  
  </contains>  
</movie>
```

This example means :

“The movie ‘m1’ ‘contains’ two sequences ‘s1’ and ‘s2’. Each sequence is characterized by its ‘startTimeCode’. The movie is characterized by its ‘author’ and its ‘date’.”

Advanced rules

Obviously, that very simple structure could be extended. First the possibility of referencing a description object, instead of directly embedding the object, is considered. This feature allow relations to form a graph. On an other hand we can parameterized relations with global or individual descriptors.

The possibility of referencing a description object

When the relation graph is more complex than a tree (when it has loops for instance) nesting is no more sufficient, since we go beyond the natural tree structure of XML document. We need to be able to reference a description object. This drawback is directly the result of the XML syntax. For that purpose we use the XML attributes ID and IDREF.

Example of a similar link within a description :

```
<Description>
  <movie>
    <dc:title>The Rope</dc:title>
    <dc:author>Hitchcock</dc:author>
    <contains>
      .....
      <sequence id="seq_10">
        <type>dialog</type>
        <contains>
          .....
          <shot id="seq_10_shot_4">
            <starttimecode>0</starttimecode>
            <endtimecode>200</starttimecode>
            <similar>
              <Locator>
                <DSID name="shot" id="seq_10_shot_6"/> <!-- the link -->
              </Locator>
            </similar>
          </shot>
          <shot id="seq_10_shot_5">
            <starttimecode>200</starttimecode>
            <endtimecode>300</starttimecode>
          </shot>
          <shot id="seq_10_shot_6">
            <starttimecode>300</starttimecode>
            <endtimecode>400</starttimecode>
          </shot>
          .....
        </contains>
      </sequence>
      .....
    </contains>
  </movie>
</Description >
```

The possibility to express individual relation attributes

When the relation involves many description objects we might want to express attributes of individual relations. For that purpose we propose to use the "LI" html tag to characterize individual relations.

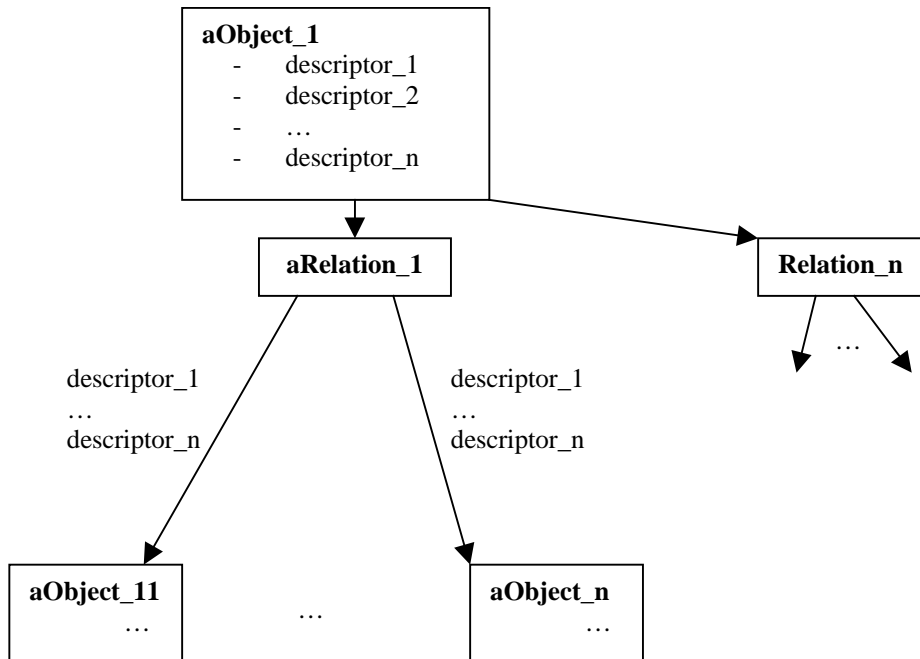
Example :

```
<Description>
  <movie>
    <dc:title>The Rope</dc:title>
    <dc:author>Hitchcock</dc:author>
    <contains>
      .....
      <sequence id="seq_10">
        <type>dialog</type>
        <contains>
          .....
          <shot id="seq_10_shot_4">
            <starttimecode>0</starttimecode>
            <endtimecode>200</endtimecode>
            <similar>
              <li>
                <Locator>
                  <DSID name="shot" id="seq_10_shot_6"/>
                </Locator>
                <likelihood>0.98</likelihood>
              </li>
            </similar>
          </shot>
          .....
        </contains>
      </sequence>
      .....
    </contains>
  </movie>
</Description >
```

```

        </similar>
    </shot>
    <shot id="seq_10_shot_5">
        <starttimecode>200</starttimecode>
        <endtimecode>300</endtimecode>
    </shot>
    <shot id="seq_10_shot_6">
        <starttimecode>300</starttimecode>
        <endtimecode>400</endtimecode>
    </shot>
    .....
</contains>
</sequence>
.....
</contains>
</movie>
</Description>
    
```

The following figure shows a graph representation of a general description :



Here is the corresponding syntax :

```

Example :
<Description>
  <aObject1>
    <aDescriptor_1>...</aDescriptor_1>    <!-- Object descriptors -->
    ...
    <aDescriptor_n>...</aDescriptor_n>
    <aRelation_1>
      <li>
        <aObject_11>...</aObject_11>
        <aDescriptor>...</aDescriptor>    <!-- Relation descriptors -->
        ...
        <aDescriptor>...</aDescriptor>
      </li>
      ...
      <li>
        ...
      </li>
    </aRelation_1>
    ...
  <aRelation_n>
    
```

```
...
  </aRelation_n>
</aObject>
...
<aObject>
...
  </aObject>
</Description>
```

We suggest to use an MPEG-7 description model with a higher semantic based on description schemes and relations between description schemes. This model is not new and has been the subject of many proposals [3] [4] [5] [6] and it is one of the RDF basic ideas. We think that respecting this model is a major requirement for the description and that it has some implication on the DDL. We propose here a way to express it within the description. It produces readable descriptions and ease their automatic parsing.

3.4.3 Locators in descriptions

Many description elements need to reference other description elements. The referenced resources can be description elements within the description (internal relation), description elements in an other description (external relation) or some external data such as image, text, ontologies or procedural code. In this chapter we propose a general method to include locators within description.

AVIR DDL Links are based on W3C-Xlink and Xpointer. W3C-Xlink can express links with multiple targets (extended links). As we aim at keeping description structures within description schemes, we decided to authorize only simple links (only one source and only one target). Indeed, links with multiple targets can express structural patterns. We suggest to define structures within description schemes only. This choice seems to be reasonable since it doesn't reduce expressiveness. It is still an open debate.

Locators are made by aggregating an URI and a pointer. The URI defines the target document while the pointer defines a precise access path to the target element within this document. The pointer is a string with a particular syntax. **We propose to keep this syntax for the legibility purpose, knowing that we swap at the end of the chapter to an equivalent XML syntax which has already been seen in preceding examples.**

Referencing a description element using pointers

The way of referencing part of a description is based on W3C XPointer working draft [9]. This chapter specifies a language for addressing internal parts of an MPEG7 description document. It provides specific reference to description elements, strings and other parts of MPEG7 description documents. Relations and constraints specification use this language.

An MPEG7Pointer locates part of a description. It is composed of an ordered set of location terms. The first location term is an absolute location term, it defines the starting point of a path which leads to the target. The following location terms are relative location terms which define steps to reach the target element. Each location terms is composed of a keyword and a set of parameters. This "reaching path" can be completed with a data specific location term, which defines a subpart of an element. This last location term allows to access regions in images or frames in movies or even words in sentences. It is not yet developed in this version of the DDL since we will rely on a coherent set of spatial datatypes.

The following example illustrate the use of links. A relation "verysimilar" is set to points at similar shots. This relation is useful to describe "shots / countershots" patterns, often used in dialogues.

Example of a verysimilar link within a description :

```
<Description>
  <movie>
    <title>The Rope</title>
    <director>Hitchcock</director>
    <Sequences>
      .....
      <sequence id="seq_10">
```

```

<type>dialog</type>
<Shots>
  .....
  <shot id="seq_10_shot_4">
    <starttimecode>0</starttimecode>
    <endtimecode>200</starttimecode>
    <verysimilar>Id(seq_10_shot_6)</verysimilar> <!-- the link -->
  </shot>
  <shot id="seq_10_shot_5">
    <starttimecode>200</starttimecode>
    <endtimecode>300</starttimecode>
  </shot>
  <shot id="seq_10_shot_6">
    <starttimecode>300</starttimecode>
    <endtimecode>400</starttimecode>
  </shot>
  .....
</Shots>
</sequence>
.....
</Sequences>
</movie>
</Description>
    
```

Lets precise the vocabulary which can be used for defining a pointer (i.e. a "reaching path").

The absolute location terms

- *Id()* : select the element with the given unique id
- *Root()* : the root element (the "mpeg7-desc" element)
- *This()* : the description scheme instance which contains the link

The relative location terms

Relative location terms allows to navigate step by step within description. They work on the tree structure, directly on the MPEG-7 stream:

- *child()* : the direct children of the source location.
- *ancestor()* : the ancestors (parent, grand parent, etc..) of the source location.
- *descendant()* : the descendants of the source location. They are selected in the order they are written in the MPEG7 stream.
- *fsibling()* : the younger brothers of the source location (defined after the source location in the mpeg7 stream).
- *psibling()* : the older brothers of the source location.

The following figure illustrates the location terms relative to the S node.

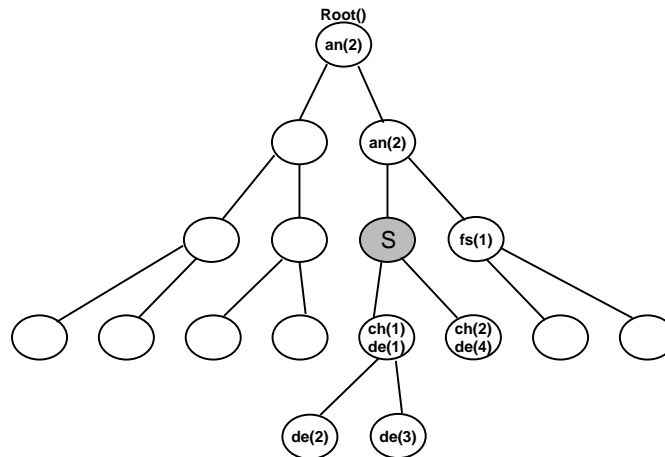


Figure 2 – Relative location terms working on tree structure
 an : ancestor, fs : fsibling, ch : child, de : descendant

Relative location terms working on the MPEG7 stream are the following :

- *following()* : the following elements taken in the MPEG-7 stream.
- *preceding()* : the preceding elements taken in the MPEG-7 stream.

The set of preceding elements is the set of all the elements that start or end before the location source. The set of following elements is the set of all the elements that start or end after the location source. It should be noted that a parent "precedes" and "follows" its children. The following figure illustrates the use of preceding and following location terms, supposing the tree node are written from left to right in the stream :

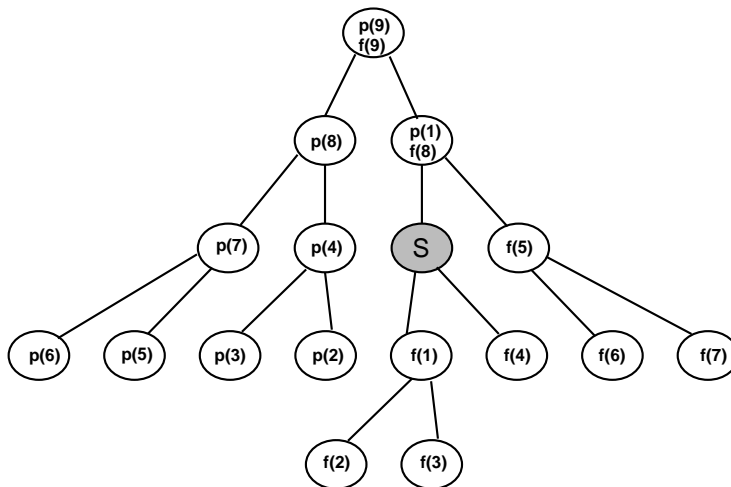


Figure 3 – Relative location terms working on mpeg-7 stream
f : following, p : preceding

The relative location terms can be parameterized with two parameters :

- the *first parameter* is an integer. It selects the element based on its instance number .
- the *second parameter* is an optional tag name. It selects the element based on its nature.

In the "verysimilar shots" example we had many possibilities to define the verysimilar link :

- *Id(seq_10_shot_6)* : absolute reference (element id is needed)
- *Root().child(1,movie).child(1,Sequences).child(10,sequence).child(1,Shots).child(6,shot)* : relative from the root element
- *This().fsibling(2)* : the second next brother of the description scheme instance containing the link
- *This().ancestor(1).child(6,shot)* : the 6th shot of the father sequence of the description scheme instance containing the link

The relation-traversal location term

The relation-traversal allows the path to jump to the target element of a relation. It is only applicable if the location source is a relation :

- *traverse()* : use to traverse a relation

The data accessing terms

The data accessing terms allow to locate parts of description between tags or parts of multimedia objects. These location terms will rely on spatial and temporal MPEG-7 datatypes. They are not yet precisely defined.

Referencing a description element in description scheme

Sometimes we need to express references on description elements within description schemes. For that purpose we use the pointer syntax with some new location terms. We add the following location terms :

- *source()* : jump to the source element of a relation. It is a relative location term.

- *target()* : jump to the target of a relation. It is a relative location term.
- *dsid()* : refers to the description scheme instance with a certain "id" in description scheme. It is a relative location term.

an XML syntax for links

It is possible to adopt an XML syntax for pointers. It allows to check syntax with a basic validating XML parser, but it reduces the description scheme legibility.

Informal examples of an pointer with the XML syntax :

```
The verysimilar example :

<versimilar>
  <Locator>
    <Id idref="seq_10_shot_6"/>
  </Locator>
</versimilar> the link -->

other :

<versimilar>
  <Locator>
    <Root/>
    <child num="1" tag="Sequences"/>
    <child num="10" tag="sequence"/>
    <child num="1" tag="Shots"/>
    <child num="6" tag="shot"/>
  </Locator>
</versimilar> the link -->

other :

<versimilar>
  <Locator>
    <This/>
    <ancestor num="1"/>
    <child num="6" tag="shot"/>
  </Locator>
</versimilar> the link -->
```

4 MPEG-7 and AVIR Description Schemes

4.1 AVIR MPEG-7 proposals analysis

In order to give a better overview of the AVIR member needs in terms of DDL we analyzed every AVIR partners description scheme proposals.

A DDL framework based on RDF and Ontologies – UPMC (p625 b)

- DS are class like / Description are objects
- Objects (Ds) are linked by properties
- Properties can have properties (Ex: likelihood)
- Ability to express properties about properties
- Specification of pseudo-inheritance rules
- Ontology, procedural code and external collection of document linking mechanism
- Ability to describe user's information

Constraints in DDL – UPMC (m4591)

- Constraints on attribute values
- Coordinate space of attribute values and transforms

Basic Semantic for Descriptions – UPMC (m4780)

- Relations between description schemes are part of the DDL

News Item Description Scheme – Tecmath (p66)

- Ability to reference audio and video segment of raw data
- Ability to reference external document
- Ability to reference particular descriptor in an external description
- Temporal datatypes
- Enumerative datatypes
- Ability to express spatial and temporal relation

A Description Scheme for TV program content – Philips NL(p655)

- Many different datatypes
- Nesting description schemes

The TOCAI DS for audio-visual documents - UNIBS (M4586)

- Hierarchical structure of description schemes
- Ordered set of description schemes
- References within description

Video Editing Work Description Scheme – UPMC (P624)

- Attributes likelihood and priority level
- Typed relation between description scheme instances

Descriptor for Camera Motion – LEP (p635)

- Compound datatypes
- Temporal datatypes

Mosaic Representation for Video Shot Description – LEP (p636)

- Temporal datatypes
- Linking to some external document (image) or ability to encode it within description

4.2 Formalization of AVIR DS

4.2.1 The TOCAI DS

```
<?xml version='1.0'?>

<schema name="http://www.mpeg7.org/tocai.xsd"
version="1.0"
xmlns="http://www.w3.org/mpeg7-ddl.xsd">

<include schemaName="http://www.w3.org/TR/xmlschema-2/datatypes.xsd"/>

<!-- ##### -->
<!-- Definition of 'AudioVisualDS' -->
<!-- ##### -->

<!-- Structure of the ToCAI DS -->

<DSType name='TOCAIDS'>
  <DSTypeRef name='TOCDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='MetaDescriptorsDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='AnalyticalIndexDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='ContextDS' minOccur='0' maxOccur='1' />
</DSType>
```

```
<DSType name='TOCDS'>
  <DSTypeRef name='AudioVisualStructureDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='AudioStructureDS' minOccur='0' maxOccur='1' />
</DSType>

<DSType name='AnalyticalDS'>
  <DSTypeRef name='AudioVisualObjectDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='AudioObjectsDS' minOccur='0' maxOccur='1' />
</DSType>

<!-- ##### -->
<!-- Audio-visual structure DS -->
<!-- ##### -->

<DSType name='AudioStructureDS'>
  <DSTypeRef name='SceneDS' minOccur='0' maxOccur='*' />
  <DTypeRef name='TimeCodeD' minOccur='2' maxOccur='2' />
</DSType>

<DSType name='SceneDS'>
  <DSTypeRef name='SceneDS' minOccur='0' maxOccur='*' />
  <DSTypeRef name='ShotDS' minOccur='0' maxOccur='*' />
  <DTypeRef name='TypeOfSceneD' minOccur='0' maxOccur='1' />
  <DTypeRef name='TimeCodeD' minOccur='2' maxOccur='2' />
</DSType>

<DSType name='ShotDS'>
  <DSTypeRef name='MosaicDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='OutlierDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='KFrameDS' minOccur='0' maxOccur='*' />
  <DTypeRef name='EditingEffectDS' minOccur='1' maxOccur='1' />
</DSType>

<!-- ##### -->
<!-- Analytical Index DS -->
<!-- ##### -->

<DSType name='AudioVisualObjectDS'>
  <DSTypeRef name='KeySceneDS' minOccur='0' maxOccur='1' />
  <DSTypeRef name='KeyImageClassDS' minOccur='0' maxOccur='1' />
  <DTypeRef name='OrderingKeysD' minOccur='1' maxOccur='1' />
</DSType>

<DSType name='KeySceneDS'>
  <DSTypeRef name='SceneTypeDS' minOccur='1' maxOccur='*' />
</DSType>

<DSType name='SceneTypeDS'>
  <DSTypeRef name='SceneGraphicRepresentationDS' minOccur='1' maxOccur='*' />
</DSType>

<DSType name='SceneGraphicRepresentationDS'>
  <DSTypeRef name='SceneContext' minOccur='0' maxOccur='1' />
  <DTypeRef name='PointerType' minOccur='1' maxOccur='*' />
</DSType>

<DescType name='PointerType'>
  <DTypeRef name='ReferencePointerDS' minOccur='1' maxOccur='*' />
</DescType>

<DSType name='KeyImageClassDS'>
  <DSTypeRef name='MPEG4VisualObjectDS' minOccur='0' maxOccur='1' />
</DSType>

<DSType name='MPEG4VisualObjectDS'>
  <DTypeRef name='MPEG4VisualObjectD' minOccur='0' maxOccur='1' />
  <DTypeRef name='PointerType' minOccur='1' maxOccur='*' />
</DSType>
```

4.2.2 Definition of 'AudioDS'

```
<DataType name='noise'>
  <BaseType name='string' />
  <Enumeration>
    <Literal>rumbling</Literal>
    <Literal>explosion</Literal>
    <Literal>jangling</Literal>
    <Literal>creaking</Literal>
    <Literal>friction</Literal>
    <Literal>purring</Literal>
    <Literal>humming</Literal>
    <Literal>backgroung</Literal>
    <Literal>scraping</Literal>
    <Literal>sputtrering</Literal>
    <Literal>unknown</Literal>
  </Enumeration>
</DataType>

<DataType name='simusic'>
  <BaseType name='string' />
  <Enumeration>
    <Literal>classical</Literal>
    <Literal>jazz</Literal>
    ...
    <Literal>rock</Literal>
  </Enumeration>
</DataType>

<DataType name='pho'>
  <BaseType name='string' />
  <Enumeration>
    <Literal>iy</Literal>
    <Literal>ih</Literal>
    ...
    <Literal>epi</Literal>
  </Enumeration>
</DataType>

<!-- ##### -->
<!-- seg DS -->
<!-- ##### -->

<DSType name='seg'>
  <DTypeRef name='startTimeCode' minOccur='1' maxOccur='1' />
  <DTypeRef name='endTimeCode' minOccur='1' maxOccur='1' />
</DSType>

<DType name='startTimeCode'>
  <subDOF name='timeCode' />
</DType>

<DType name='endTimeCode'>
  <subDOF name='timeCode' />
</DType>

<DType name='timeCode'>
  <DataTypeRef name='timeCode' />
</DType>

<!-- ##### -->
<!-- segpho DS -->
<!-- ##### -->

<DSType name='segpho'>
  <subDSOf name='seg' />
  <DTypeRef name='trpho' minOccur='1' maxOccur='1' />
</DSType>

<DType name='trpho'>
  <DataTypeRef name='pho' />
</DType>

<!-- ##### -->
<!-- Domains and set of words datatype -->
```

```
<!-- ##### -->
<DataType name='domainNews'>
  <BaseType name='string' />
  <Enumeration>
    <Literal>the</Literal>
    <Literal>a</Literal>
    ...
    <Literal>is</Literal>
  </Enumeration>
</DataType>

<!-- ##### -->
<!-- Topics and set of Keyword datatype -->
<!-- ##### -->

<DataType name='topicAngola'>
  <BaseType name='string' />
  <Enumeration>
    <Literal>UNITA</Literal>
    <Literal>Dos Santos</Literal>
    ...
    <Literal>Muhango</Literal>
  </Enumeration>
</DataType>

<!-- ##### -->
<!-- vocal dictation DS -->
<!-- ##### -->

<DSType name='segWord' />
  <subDSOf name='seg' />
  <DTypeRef name='word' minOccurs='1' maxOccurs='1' />
</DSType>

</schema>
```

4.2.3 The VDX/ADX format

```
<schema name="http://www.mpeg7.org/vdx.xsd"
  version="1.0"
  xmlns="http://www.w3.org/mpeg7-ddl.xsd">

  <!-- ##### -->
  <!-- DataType definition -->
  <!-- ##### -->

  <DataType name='data'>
    <BaseType name='String' />
  </DataType>

  <DataType name='Longdata'>
    <BaseType name='String' />
    <MaxLength>2047</MaxLength>
  </DataType>

  <!-- 32-bit integer -->
  <DataType name='int'>
    <BaseType name='Real' />
    <MinInclusive>-2147483647</MinInclusive> <!-- -((2^32)/2-1) -->
    <MaxInclusive>2147483648</MaxInclusive> <!-- (2^32)/2 -->
    <Precision>0</Precision>
  </DataType>

  <!-- 64-bit integer -->
  <DataType name='long'>
    <BaseType name='Real' />
    <MinInclusive>-9223372036854775807</MinInclusive> <!-- -((2^64)/2-1) -->
    <MaxInclusive>9223372036854775808</MaxInclusive> <!-- (2^64)/2 -->
    <Precision>0</Precision>
  </DataType>
```

```
<!-- 64-bit integer -->
<DataType name='longPositive'>
  <BaseType name='Real' />
  <MinInclusive>0</MinInclusive>
  <MaxInclusive>18446744073709551616</MaxInclusive> <!-- (2^64)-1 -->
  <Precision>0</Precision>
</DataType>

<!-- floating-point number -->
<DataType name='float'>
  <BaseType name='Real' />
</DataType>

<DataType name='bits'>
  <BaseType name='Real' />
  <MinInclusive>-2147483647</MinInclusive> <!-- -((2^32)/2-1) -->
  <MaxInclusive>2147483648</MaxInclusive> <!-- (2^32)/2 -->
  <Precision>0</Precision>
</DataType>

<DataType name='metaname'>
  <BaseType='String' />
  <MaxLength>255</MaxLength>
</DataType>

<DataType name='stratumname' />
  <BaseType='String' />
</DataType>

<DataType name='doccontent' />
  <BaseType='String' />
</DataType>

<!-- ##### -->
<!-- general DS -->
<!-- ##### -->

<DSType name='video'>
  <DSTypeRef name='meta' minOccur='0' maxOccur='*' />
  <DSTypeRef name='stratum' minOccur='0' maxOccur='*' />
  <AttrDecl name='version' required='yes'>
    <DataTypeRef name='data' />
    <default>3.1</default>
  </AttrDecl>
  <AttrDecl name='videoname' required='yes'>
    <DataTypeRef name='data' />
  </AttrDecl>
  <AttrDecl name='videoformat' required='yes'>
    <DataTypeRef name='int' />
    <default>1</default>
  </AttrDecl>
  <AttrDecl name='videostart' required='yes'>
    <DataTypeRef name='long' />
    <default>0</default>
  </AttrDecl>
  <AttrDecl name='videoduration' required='yes'>
    <DataTypeRef name='longPositive' />
    <default>0</default>
  </AttrDecl>
  <AttrDecl name='videoheight' required='yes'>
    <DataTypeRef name='int' />
    <default>1</default>
  </AttrDecl>
  <AttrDecl name='videowidth' required='yes'>
    <DataTypeRef name='int' />
    <default>-1</default>
  </AttrDecl>
  <AttrDecl name='framenameformat' required='yes'>
    <DataTypeRef name='data' />
    <default>f%tc%.jpg</default>
  </AttrDecl>
  <AttrDecl name='framedir' required='yes'>
    <DataTypeRef name='data' />
    <default></default>
  </AttrDecl>
</DSType>
```

```
<AttrDecl name='framerate' required='yes'>
  <DataTypeRef name='float' />
  <default>25.0</default>
</AttrDecl>
<AttrDecl name='frameheight' required='yes'>
  <DataTypeRef name='int' />
  <default>-1</default>
</AttrDecl>
<AttrDecl name='framewidth' required='yes'>
  <DataTypeRef name='int' />
  <default>-1</default>
</AttrDecl>
</DSType>

<DSType name='meta'>
  <AttrDecl name='name' required='yes'>
    <!-- name must be unique across all the meta elements -->
    <DataTypeRef name='metaname' />
  </AttrDecl>
  <AttrDecl name='value' required='yes'>
    <DataTypeRef name='Longdata' />
  </AttrDecl>
</DSType>

<DSType name='stratum'>
  <DTypeRef name='doc' minOccur='0' maxOccur='*' />
  <DSTypeRef name='frame' minOccur='0' maxOccur='*' />
  <DSTypeRef name='chunk' minOccur='0' maxOccur='*' />
  <AttrDecl name='name' required='yes'>
    <!-- name must be unique across all the stratum elements -->
    <DataTypeRef name='stratumname' />
  </AttrDecl>
</DSType>

<DSType name='chunk'>
  <DTypeRef name='doc' minOccur='0' maxOccur='*' />
  <DSTypeRef name='frame' minOccur='0' maxOccur='*' />
  <AttrDecl name='start' required='yes'>
    <!-- start must not be greater than end -->
    <DataTypeRef name='long' />
  </AttrDecl>
  <AttrDecl name='end' required='yes'>
    <DataTypeRef name='long' />
  </AttrDecl>
  <AttrDecl name='reasons' required='yes'>
    <DataTypeRef name='bits' />
  </AttrDecl>
</DSType>

<DSType name='frame'>
  <DTypeRef name='doc' minOccur='0' maxOccur='*' />
  <AttrDecl name='tc' required='yes'>
    <DataTypeRef name='long' />
  </AttrDecl>
  <AttrDecl name='reasons' required='yes'>
    <DataTypeRef name='bits' />
  </AttrDecl>
</DSType>

<DType name='doc'>
  <DataTypeRef name='doccontent' />
  <AttrDecl name='reasons' required='yes'>
    <DataTypeRef name='bits' />
  </AttrDecl>
</DSType>

</schema>
```

5 The use of DS in AVIR architecture

The meta-data should be described using MPEG-7 rules and each involved system has to be aware of DDL / DSs and Ds. The following figure shows a possible version for the AVIR architecture.

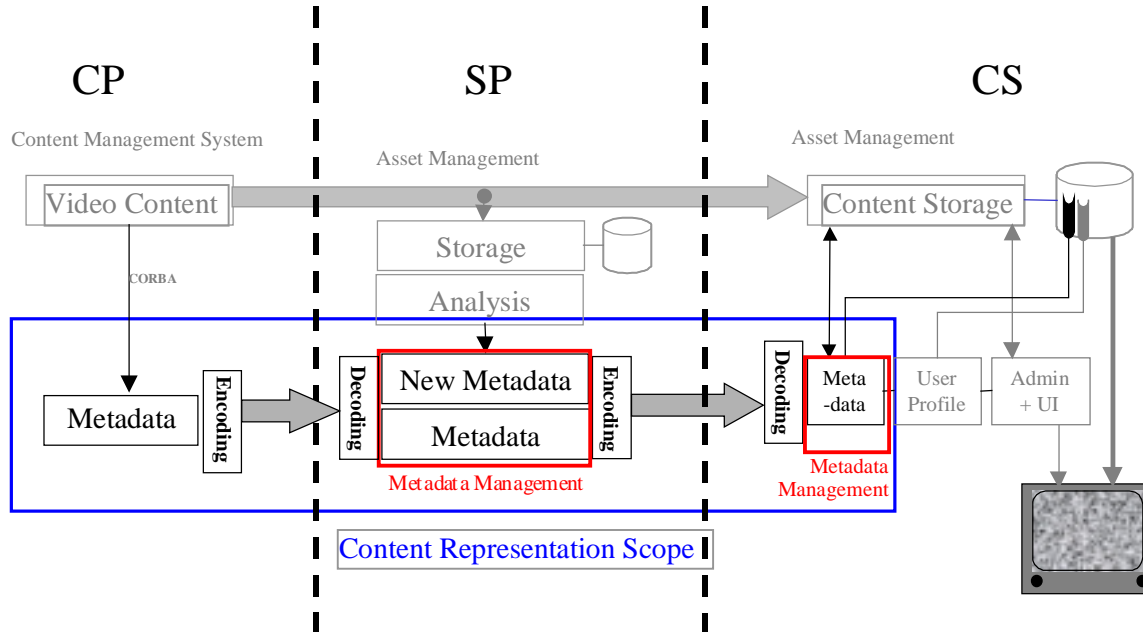


Figure 4 – Content Representation Scope

In this architecture, the Service Provider, the Content Provider and the Consumer System should manage and exchange MPEG-7 descriptions. Here is summarized the possible usage of description schemes by the different actors:

- **Content provider:** The content provider extracts meta-data and sends MPEG-7 descriptions to the Service provider. The meta-data can be automatically extracted from the content MPEG-2 stream or can be generated through manual annotations. In both cases, using predefined description schemes aware tools has many advantages. Annotation tools can generate graphical user interfaces by parsing description schemes. It allows an early detection of syntactic and semantic errors of descriptions. The creation and encoding phases are of course dependent of the DSs definition expressed with the DDL.
- **Service provider side :** The service provider receives an encoded MPEG-7 stream, which should be decoded, parsed and stored. The storage systems will allow to collect extracted information. The service provider will periodically generate descriptions by assembling meta-data stored in the metadata database. These descriptions will be multiplexed with the MPEG-2 stream. The generated descriptions are based on the same meta-data but differ in structure or relevance. For instance, the service provider can periodically send a general description of the current TV program (every 30 sec.) and a more detailed before the program starts (5 minutes). These descriptions are different but overlap for some meta-data (the name of the program, the genre, etc.). Furthermore, the service provider could also generate on-demand descriptions for specialized or advanced users through internet access. The service provider should have tools capable of assembling descriptions using description schemes.
- **Consumer System side :** The consumer system receives MPEG-7 streamed descriptions. It should store them in a local database system. The model of the database and its behavior should be influenced by the DS definitions associated with each description. Some “basic” MPEG-7 description schemes are used for the main CS features. These description schemes will certainly look like the ones proposed at recent MPEG-7 meetings (cf. chapter 2). But the consumer systems should have some degree of adaptability. This means that it could receive other MPEG-7 descriptions than those defined in the AVIR DS core set.

These “new” descriptions schemes could import advanced features to the CS, like some smart similarity based retrieval (i.e. importing intelligent agents for smart retrieval) or new kind descriptors. If a back channel is available, the newly acquired description scheme may help to navigate through related sites or import some small domain ontologies. It should be noted that linking descriptions with external ontologies and procedural code is one of the MPEG-7 requirements.

In the following chapter we describe ways to use Description Schemes and associated mechanisms within the AVIR data flow. We show first some solutions for DDL parser and DS-aware description parsers as they are the core elements of MPEG-7 compliant tools. We then describe the relations with database architecture through the storing and querying process.

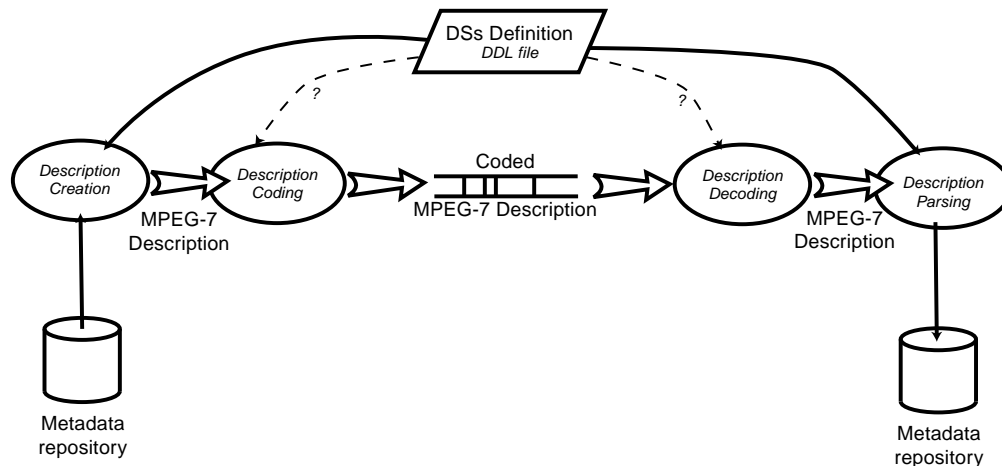


Figure 5 – The influence of Description Schemes in the overall process chain

5.1 The Parsing of Description Schemes and Descriptions

Every MPEG-7 compliant tool should incorporate some Description Scheme analysis tools and some adaptive Description parser. The main goal of the Description Scheme parser is to ensure some static checking of data model described by the DS. It should detect inheritance loops or impossible constraint specifications. This parser then instantiates a MPEG7 scheme element used by the description parser which one will verify the validity of the description structure (syntactic validation) and the correctness of its values (semantic validation).

The MPEG-7 parser should check the validity of an MPEG-7 description relative to its MPEG-7 schema. This includes both checking the syntax and grammar and the validation of constraints. As XML parser only do some syntactic checking we will have to adapt existing C, C++ or Java XML Parser. Ideally the MPEG7-style validation and processing to extract the key descriptors will be implemented via a standard API defined by MPEG7 consortium. Currently there are two approaches to standard APIs for XML parsers:

- **Simple Api for XML (SAX)**: as a basis for an event-based API parser [14]
- **Document Object Model API (DOM)** : as a basis for a tree-based XML parser [15]

The advantages of using SAX (an event-based parser) over DOM (a tree-based parser) include:

- Tree-based APIs put a greater strain on resources (such as system memory) especially if the document is large - which is likely to be the case for multimedia documents - since the entire document is cached in memory.
- Some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

- It can be very inefficient to construct and traverse an in-memory parse tree just to locate a single piece of contextual information when an event-based interface would allow you to find it in a single pass using very little memory.
- Since the DOM SDK supports SAX 1.0, it is still possible to construct and traverse a DOM in-memory parse tree using the SAX API.
- DOM is still at the proposal stage whilst SAX was released in May 1998. There are a large number of existing XML parsers and applications which support SAX.

The XML parser needs to be written in C++ for the Experimentation Model of MPEG-7, and this seems to better cope with AVIR architecture (specially for the CS). The choice of existing parsers is limited e.g. expat, James Clark's XML parser in C, the Language Technology Group's XML developers' toolkit LT XML or Microsoft's XML parser in C++ [25]. Unfortunately these are all non-validating parsers which means that they don't take DTD into account. Because SAX implementations are currently available only in Java and Python, it may also be necessary to write a C++ SAX driver to provide an events-based API. In the case of the use of BNF grammar some C or C++ parser should be available.

Consequently, the mock-up presented in the document is written in JAVA and developed over a SAX basis.

5.1.1 Parsing Description Schemes Definitions

Remark :

The DDL AHG is initiating the development of a BNF grammar with the recommendation to stay as close as possible to XML Schema. Due to the on-going development of the MPEG7 grammar, the following example may change (it corresponds with a July-99 version of the DDL grammar) The new version will be available after the Melbourne Meeting.

Apart from this implementation issue the DS parser offer mechanisms for DS static validity checking:

- *Grammatically correct definitions of DSs* : syntactic parsing of grammar and expression of constraints, datatypes format, etc.
- *Datatype consistence* : the constraints associated with datatypes are plausible (ex : min**Value** < Max**Value**).
- *Inheritance loops* : Loops checking in the inheritance graph.
- *Inheritance and relation Source and Target* : The subclassing of relations are correct if the subclassing of a relation source (which is a DS) is a "subDSOF" of the source of its super-Relation.

Moreover, The MPEG7 parser create an MPEG7 Structure, which does not exactly correspond to the DOM structure (oriented to XML). The API has to be defined in the MPEG7 DDL AD-HOC Group. Nevertheless, our current version generates the DTD corresponding with Descriptions Schemes Definitions.

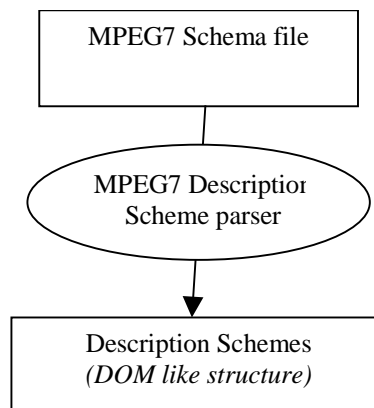


Figure 6 - Description Scheme Parsing

In the above example :

- The tree structure is generated using inheritance property of the DS. (for instance, VisualObject is a subDS of MovieGranule)
- It is possible to read a description of the object (DS, D or other) and, if available, the DTD generated by the description scheme definition.

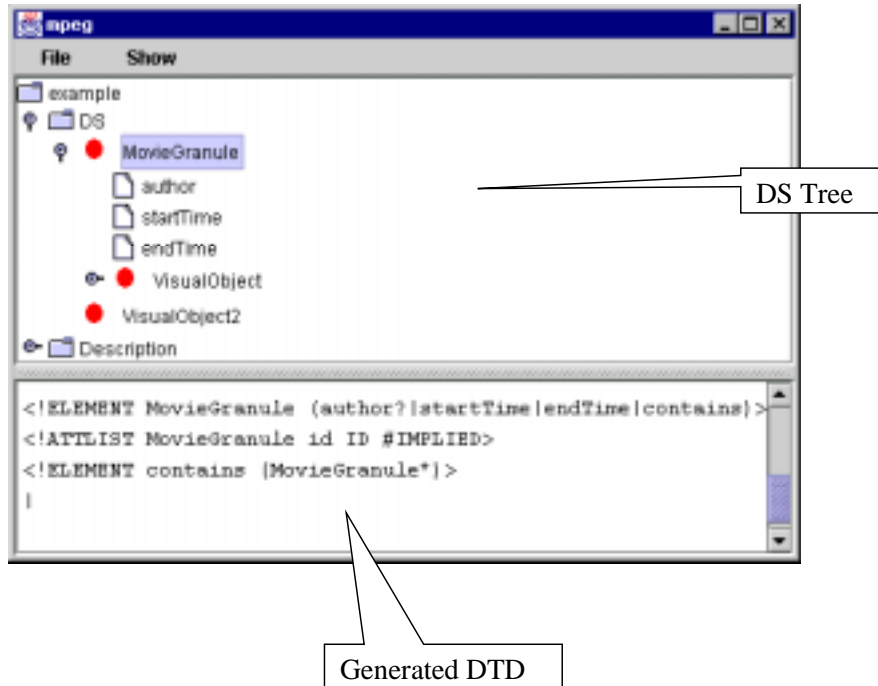


Figure 7 - example of description scheme parsing

5.1.2 Parsing description using description schemes

Using description schemes, the DS parser create some MPEG7 Scheme Elements. These elements are used by the MPEG7 parser during the description parsing. As the MPEG-7 Scheme Elements can generate their own DTD, it could be possible using a validating parser (i.e. verifying the xml syntax with the DTD) to validate a MPEG-7 Description relative to MPEG-7 Scheme Elements. But, we should add data type validation module. In the mock-up, we have created our own validating parser without the DTD (not expressive enough to handle every validity mechanism). Then, from a description file and MPEG-7 Scheme Elements, the MPEG7 parser creates some MPEG7 Elements holding the description.

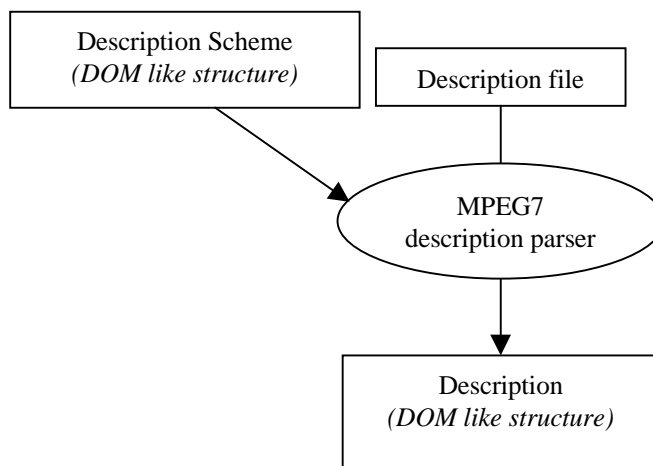


Figure 8 - Description Parsing

Semantic validation of description

In addition to basic syntax and grammar checking, the parser must perform the following types of validation and return appropriate error messages when the MPEG-7 description is non-conformant:

- **Validation of Property Values.** Through the API, the processor will have to provide procedures which can convert strings (PCDATA) to each of the supported data types. It will also need to check that the given value lies within the Min and Max or MinExclusive and MaxExclusive bounds or that the default value is substituted if no value is supplied. If a property is defined to be a particular class, then this must also be checked.
- **Validation of Cardinality.** The processor must check that all cardinality constraints are conformed to.
- **Validation of Inheritance.** Single inheritance only will be supported. Multiple inheritance can easily lead to complex inconsistencies. Inheritance will simply be provided by repetition of the properties and attributes of the superclass within the subclass and repetition of the property components in the sub-property.
- **Validation of Relations and Structures.** The processor will need to check that the properties and relations used in the MPEG-7 description all conform to the range and domain constraints specified in the MPEG-7 schema.
- **Validation of Property Constraints on Related Classes.** The processor will also need to be able to parse and evaluate the algebraic expressions used to define constraints on the property values of related classes.

Here, the example in case of a Description File :

- The tree is built using relations structure. It represents the triplet (object, relation, reference to an other object). For example, the instance of MovieGranule (mv41) is in relation named '*contains*' with the instance (mv423).
- In the other panel, a description of the object is shown.

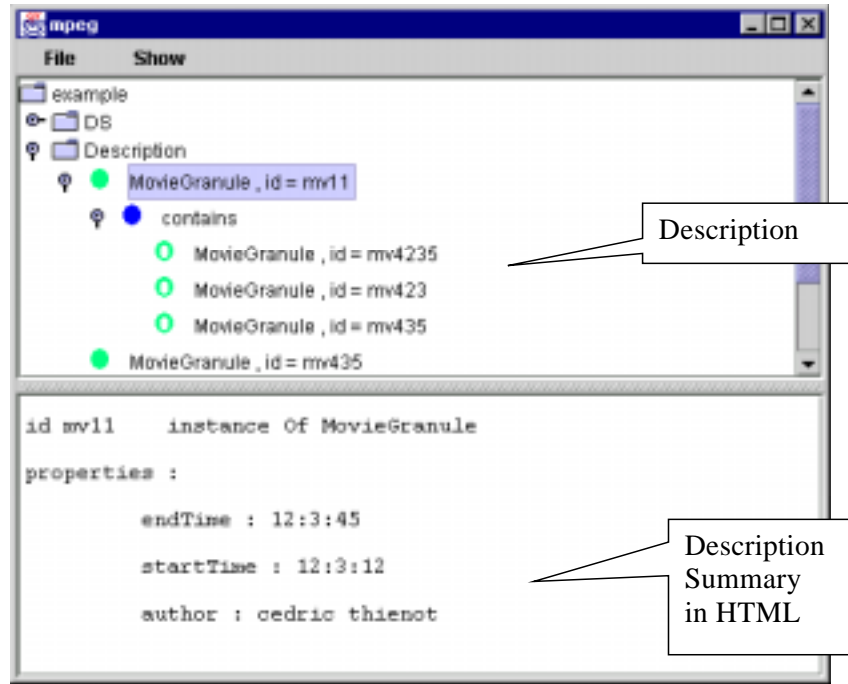


Figure 9 - Example of description parsing

5.2 AVIR databases, Descriptions and Description Schemes

In this section we concentrate on the relationship between descriptions and databases. Description Schemes are used for parsing descriptions, filling and querying database.

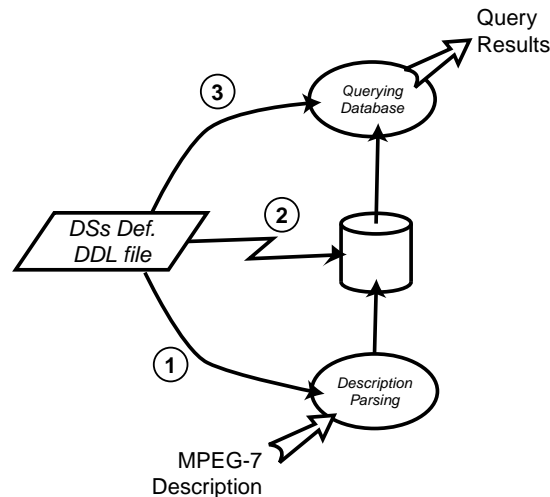


Figure 10 – Roles of Description Schemes in database management

MPEG-7 descriptions should be stored in a database manager system. In this context, the description schemes are used to create database tables in which description will be stored (arrow 2 in figure 10). The database manager data model has to be flexible enough to handle such requirements. The database interface specification proposed in deliverable 10 seems to correspond to those needs.

With this database interface DS instances are stored in separate tables. Descriptor datatypes should also be mapped to the database model with particular attention to compound datatypes. The description tree will be

stored in the database up to a defined level of granularity. Some parts of the description can be stored in an “unparsed” format and reparsed at query time. This “reparsing” effort can be done with a SAX-like API. The implementation of an abstract DOM-like API for the AVIR database would be a nice feature.

A multimedia description is not only a set of nested DS instances. It will contain many relations linking it to distant DS instances. Accessing DS instances through these links should also be considered.

Finally the initial description can be expanded with both ontology rules and coordinate rules defined in description schemes. The linking model between description and ontology is not yet defined but it is a major requirement for DDL.

6 Conclusions

The AVIR contribution to MPEG-7 include participation to the DDL development, and to parts of MPEG-7 DSs. Participation in the DDL development is mainly provided by UPMC/ LIP6 who is in charge in MPEG of the DDL parser. Some description schemes required by the project have been proposed by several AVIR members and are already enclosed by the MPEG documents (see AVIR Deliverable 3 and 4) and can all be written using the DDL grammar.

Whereas the MPEG7 group will still provide new features on the DDL in the next months, the present state of the grammar (presented in the previous pages) appears to be reliable enough to be used in the AVIR project. In the next months, the MPEG-DDL group will work on the development of mechanisms to map between the grammar and XML Schema. This let us assume that there will be a high level of interoperability between descriptions developed regarding to both MPEG and W3C specifications.

Nevertheless, AVIR will still contribute to the future evolution of MPEG-7, and will as much as possible, propose solutions to problems that could be raised up during the development step.

7 References

- [1] *MPEG Requirements Group*, "MPEG-7: Requirements Document", Doc. ISO/MPEG NXXXX, MPEG Seoul, March 1999
- [2] *MPEG Requirements Group*, "MPEG-7: Requirements Document", Doc. ISO/MPEG N2461, MPEG Atlantic City, October 1998
- [3] *Hunter J., DSTC*, "A Proposal for an MPEG-7 DDL", P547, MPEG-7 AHG Test and Evaluation Meeting, Lancaster, Feb 1999
- [4] *Hunter J., DSTC*, "A Revised Proposal for an MPEG-7 DDL", M4518, MPEG Seoul, 1999
- [5] *Faudemay P., Joly P., Thienot C., Seyrat C., University of Paris 6*, "Constraints expression in a DDL based on XML and classes", M4591, MPEG Seoul, 1999
- [6] *Faudemay P., Joly P., Thienot C., Seyrat C., University of Paris 6*, "An Extensible DDL Framework based on RDF and Ontologies", M4542, MPEG Seoul, 1999
- [7] *ACTS-DICEMAN*, "DICEMAN Description Definition Language", P184, MPEG-7 AHG Test and Evaluation Meeting, Lancaster, Feb 1999
- [8] *W3C*, "XML Namespaces", <http://www.w3.org/TR/REC-xml-names>, 14-January-1999.
- [9] *W3C*, "XML Pointer Language (Xpointer) working draft", <http://www.w3.org/TR/1998/WD-xptr-19980303>, 03-March-1998.
- [10] *W3C*, "Mathematical Markup Language (MathML) 1.0 specification", <http://www.w3.org/TR/REC-MathML>, 7-April-1998
- [11] *W3C*, "Resource Description Framework (RDF) Model and Syntax Specification", <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>, W3C Recommendation, 22 February 1999
- [12] *MPEG*, "Generic Audio Visual Description Scheme", *MPEG-7 mailing list*, 1-July-1999.
- [13] *MPEG Requirements Group*, "MPEG-7: Requirements Document", Doc. ISO/MPEG W2859, MPEG Vancouver, July 1999
- [14] *Meggison Technologies*, "SAX 1.0: The Simple API for XML", <http://www.meggison.com/SAX/index.html>
- [15] *W3C Recommendation*, "Document Object Model (DOM) Level 1 Specification", <http://www.w3.org/TR/REC-DOM-Level-1>, October, 1998.

8 Annex 1 – Document Type Definition of the MPEG-7 DDL

```
<?xml version='1.0'?>
<!DOCTYPE schema [
<!ELEMENT schema (DSType|DType|AttrGroup|DataType)*>
<!ATTLIST schema version CDATA #REQUIRED
                name CDATA #REQUIRED
                xmlns CDATA #IMPLIED>

<!ENTITY % idName "name ID #REQUIRED">

    <!-- ##### -->
    <!-- DS & D -->
    <!-- ##### -->

<!ENTITY % DSModel "DSTypeRef|DTypeRef|all|sequence|choice">

<!ENTITY % Attribute "minOccur CDATA #IMPLIED
                    maxOccur CDATA #IMPLIED
                    minOccurPar CDATA #IMPLIED
                    maxOccurPar CDATA #IMPLIED">

<!ELEMENT DSType ((AttrDecl|AttrGroupRef)*,subDSOf?,(%DSModel;)*)>
<!ATTLIST DSType %idName;
            abstract (true|false) "false">

<!ELEMENT DSTypeRef EMPTY>
<!ATTLIST DSTypeRef %Attribute;>

<!ELEMENT subDSOf EMPTY>
<!ATTLIST subDSOf %idName;>

<!ENTITY % DModel "DataTypeRef|DTypeRef|all|sequence|choice">

<!ELEMENT DType ((AttrDecl|AttrGroupRef)*,subDOF?,(%DModel;)*)>
<!ATTLIST DType %idName;>

<!ELEMENT DTypeRef EMPTY>
<!ATTLIST DTypeRef %Attribute;>

<!ELEMENT subDOF EMPTY>
<!ATTLIST subDOF %idName;>

<!-- ##### -->
<!-- DS & D -->
<!-- ##### -->

<!ELEMENT all ANY>
<!ATTLIST all %Attribute;>

<!-- ##### -->
<!-- DS & D -->
<!-- ##### -->

<!ELEMENT choice ANY>
<!ATTLIST choice %Attribute;>

<!-- ##### -->
<!-- DS & D -->
<!-- ##### -->

<!ELEMENT sequence ANY>
<!ATTLIST sequence %Attribute;>
```

```
<!-- ##### -->
<!-- Attribute -->
<!-- ##### -->

<!ELEMENT attrGroup (attrDecl)*>
<!ATTLIST attrGroup %idName;>

<!ELEMENT attrDecl (datatypeRef,default?)>
<!ATTLIST attrDecl %idName;
           required (true|false) #REQUIRED>

<!ELEMENT default (PCDATA)>

<!ELEMENT AttrGroupRef EMPTY>
<!ATTLIST AttrGroupRef %idName;>

<!-- ##### -->
<!-- DataType -->
<!-- ##### -->

<!ELEMENT DataTypeRef EMPTY>
<!ATTLIST DataTypeRef %idName;>

<!ELEMENT
           DataType
           (BaseType,
           (Length|MaxLength|MinLength|MinInclusive|MaxInclusive|MinInclusive|MaxInclu
           sive|Precision|Scale|Enumeration|LexicalRepresentation))>
<!ATTLIST DataType name ID #REQUIRED>

<!ELEMENT BaseType EMPTY>
<!ATTLIST BaseType name (NMTOKEN | boolean | real | dateTime | binary | uri
|language) #REQUIRED>

<!ELEMENT MaxInclusive (#PCDATA)>

<!ELEMENT Length (#PCDATA)>

<!ELEMENT MaxLength (#PCDATA)>

<!ELEMENT MinLength (#PCDATA)>

<!ELEMENT MinInclusive (#PCDATA)>
<!ELEMENT MaxInclusive (#PCDATA)>
<!ELEMENT MinExclusive (#PCDATA)>
<!ELEMENT MaxInclusive (#PCDATA)>
<!ELEMENT Precision (#PCDATA)>
<!ELEMENT Scale (#PCDATA)>

]>
```

9 Annex 2 - Datatype masks

A mask is a datatype format constraint. A mask consists of symbols, groups of symbols, and patterns, any of which may be modified by occurrence specifiers. Each symbol is a placeholder that stands for a character or a class of characters. Date and time masks tokens are taken from those defined in [ISO-8601]

A or a	A single alphabetic character
B or b	Any one of the boolean characters (0 or 1)
D	A single digit representing the day of the week, in the range 1-7 (Monday-Sunday) (ISO8601-5.1.2)
DD	Two digits representing a day in a month in the Gregorian calendar, in the range 01-31 (ISO8601-5.1.2)
DDD	Three digits representing a day in a year in the Gregorian calendar, in the range 001-366 (ISO8601-5.1.2)
E	The character "E", used to indicate floating point numbers
hh	Two digits representing hours in a day, in the range 00-24 (ISO8601-5.1.2)
MM	Two digits representing a month in the Gregorian calendar, in the range 01-12 (January-December) (ISO8601-5.1.2)
mm	Two digits representing minutes in an hour, in the range 00-59 (ISO8601-5.1.2)
N	Any valid XML name character
n	An integer number consisting of one or more digits.
P	The character "P", used as a "period designator" to indicate the duration of a period of time. (ISO8601-5.1.2)
p	Any one of the punctuation characters (. or : or ; or ,)
Q or q	Any one of the quote characters (" or ' or `)
S	Indicates a signed number. The characters "+" or "-" must appear in this position
ss	Two digits representing seconds in a minute, in the range 00-59 (ISO8601-5.1.2)
s	One or more digits representing a decimal fraction of a second (ISO8601-5.1.2)
T	The character "T", used as a "time designator" to indicate the start of date time of day field. (ISO8601-5.1.2)
U or u	Any character that is valid in a [URI], [URL], or [URN]
W	The character "W", used as a "week designator" to indicate the start of date week field. (ISO8601-5.1.2)
ww	Two digits representing the week number in a year, in the range 1-52 (ISO8601-5.1.2)
X or x	Any character
YYYY	Four digits of a year (ISO8601-5.1.2)
Z	The leftmost leading numeric character that can be replaced by a space character when the content of the Z position is the numeral 0
space	A single blank character
#	Any numeric character
\$	A currency symbol
0	The single numeric character "0"
1	The single numeric character "1"
2	The single numeric character "2"
3	The single numeric character "3"
4	The single numeric character "4"
5	The single numeric character "5"
6	The single numeric character "6"
7	The single numeric character "7"
8	The single numeric character "8"
9	The single numeric character "9"
(...)	Represents a grouping of the symbols found between the parentheses. Within parentheses the meaning of mask symbols apply.
[...]	Represents one of the characters found between the square brackets. Within square brackets any character, except for "-" represents itself. The character "-" indicates a range of characters beginning with the character to the left of the "-" and ending with the character to the right.

- * Indicates that the preceding character, or group, may occur zero or more times.
 - + Indicates that the preceding character, or group, may occur one or more times.
 - ? Indicates that the preceding character, or group, may occur zero or one time.
 - {n,m} Indicates that the preceding character, or group, must occur at least n times and no more than m times.
n must be a positive integer or zero
m may be an integer greater than n, or
m may be the character "*", indicating that the maximum is unbounded.
- !, @, %, _ , = , / , : , ; , - , . and , Represent themselves

10 Annex 3 - Regular Expressions

Note: The following description of regular expressions is copied from the documentation of the Perl programming language. It is expected to be used for XML schema datatyping.

[Definition:] Regular expressions, similar to those in [Perl], can be used to constrain the format of strings. A regular expression is an alphanumeric string consisting of character symbols. Each symbol, which is usually one character but may be two characters, is a placeholder that stands for a set of characters.

Any single character matches itself, unless it is a metacharacter with a special meaning described here or above. You can cause characters that normally function as metacharacters to be interpreted literally by prefixing them with a "\" (e.g., "\" matches a ".", not any character; "\" matches a "\").

A series of characters matches that series of characters in the target string, so the pattern `blurfl` would match "blurfl" in the target string.

You can specify a character class, by enclosing a list of characters in [], which will match any one character from the list. If the first character after the "[" is "^", the class matches any character not in the list. Within a list, the "-" character is used to specify a range, so that a-z represents all characters between "a" and "z", inclusive. If you want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash. (The following all specify the same class of three characters: [-az], [az-], and [a\ -z]. All are different from [a-z], which specifies a class containing twenty-six characters.)

Certain characters as used as metacharacters. The following list contains all of the metacharacters and their meanings.

\	Quote the next metacharacter
^	Match the beginning of the line
.	Match any character (except newline)
\$	Match the end of the line (or before newline at the end)
	Alternation
()	Grouping
[]	Character class

Within a regular expression, the following standard quantifiers are recognized:

*	Match 0 or more times
+	Match 1 or more times
?	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

The following character sequences also have special meaning within a regular expression.

\t	tab
\n	newline
\r	return
\033	octal char 003
\x1B	hex char 1B
\w	Match a "word" character (alphanumeric plus "_")
\W	Match a non-word character
\s	Match a whitespace character
\S	Match a non-whitespace character
\d	Match a digit character
\D	Match a non-digit character

Note: we should probably define XML-specific character sequences for things like Nmtoken, Name, etc., as well as ones for the character classes listed in XML 1.0 Appendix B. Character Classes

Regular expressions may also contain the following zero-width assertions:

<code>\b</code>	Match a word boundary
<code>\B</code>	Match a non-(word boundary)

A word boundary (`\b`) is defined as a spot between two characters that has a `\w` on one side of it and a `\W` on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a `\W`.