INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG2001/M7295
July 2001, Sydney

Source: **AIST and University of Brescia**

Title: **Report of CE on Abstraction and Instantiation**

Author: **Kôiti Hasida and Riccardo Leonardi**

Status: **for discussion**

1 Introduction

The CE on Abstraction and Instantiation [1] is reported below according to the workplan proposed at the Singapore meeting [2].

2 Specification

The specification revised in the course of the CE is in the appendix. The new syntax simplifies descriptions a lot compared with the previous version. The new specification employs the idea of representing each scope of instantiation by the `id` value of an `<Instantiation>` element rather than by the element itself. Namely, at any rate we have an `<Instantiation>` element to represent a scope of instantiatoin, but in the new specificatoin each `<Bind>` element encoding a variable binding refers to the `<Instantiation>` encoding the instantiatoin, whereas in the old specification all such `<Bind>` elements are enclosed in the `<Instantiation>` element.

The merit of the new specifiation is at least twofold. First, it requires fewer `id` attributes than the old one does. That is, the old specification needs `id` attributes of the elements which as values bind variables in the `<Abstraction>`, but in the new specification such `id` attributes are not necessary because those elements contain `<Bind>` elements rather than being referenced by them.

Another new feature of the new specification is that it allows default inference in instantiation. Namely, as much information as possible is incorporated from an abstraction to its instantiations, but if there is any contradiction between the variables and the values binding them, then the information in the values overrides that of the variables. This is very useful because it extends the usage of the tools. For instance, you can describe typical structure of, say, a car, in an `<Abstraction>` and instantiate it while overriding some details of that typical structure, for instance to describe a car without a steering wheel.

3 Applicatoin

Abstraction is a very common tool employed in programming languages as reentrant codes such as loops, procedures, functions, macros, and so forth. Computer programming is practically impossible without them. Abstraction generally abound also in traditional documentation, perhaps in technical documentation in particular. Documents introduce new terms and refer to them afterwards to both simplify the description and highlight common strcutures.

Multimedia documents can equally enjoy such functionalities as well. For instance, there are a number of desriptions of the following form, among a lot of others, in the example description in the FCD [3].

```
<Content xsi:type="ImageType" id="soccer1">
    <Image>
        <MediaLocator xsi:type="ImageLocatorType">
            <MediaUri>soccer1.jpg</MediaUri>
        </MediaLocator>
    </Image>
</Content>
```

We can both simplify the whole description and highlight this repetition by replacing each of these description with:

```
<Content><Bind variable="soccer1"/></Content>
```

where the shared strcuture is addressed by the following abstraction.

```
<Abstraction>
    <Content xsi:type="ImageType" id="soccer1">
        <Image>
            <MediaLocator xsi:type="ImageLocatorType">
                <MediaUri>soccer1.jpg</MediaUri>
            </MediaLocator>
        </Image>
    </Content>
</Abstraction>
```

4 Comparative Evaluation

We have tried to compare our new abstraction tool with related tools in MPEG-7 FCD [3]. These tools are the AbstractionLevel DataType and the Graph DS.

Although the AbstractionLevel DataType is claimed to address abstraction and instantiatoin, how to do that is not discussed clearly enough in the FCD. In particular, how to bind each variable to a particular value is not mentioned at all. So it is impossible to compare our tool with the AbstractionLevel DataType in terms of their functionality. Incidentally, a related problem of the AbstractionLevel DataType is that it is unclear how to specify the scope to which a variable belongs. Of course these issues are all addressed with respect to the Abstraction DS and the Bind DataType.

The Graph DS is comparable with our tool in terms of abstraction and instantiation functionalities. Although examples in Section 7.6.1 do not show how to associate Graphs with other types of descriptions, the only possible purpose of this DS seems to be to address correspondences among distinct descriptions.

A correspondence between two descriptions is captured by specifying the scopes of those descriptions and what they share. So three Graphs are necessary: Two (the first two in the example below; let us call them **scoping graphs**) specify the two descriptions and the other one (the last one below; let us call it the **correspondence graph**) describes the structure shared between them, as follows:

```
<Entity id=”A”/>
<Entity id=”B”/>
<Entity id=”C”/>
<Entity id=”D”/>
<Entity id=”E”/>

<Graph> <!-- a scoping graph -->
    <Node id="a" idref="A"/>
    <Node id="b" idref="B"/>
    <Node id="c" idref="C"/>
</Graph>

<Graph> <!-- a scoping graph -->
    <Node id="d" idref="D"/>
    <Node id="e" idref="E"/>
</Graph>

<Graph> <!-- a correspondence graph -->
    <Relation xsi:type="RelationType"
        name="morph" source="#a" target="#e"/>
    <Relation xsi:type="RelationType"
        name="morph" source="#b" target="#d"/>
    <Relation xsi:type="RelationType"
        name="morph" source="#c" target="#e"/>
</Graph>
```

However, the intended meaning of the correspondence does not follow from the semantics of the Graph DS, because the relation `morph` is not defined anywhere.

The Abstraction DS and the Bind DataType clearly distinguish scoping and correspondence:

```
<A><Bind scope=”s1” variable="X"/></A>
<B><Bind scope=”s1” variable="Y"/></B>
<C><Bind scope=”s1” variable="Z"/></C>
<D><Bind scope=”s2” variable="Y"/></D>
<E><Bind scope=”s2” variable="X Z"/></E>

<Abstraction>
    <Instantiation id=”s1”/>
    <Instantiatoin id=”s2”/>
    <Entity id="X"/>
    <Entity id="Y"/>
    <Entity id="Z"/>
</Abstraction>
```

This is more efficient than the Graph-based description for at least the following three reasons.

First, it is simpler in the sense that it uses less `id` attributes, because only the `<Abstraction>` element contains them and the `<Bind>` elements refer to their values, whereas in the Graph-based description the scoping graphs contain `id` attributes and the correspondence graph refer to their values. So there are ten `id` attributes in the previous Graph-based description, whereas there are just five in the above description based on the Abstraction DS and the Bind DataType.

Second, in our approach it is more natural to address the intended meaning by defining the semantics, unlike in the Graph-based approach, which depends on the specific relation name `morph`.

Third, in this connection, the usual semantics of abstraction and instantiation allows you to simplify the two corresponding descriptions by merely attributing what they share to the abstraction, as shown in the following example,

```
<A><Bind scope=”s1” variable="X"/></A>
<B><Bind scope=”s1” variable="Y"/></B>
<C><Bind scope=”s1” variable="Z"/></C>
<D><Bind scope=”s2” variable="Y"/></D>
<E><Bind scope=”s2” variable="X Z"/></E>

<Abstraction>
    <Instantiation id=”s1”/>
    <Instantiatoin id=”s2”/>
    <Entity id="X"/>
    <Entity id="Y"/>
    <Entity id="Z"/>
    <Relation name="cause" source="Z" target="X"/>
</Abstraction>
```

In this example, the below two `<Relation>` elements are entailed and hence may be omitted if they are not referred to elsewhere in the whole description.

```
<Relation name="cause" source="C" target="A"/>
<Relation name="cause" source="E" target="E"/>
```

This superiority of the our approach to the Graph-based one is more evident when we consider a correspondence among three or more descriptions. Although not discussed in 7.6.1 of the FCD, the Graph DS can describe a correspondence among N descriptions for all $N > 1$. To capture a correspondence among N descriptions for $N > 2$, each `<Relation>` element in the correspondence graph must point to N `<Node>`s in the N scoping graphs, as follows. Here the first three Graphs are scoping graphs and the last one is a correspondence graph.

```
<Entity id=”A”/>
<Entity id=”B”/>
<Entity id=”C”/>
<Entity id=”D”/>
<Entity id=”E”/>
<Entity id=”F”/>
<Entity id=”G”/>
<Entity id=”H”/>
```

```
<Graph>
    <Node id="a" idref="A"/>
    <Node id="b" idref="B"/>
    <Node id="c" idref="C"/>
</Graph>

<Graph>
    <Node id="d" idref="D"/>
    <Node id="e" idref="E"/>
</Graph>

<Graph>
    <Node id="f" idref="F"/>
    <Node id="g" idref="G"/>
    <Node id="h" idref="H"/>
</Graph>

<Graph>
    <Relation xsi:type="RelationType" name="morph">
        <Argument>#a</Argument>
        <Argument>#e</Argument>
        <Argument>#f</Argument>
    </Relation>
    <Relation xsi:type="RelationType" name="morph">
        <Argument>#b</Argument>
        <Argument>#d</Argument>
        <Argument>#g</Argument>
    </Relation>
    <Relation xsi:type="RelationType" name="morph">
        <Argument>#c</Argument>
        <Argument>#e</Argument>
        <Argument>#h</Argument>
    </Relation>
</Graph>
```

Compare this with the following description in the abstraction/instantiation approach.

```
<A><Bind scope=”i1” variable=”X”/></A>
<B><Bind scope=”i1” variable=”Y”/></B>
<C><Bind scope=”i1” variable=”Z”/></C>

<D><Bind scope=”i2” variable=”Y”/></D>
<E><Bind scope=”i2” variable=”X Z”/></E>

<F><Bind scope=”i3” variable=”X”/></F>
<G><Bind scope=”i3” variable=”Y”/></G>
<H><Bind scope=”i3” variable=”Z”/></H>

<Abstraction>
    <Instantiation id=”i1”/>
    <Instantiation id=”i2”/>
    <Instantiation id=”i3”/>
    <Entity id="X"/>
```

```
    <Entity id="Y"/>
    <Entity id="Z"/>
</Abstraction>
```

The conclustion of this comparison is that we have better reason to incorporate the Abstraction DS and the Bind DataType into MPEG-7 if we have any reason to incorporate the Graph DS, as the usage of the Graph DS other than the description of correspondence is unclear.

Reference

[1] Kôiti Hasida and Riccardo Leonardi. *Abstraction and Instantiation*. ISO/MPEG M7048, MPEG Singapore Meeting, March 2001.

[2] Kôiti Hasida and Riccardo Leonardi. *Workplan of CE on Abstraction and Instantiation*. ISO/MPEG W3970, MPEG Singapore Meeting, March 2001.

[3] Peter van Beek, Ana B. Benitez, Joerg Heuer, Jose Martinez, Philippe Salembier, Yoshiaki Shibata, John R. Smith, Toby Walker. *Text of 15938-5 FCD Information Technology – Multimedia Content Description Interface – Part 5 Multimedia Description Schemes*. ISO/MPEG N3966.

Appendix: Revised Specification of the Tools

introduction

Discussed below are tools for describing abstractions and their instantiations. The merit of abstraction is the economy of description and clarification of structure sharing. Abstractions allow you to avoid repeating similar descriptions over and over again. You write one abstraction to address the shared structure of descriptions and reuse it when you want similar descriptions. Suppose you want to describe many occurrences of a common pattern of events and states of affairs, such as a type of configuration of soccer players in the field. You will describe the positions of eleven or twenty two people to address this configuration. Once you set up an abstraction to describe this common pattern, you can address each occurrence of the pattern by just binding at most the eleven or twenty two non-specific people to particular players, without repeating the descriptions of their positions.

The Abstraction DS and the Bind DataType

The Abstraction DS addresses how to describe abstractions, and the Bind DataType addresses how to instantiate those abstractions. That is, an `<Abstraction>` element defines a type of descriptions, and a `<Bind>` element describes the binding of a variable in an instantiation of that type. An `<Abstraction>` element may contain `<Instantiation>` elements, each of which declares an instantiation of the abstraction. The `<Instantiation>` elements are referenced by `<Bind>` elements. The instantiation consists of the bindings referring to it. A `<Bind>` element refers an `<Instantiatoin>` and variables to bind.

The Abstraction DS and the Bind DataType are meta-level tools. That is, the content of an Abstraction element makes no sense as it is, but is interpreted only after instantiation. In

particular, `AbstractionLevel` is not interpreted inside an `<Abstraction>` element. Of course the ID values defined in an `<Abstraction>` element are substituted in an instantiation.

Here is the syntax of the DS and the DataType:

```
<!-- ################################################  -->
<!-- Definition of the Abstraction DS                 -->
<!-- ################################################  -->

<complexType name="AbstractionType">
    <complexContent>
        <extension base="mpeg7:DSType">
            <sequence>
                <element name="Instantiation"
                      minOccurs="0" maxOccurs="unbounded">
                    <complexContent>
                        <attribute name=”id” type=”ID” use=”required”/>
                    </complexContent>
                </element>
                <choice minOccurs="1" maxOccurs="unbounded">
                    <element name=”Entity” type="mpeg7:DSType"/>
                    <element name=”Relation” type="mpeg7:RelationType"/>
                    <element name=”NaryRelation”
                       type="mpeg7:NaryRelationType"/>
                </choice>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<!-- ################################################  -->
<!-- Definition of the Bind DataType                  -->
<!-- ################################################  -->

<complexType name="BindType">
    <complexContent>
        <element name=”Entity” type=”mpeg7:DSType”
            minOccurs=”0” maxOccurs=”1”/>
        <attribute name=”scope” type=”IDREF” use=”optional”/>
        <attribute name=”variable” type=”IDREFS” use=”required”/>
    </complexContent>
</complexType>
```

Semantics for AbstractionType:

Name	*Definition*
`AbstractionType`	Type of abstraction. The elements in it are variables. All the descendant DS instances are variables, representing nonspecific entities. Elements of this type may appear as children of elements of DSType.
`Instantiation`	An instantiation of the abstraction. This element just introduces the `id` value by which to identify the scope an instantiation, which is a set of bindings. An instantiation with only one binding need not be encoded by an `<Instantiation>` element. An instantiation copies the

Name	*Definition*
	content (except for <Instantiation> elements) of the <Abstraction> by substituting its parts as specified by the <Bind> elements referring to this <Instantiatoin>. The information in a variable is incorporated into the binding value as much as consistent with the information already in the value. Each ID values in the copied description is deleted and replaced by the corresponding ID values if any. See the example given later.

Semantics of BindType:

Name	*Definition*
BindType	Type of binding of variables in instantiation of abstraction. A <Bind> element may appear as a child of any DS instance and specifies that its parent or child element is the value to bind the variables referred to by the variable attribute. Elements of this type may appear as children of elements of DSType.
Entity	The binding value. If missing, the binding value is the parent element.
scope	Refers to the <Instantiation> which specifies the scope of the instantiation in which to bind the variables (referred to by the variable attribute) to the parent element. If missing, the scope of the instantiation consists of the current binding only.
variable	Refers to the variables to bind. The id values contained in the value of a variable attribute must be those of elements in the <Abstraction> containing the <Instantiation> referred to by the scope attribute. (See the example below.) This condition is not checked by the parser but dealt with by the applicatoin program such as a search engine.

Example

The following is an abstraction which may mean that a woman kisses a man. The kisser object, the kissed object, and the kissing event are all variables (i.e., non-specific entities), because they are descendant elements of the <Abstraction>.

```
<Abstraction>
    <Instantiation id="kiss1"/>
    <Instantiation id="kiss2"/>
    <Entity type="Mpeg7:ObjectType" id="kisser"/>
    <Entity type="Mpeg7:ObjectType" id="kissed"/>
    <Entity type="Mpeg7:EventType" id="kissing">
        <Relation name="agent" target="kisser"/>
        <Relation name="theme" target="kissed"/>
    </Entity>
</Abstraction>
```

Each <Instantiation> elements above declares an instantiation of this abstraction. These instantiatoins may be embodied as follows.

```
<Object id="mary">
    <Bind scope="kiss1" variable="kisser"/>
    <Bind scope="kiss2" variable="kissed"/>
</Object>

<Object id="tom">
    <Bind scope="kiss1" variable="kissed"/>
</Object>

<Event id="kiss1">
    <Bind scope="kiss2" variable="kissing"/>
</Event>
```

The following is an equivalent description.

```
<Object id="mary">
    <Bind scope="kiss1" variable="kisser"/>
    <Bind scope="kiss2" variable="kissed"/>
</Object>

<Bind scope="kiss1" variable="kissed"><Object id="tom"/></Bind>

<Bind scope="kiss2" variable="kissing"><Event id="kiss1"/></Bind>
```

Note that the condition described above in the semantics of the variable attribute holds here. In the first <Bind> element, for instance, the variable attribute refers (via id value kisser) to an element in the <Abstraction> which is the parent of the <Instantiation> referred to (via kiss1) by the scope attribute.

Each of the above descriptions entails that Mary kisses Tom and that somebody kisses Mary. More precisely, abstraction kiss1 means that Mary kisses tom, and kiss2 means that somebody kisses Mary. That is, the below description follows. The <Event> element below replaces the <Event> element above. So the search engine should be able to find the above description in response to a query for "Mary kisses Tom" or "Mary is kissed".

```
<Entity type="Mpeg7:DSType">
    <Relation name="agent" target="mary"/>
    <Relation name="patient" target="tom"/>
</Entity>

<Event id="kiss1">
    <Relation name="theme" target="mary"/>
</Event>
```