Use of Architectural Forms for Early to Late bound mapping

Author: Robin La Fontaine
Date: 18 Oct 1999
Updated 21 October 1999

Purpose

This document explains the basics of SGML Architectures as needed to represent the relationship between the early-bound and late-bound XML formats for Express-driven data. There are many aspects of Architectures that are not covered here: only those relevant to the structural mapping required within STEP are discussed.

This document is an informal description of this area. When the details are agreed, a more formal specification will be developed for inclusion in the standards document.

Terminology for Architectures

AFDR: Architectural Forms Definition Requirements is the document describing architectural forms, Annex A.3, ISO/IEC 10744, 2nd ed., see HyTime web site www.hytime.org).

Architectural Engine: software specifically written to process architectural forms.

Architectural forms: rules for creating and processing components of documents.

Base architecture: a collection of architectural forms that may apply to a document described as a single meta-DTD.

Client DTD: the DTD of a document that uses architectural forms, also known as a **derived DTD**.

Basic Principles

Given a document in XML which corresponds with a particular DTD, architectural forms provide a standard mechanism for viewing it as if it were consistent with another DTD (the meta-DTD or base architecture).

This is being used within STEP to allow one or more early-bound data sets to be viewed as if they were defined in terms of the standard late-bound DTD. Thus software written against the late-bound DTD can, without modification, process data that complies with any compliant early-bound DTD. ‘Compliant’ here means that the early-bound DTD has the late-bound DTD as its base architecture.

This gives some flexibility in defining early-bound DTDs which can be optimised for different purposes, e.g., for display, for data exchange, for compactness.

The principle flexibility that this gives to an early-bound format are as follows:

- Choice of names for element types
- Choice of names for xml-attributes
- Choice of using xml-attributes or content for data items

- Ability to define additional ‘wrapper’ element types which do not appear in the base architecture
- Ability to define extra data which does not appear at all in the base architecture

However, architectural forms do not give complete flexibility in terms of structure – it is not a completely general mapping of one DTD to another (XSLT can be used for more complex mappings). The most important restriction that applies is that the structure of the client DTD must be consistent with that of the base architecture DTD.

What sort of flexibility does this provide?

An example is used to illustrate just how far this provides flexibility in defining an early-bound DTD for a specific model. There are different aspects of this as noted above and there is an illustration of each of the following:

- Element type names
- Attribute names
- Attribute or content for data items
- Additional ‘wrapper’ element types
- Extra data not in the architectural form DTD
- Nesting subtypes in a tree structure
- Using in-line instance definitions

4.1 Example for illustration

Start with a simple schema[1]

```
SCHEMA pets;

ENTITY pet;
SUPERTYPE OF (ONEOF(dog, cat, chinchilla));
  name : STRING;
  owner : person;
END_ENTITY;

ENTITY person ;
  name : STRING;
END_ENTITY;

ENTITY dog SUBTYPE  OF (pet);
END_ENTITY;
ENTITY cat  SUBTYPE  OF (pet);
END_ENTITY;
ENTITY  chinchilla  SUBTYPE  OF (pet);
END_ENTITY;

END_SCHEMA;
```

The data population for the example is based on Nigel Shaw’s household. There are two sons, Iain and Andrew, who each own a pet, respectively Rita the chinchilla and Maddy the dog. There is also a cat (Whiskey) which they consider to belong to Nigel.

[1] This example was developed by Nigel Shaw.

Instances using part 21:

```
#1=CAT('Whiskey', #10);
#2=DOG('Maddy', #11);
#3=CHINCHILLA('Rita', #12);

#10=PERSON('Nigel Shaw');
#11=PERSON('Andrew Shaw');
#12=PERSON('Iain Shaw');
```

The corresponding late-bound data for XML (without the model itself) would be as follows:

```
<?xml version="1.0"?>
<!DOCTYPE ISO-10303-data SYSTEM "late-bound-dtd-v5-exp.dtd">
<ISO-10303-data>
 <data data_id="data_set_1">
  <schema_instance express_schema_name="pets">
   <constant_instances>
   </constant_instances>
   <non_constant_instances>
    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E1">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E10"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype express_entity_name="cat">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>

    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E2" >
      <attribute_instance express_attribute_name="name">
       <string_literal>Maddy</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E11"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype express_entity_name="dog">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>

    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E3">
      <attribute_instance express_attribute_name="name">
       <string_literal>Rita</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E12"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype
express_entity_name="chinchilla">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E10">
     <attribute_instance express_attribute_name="name">
      <string_literal>Nigel Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E11">
     <attribute_instance express_attribute_name="name">
      <string_literal>Andrew Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E12">
     <attribute_instance express_attribute_name="name">
      <string_literal>Iain Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
   </non_constant_instances>
  </schema_instance>
 </data>
</ISO-10303-data>
```

An early-bound data file for this example (developed along the lines of the PDML algorithm) is shown below:

```
<?xml version="1.0"?>
<!DOCTYPE pets SYSTEM "F:\design docs\DTD development\master DTD
tables\late bound data\pets-eb-v1.dtd">
<pets id="Example_schema_1">
 <pet id="E1">
  <pet.name>
   <string>Whiskey</string>
  </pet.name>
  <pet.owner><person-ref refid="E10"/>
  </pet.owner> <cat id="E1.1"/>
 </pet>
 <pet id="E2">
  <pet.name>
   <string>Maddy</string>
  </pet.name>
  <pet.owner><person-ref refid="E11"/>
  </pet.owner><dog id="E2.1"/>
 </pet>
 <pet id="E3">
  <pet.name>
   <string>Rita</string>
  </pet.name>
  <pet.owner><person-ref refid="E12"/>
  </pet.owner><chinchilla id="E3.1"/>
 </pet>
 <person id="E10">
  <person.name>
   <string>Nigel Shaw</string>
  </person.name>
 </person>
 <person id="E11">
  <person.name>
   <string>Andrew Shaw</string>
  </person.name>
 </person>
 <person id="E12">
  <person.name>
   <string>Iain Shaw</string>
  </person.name>
 </person>
</pets>
```

4.2 General

In the DTD for our early-bound data the following declaration is needed:

```
<?IS10744:arch name=”EX-AP-1”
      dtd-system-id="late-bound-dtd-v4.dtd"
      form-att=”late-bound-element”
      renamer-att=”late-bound-rename”
      suppressor-att=”late-bound-processing”
      ?>
```

This says that the DTD corresponds with the architectural form DTD "late-bound-dtd-v4.dtd". The name of the architecture is “EX-AP-1” . An xml-attribute with the name **late-bound-element** indicates the name of the corresponding form in the base architecture. Renaming of xml-attributes is achieved using an xml-attribute named **late-bound-rename**. Architectural processing can be controlled using an xml-attribute named **late-bound-processing**. All these names can be changed of course to avoid any name clashes with existing xml-attributes.

4.3 Element type names

Looking at a possible representation of an instance of the **pet** entity, the late-bound and early-bound segments are shown below.

Late-bound segment:

```
<nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E1">
      …
</nested_complex_entity_instance>
```

Corresponding early-bound segment:

```
<pet id="E1">
      …
</pet>
```

In the DTD, in order to indicate that the element type **pet** corresponds with the late-bound element type **nested_complex_entity_instance**, it is necessary to have these xml-attributes defined:

```
<!ATTLIST pet
      late-bound-element NMTOKEN #FIXED “nested_complex_entity_instance”
      express_entity_name CDATA #FIXED “pet”>
```

The late-bound-element xml-attribute indicates that the element type **pet** in the early-bound DTD maps to the element type **nested_complex_entity_instance** in the architectural DTD. The value of the **express_entity_name** xml-attribute for this element is “**pet**”. A different xml-attribute name could have been used here, and the mapping would become more complex.

The early-bound xml-attribute named **id** corresponds with the late-bound xml-attribute **entity_instance_id**. By default these will be mapped by the architectural engine because they are both of type ID, and xml-attributes of this particular type are automatically mapped. However, to be explicit, the following xml-attribute should also be included:

```
<!ATTLIST pet
      late-bound-rename CDATA #FIXED “id entity_instance_id”>
```

In a similar way, the nested subtype can also be mapped from the late bound:

```
<nested_complex_entity_instance_subtype express_entity_name="cat">
     </nested_complex_entity_instance_subtype>
```

to the corresponding early-bound:

```
<cat id="E1.1"/>
```

This is achieved using the following xml-attribution:

```
<!ATTLIST cat
      late-bound-element NMTOKEN #FIXED
“nested_complex_entity_instance_subtype”
      express_entity_name CDATA #FIXED “cat”>
```

4.4 Express-Attribute names

Late-bound segment:

```
<nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E1">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E10"/>
      </attribute_instance>
      …
       </nested_complex_entity_instance>
```

Corresponding early-bound segment:

```
<pet id="E1">
  <pet.name>
   <string>Whiskey</string>
  </pet.name>
  <pet.owner><person-ref refid="E10"/>
  </pet.owner> <cat id="E1.1"/>
 </pet>
```

In the DTD, in order to indicate that the element type **pet.name** corresponds with the late-bound element type **attribute_instance**, these xml-attributes are defined:

```
<!ATTLIST pet.name
      late-bound-element NMTOKEN #FIXED “attribute_instance”
      express_attribute_name CDATA #FIXED “name”>
```

Similarly for the other items:

```
<!ATTLIST string
      late-bound-element NMTOKEN #FIXED “string_literal”>
```

```
<!ATTLIST pet.owner
      late-bound-element NMTOKEN #FIXED “attribute_instance”
      express_attribute_name CDATA #FIXED “owner”>
<!ATTLIST person-ref
      late-bound-element NMTOKEN #FIXED “entity_instance_ref”
      late-bound-rename CDATA #FIXED “refid entity_instance_idref”>
```

4.5 Attribute or content for data items

If the following early-bound representation had been used instead for the above Express-attribute:

```
<string value=”Maddy”/>
```

This could be achieved by changing the xml-attribution for **string** as follows:

```
<!ATTLIST string
      late-bound-element NMTOKEN #FIXED “string_literal”
      late-bound-rename CDATA #FIXED “#ARCCONT value”>
```

This can be made to work the other way round also, so that content in the early-bound can become an xml-attribute value in the late-bound.

4.6 Additional ‘wrapper’ element types

It is possible to introduce additional wrapper element types in the early-bound DTD and indicate that these are to be ignored by the architectural engine. For example, it might be useful to wrap the Express-attribute values in an element **pet_details**:

```
<pet …>
      <pet_details>
       <pet.name>
            …
       </pet.name>
       …
      </pet_details>
</pet>
```

This is achieved this us by using the xml-attribution:

```
<!ATTLIST pet_details
      late-bound-processing NMTOKEN #FIXED “sArcForm”>
<!ATTLIST pet.name
      late-bound-processing NMTOKEN #FIXED “sArcNone”>
<!ATTLIST pet.owner
      late-bound-processing NMTOKEN #FIXED “sArcNone”>
```

This tells the architectural engine not to represent **pet_details** in the result, in other words suppress architectural processing. But the architectural engine continues to look through the descendants until it finds a form which says ‘do not suppress architectural processing’, which is indicated by the value “sArcNone”.

4.7 Extra data not in the architectural form DTD

If there is in the DTD some additional data that is not represented in the architectural DTD, then by default it will be ignored provided its name does not correspond with an element type that is allowed in the same place in the architectural DTD. This can be controlled more explicitly, which is probably good practice, using the xml-attribute described above, the **suppressor-att** which in our case has been renamed **late-bound-processing**.

4.8 Nesting subtypes in a tree structure

In the late-bound DTD that there are two possible representations for entity instances. One of these has the subtypes represented as a flat list of items within the complex entity instance, and the other has a more structured representation using nesting to start from an entity which is not a subtype of any other and working down through the sub/super type hierarchy.

This is an instance of the **cat** entity represented as a nested structure, where **pet,** which is not a subtype of any other entity, is the top entity:

```
<nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E1">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E10"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype express_entity_name="cat">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>
```

And this is the same represented using a flat structure (in this structure the ordering of the **partial_entity_instance** elements is not significant):

```
<flat_complex_entity_instance entity_instance_id="E1">
     <partial_entity_instance express_entity_name="cat"/>
     <partial_entity_instance express_entity_name="pet">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E10"/>
      </attribute_instance>
     </partial_entity_instance>
    </flat_complex_entity_instance>
```

The structure that is appropriate for any given situation will depend on the requirements, but the important point relating to architectural mapping is that an early-bound structure can map to either of these structures. An early-bound representation for the nested structure has been discussed. An example of the data with an early-bound DTD corresponding to the flat structure is given below.

```
<cat-chinchilla-dog-pet id="E1">
  <pet id="E1.2">
   <pet.name>
    <string>Whiskey</string>
   </pet.name>
   <pet.owner><person-ref refid="E10"/>
   </pet.owner>
  </pet>
  <cat id="E1.1"/>
 </cat-chinchilla-dog-pet>
```

The xml-attributes to map the Express-attributes would be the same as before. The xml-attributes for the structural parts would be as follows:

```
<!ATTLIST cat-chinchilla-dog-pet
      late-bound-element NMTOKEN #FIXED “flat_complex_entity_instance”>
<!ATTLIST cat
      late-bound-element NMTOKEN #FIXED “partial_entity_instance”>
      express_entity_name CDATA #FIXED “cat”>
<!ATTLIST pet
      late-bound-element NMTOKEN #FIXED “partial_entity_instance”>
      express_entity_name CDATA #FIXED “pet”>
```

4.9 Using in-line instance definitions

Consider a slightly different representation of the model where the person has an Express-attribute which details his or her pets, and the pet has an INVERSE Express-attribute for its owner (one, which is required) as follows:

```
SCHEMA pets;

ENTITY pet;
SUPERTYPE OF (ONEOF(dog, cat, chinchilla));
  name : STRING;
INVERSE
  owner : person FOR pets;
END_ENTITY;

ENTITY person ;
  name : STRING;
  pets : OPTIONAL SET[1:?] OF pet;
END_ENTITY;

ENTITY dog SUBTYPE  OF (pet);
END_ENTITY;
ENTITY cat  SUBTYPE  OF (pet);
END_ENTITY;
ENTITY  chinchilla  SUBTYPE  OF (pet);
END_ENTITY;

END_SCHEMA;
```

A very reasonable representation of this in XML would be one in which the **pet** instances occur in the file nested within the **person** entity instance. For example:

```
#1=CAT('Whiskey');
#2=DOG('Maddy');
#3=CHINCHILLA('Rita');

#10=PERSON('Nigel Shaw',(#1));
#11=PERSON('Andrew Shaw',(#2));
#12=PERSON('Iain Shaw',(#3));
```

becomes as a late-bound XML file:

```
<?xml version="1.0"?>
<!DOCTYPE ISO-10303-data SYSTEM "late-bound-dtd-v5-exp.dtd">
<ISO-10303-data>
 <data data_id="data_set_1">
```

```
<schema_instance express_schema_name="Example_schema_1">
 <constant_instances>
 </constant_instances>
 <non_constant_instances>
  <nested_complex_entity_instance express_entity_name="person"
   entity_instance_id="E10">
   <attribute_instance express_attribute_name="name">
    <string_literal>Nigel Shaw</string_literal>
   </attribute_instance>
   <attribute_instance express_attribute_name="pets">
    <set_literal>
     <nested_complex_entity_instance express_entity_name="pet"
      entity_instance_id="E1">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <nested_complex_entity_instance_subtype
       express_entity_name="cat">
      </nested_complex_entity_instance_subtype>
     </nested_complex_entity_instance>
    </set_literal>
   </attribute_instance>
  </nested_complex_entity_instance>
  <nested_complex_entity_instance express_entity_name="person"
   entity_instance_id="E11">
   <attribute_instance express_attribute_name="name">
    <string_literal>Andrew Shaw</string_literal>
   </attribute_instance>
   <attribute_instance express_attribute_name="pets">
    <set_literal>
     <nested_complex_entity_instance express_entity_name="pet"
      entity_instance_id="E2">
      <attribute_instance express_attribute_name="name">
       <string_literal>Maddy</string_literal>
      </attribute_instance>
      <nested_complex_entity_instance_subtype
       express_entity_name="dog">
      </nested_complex_entity_instance_subtype>
     </nested_complex_entity_instance>
    </set_literal>
   </attribute_instance>
  </nested_complex_entity_instance>
  <nested_complex_entity_instance express_entity_name="person"
   entity_instance_id="E12">
   <attribute_instance express_attribute_name="name">
    <string_literal>Iain Shaw</string_literal>
   </attribute_instance>
   <attribute_instance express_attribute_name="pets">
    <set_literal>
     <nested_complex_entity_instance express_entity_name="pet"
      entity_instance_id="E3">
      <attribute_instance express_attribute_name="name">
       <string_literal>Rita</string_literal>
      </attribute_instance>
      <nested_complex_entity_instance_subtype
       express_entity_name="chinchilla">
      </nested_complex_entity_instance_subtype>
     </nested_complex_entity_instance>
    </set_literal>
   </attribute_instance>
  </nested_complex_entity_instance>
```

```
   </non_constant_instances>
  </schema_instance>
 </data>
</ISO-10303-data>
```

In the above there are no references although the **entity_instance_id** has been preserved for each entity instance as it is a required xml-attribute.

Taking a section of this, representing the first person of the three, this is an example of an equivalent early-bound version (n.b. this example not yet checked against a dtd):

```
<person id="E10">
  <person.name>
   <string>Nigel Shaw</string>
  </person.name>
  <person.pets>
    <pet-set>
     <pet id="E1">
      <pet.name>
       <string>Whiskey</string>
      </pet.name>
     <cat id="E1.1"/>
     </pet>
    </pet-set>
   </person.pets>
 </person>
```

This can also be mapped back to the late-bound representation using the following xml-attribution:

```
<!ATTLIST person
      late-bound-element NMTOKEN #FIXED “nested_complex_entity_instance”>
      express_entity_name CDATA #FIXED “person”>
<!ATTLIST person.name
      late-bound-element NMTOKEN #FIXED “attribute_instance”
      express_attribute_name CDATA #FIXED “name”>
<!ATTLIST string
      late-bound-element NMTOKEN #FIXED “string_literal”>
<!ATTLIST person.pets
      late-bound-element NMTOKEN #FIXED “attribute_instance”
      express_attribute_name CDATA #FIXED “pets”>
<!ATTLIST pet_set
      late-bound-element NMTOKEN #FIXED “set_literal”>
<!ATTLIST pet
      late-bound-element NMTOKEN #FIXED “nested_complex_entity_instance”>
      etc.
```

(TBA: Look also at nesting of types in a complex type tree.)

Late-bound Express data DTD

This DTD is for the data section in an XML file which might also contain the full Express model. This DTD will be integrated with the Express language DTD but these are kept separate at this time for ease of development.

```
<!-- Late binding data section of XML based on Eliot's original.
Suitably improved/corrupted by Robin La Fontaine
Version 1, 7 Sept 1999

Version 2, 11 October 1999: Updated using Eliot Kimber's architectural DTD
for late-bound data which was developed at the Early-binding workshop at
Long Beach, CA, in September 1999.

Version 3: intermediate

Version 4: 19 Oct 1999: Changed in order to allow early to late mapping
with architectures. This meant that some of the sub-element content had to
be brought up into xml-attributes in order to be able to do the mapping.

Version 5: 21 Oct 1999: Changed entity instance IDs and refs to be just
one type rather than distinguish between
partial/complex/nested/nested_subtype which was unnecessarily complicated.
Also made entity_literal into an xml-entity to get rid of an unnecessary
layer of nesting in the data.
-->

<!--==============================================
    Items brought over from main language schema
    ==============================================
(I know there are other ways to do this, but this is the simplest
for the software I am using!) -->

<!ENTITY % literal ' binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal '>

<!ELEMENT enumeration_ref (#PCDATA)>
<!ELEMENT string_literal (#PCDATA)>
<!ELEMENT integer_literal (#PCDATA)>
<!ELEMENT real_literal (#PCDATA)>
<!ELEMENT binary_literal (#PCDATA)>
<!ELEMENT logical_literal (false | true | unknown)>
<!ELEMENT false EMPTY>
<!ELEMENT true EMPTY>
<!ELEMENT unknown EMPTY>

<!-- data is brought over and modified here -->
<!ELEMENT data
  (schema_instance)
>

<!ATTLIST data
  data_id ID #REQUIRED>

<!--==============================================
    Organizing Parameter Entities
    ==============================================-->

<!ENTITY % entity_instance
  "nested_complex_entity_instance |
   flat_complex_entity_instance"
>

<!ENTITY % entity_literal
  "%entity_instance; |
   entity_instance_ref"
>


<!ENTITY % simple_value
  "%literal;

   bag_literal |
   list_literal |
   set_literal |
   array_literal |

   type_literal |
   %entity_literal;"
>

<!ENTITY % aggregate-contents
  "binary_literal* |
   integer_literal* |
   logical_literal* |
   real_literal* |
   string_literal* |

   bag_literal* |
   list_literal* |
   set_literal* |
   array_literal* |

   type_literal* |
   (%entity_literal;)*"
>

<!ENTITY % schema-ref-att
  "express_schema_name NMTOKEN #REQUIRED">

<!ENTITY % schema-ref-att-opt
  "express_schema_name NMTOKEN #IMPLIED">

<!ENTITY % entity-ref-att
  "express_entity_name NMTOKEN #REQUIRED">

<!ENTITY % type-ref-att
  "express_type_name NMTOKEN #REQUIRED">

<!ENTITY % attribute-ref-att
  "express_attribute_name NMTOKEN #REQUIRED">

<!--==============================================
    Main section of DTD
    ==============================================-->
```

```
<!ELEMENT ISO-10303-data
  (data+)
>

<!ELEMENT schema_instance
  (constant_instances,
   non_constant_instances)
>

<!ATTLIST schema_instance
  %schema-ref-att;>

<!ELEMENT constant_instances
  (%entity_instance;)*
>

<!ELEMENT non_constant_instances
  (%entity_instance;)*
>

<!ELEMENT flat_complex_entity_instance
  (partial_entity_instance+)
>
<!ATTLIST flat_complex_entity_instance
  entity_instance_id ID #REQUIRED>

<!ELEMENT nested_complex_entity_instance
  (attribute_instance*,
   nested_complex_entity_instance_subtype*)
>

<!ATTLIST nested_complex_entity_instance
  entity_instance_id ID #REQUIRED>

<!ATTLIST nested_complex_entity_instance
  %entity-ref-att;
  %schema-ref-att-opt;>

<!ELEMENT nested_complex_entity_instance_subtype
  (attribute_instance*,
   nested_complex_entity_instance_subtype*)
>

<!ATTLIST nested_complex_entity_instance_subtype
  %entity-ref-att;
  %schema-ref-att-opt;
  entity_instance_id ID #IMPLIED
>

<!ELEMENT partial_entity_instance
  (attribute_instance*)
>

<!ATTLIST partial_entity_instance
  %entity-ref-att;
  %schema-ref-att-opt;
  entity_instance_id ID #IMPLIED
>

<!ELEMENT attribute_instance
  (%simple_value;)
```

```
>

<!ATTLIST attribute_instance
  %attribute-ref-att;
>


<!NOTATION uuencode PUBLIC "uuencoded data" >
<!NOTATION mime     PUBLIC "mime encoded data" >
<!NOTATION base64   PUBLIC "base-64 encoded data" >

<!ATTLIST binary_literal
  notation
    NOTATION
    (uuencode |
     mime |
     base64)
    "uuencode"
>

<!ELEMENT set_literal
  (%aggregate-contents;)
>

<!ELEMENT list_literal
  (%aggregate-contents;)
>

<!ELEMENT bag_literal
  (%aggregate-contents;)
>

<!ELEMENT array_literal
  (%aggregate-contents;)
>

<!ELEMENT type_literal
  (%simple_value; | enumeration_ref)
>

<!ATTLIST type_literal
  %type-ref-att;
  %schema-ref-att-opt;
>

<!ELEMENT entity_instance_ref
  EMPTY
>

<!ATTLIST entity_instance_ref
  entity_instance_idref IDREF #REQUIRED>
```

Late-bound Express data DTD expanded with Used-by

This is the same DTD as above but showing the entities and element types fully expanded. Element types are in alphabetical order and comments indicate where each is used.

```
<!-- DTD elements: total number = 25
Referenced not defined: NIL
Defined not referenced: (ISO-10303-data)
Duplicate definitions: NIL
 -->
<!ENTITY % literal '  binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  '>
<!ATTLIST data
      data_id ID #REQUIRED >
<!ENTITY % entity_instance ' nested_complex_entity_instance |
   flat_complex_entity_instance '>
<!ENTITY % entity_literal '  nested_complex_entity_instance |
   flat_complex_entity_instance  |
   entity_instance_ref '>
<!ENTITY % simple_value '   binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal

   bag_literal |
   list_literal |
   set_literal |
   array_literal |

   type_literal |
     nested_complex_entity_instance |
   flat_complex_entity_instance  |
   entity_instance_ref  '>
<!ENTITY % aggregate-contents ' binary_literal* |
   integer_literal* |
   logical_literal* |
   real_literal* |
   string_literal* |

   bag_literal* |
   list_literal* |
   set_literal* |
   array_literal* |

   type_literal* |
   (  nested_complex_entity_instance |
   flat_complex_entity_instance  |
   entity_instance_ref )* '>
<!ENTITY % schema-ref-att ' express_schema_name NMTOKEN #REQUIRED '>
<!ENTITY % schema-ref-att-opt ' express_schema_name NMTOKEN #IMPLIED '>
<!ENTITY % entity-ref-att ' express_entity_name NMTOKEN #REQUIRED '>
<!ENTITY % type-ref-att ' express_type_name NMTOKEN #REQUIRED '>
<!ENTITY % attribute-ref-att ' express_attribute_name NMTOKEN #REQUIRED '>
<!ATTLIST schema_instance
      express_schema_name NMTOKEN #REQUIRED >
```

```
<!ATTLIST flat_complex_entity_instance
      entity_instance_id ID #REQUIRED >
<!ATTLIST nested_complex_entity_instance
      entity_instance_id ID #REQUIRED >
<!ATTLIST nested_complex_entity_instance
      express_entity_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED >
<!ATTLIST nested_complex_entity_instance_subtype
      express_entity_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED
      entity_instance_id ID #IMPLIED >
<!ATTLIST partial_entity_instance
      express_entity_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED
      entity_instance_id ID #IMPLIED >
<!ATTLIST attribute_instance
      express_attribute_name NMTOKEN #REQUIRED >
<!NOTATION uuencode PUBLIC "uuencoded data">
<!NOTATION mime PUBLIC "mime encoded data">
<!NOTATION base64 PUBLIC "base-64 encoded data">
<!ATTLIST binary_literal
      notation  (uuencode | mime | base64) "uuencode" >
<!ATTLIST type_literal
      express_type_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED >
<!ATTLIST entity_instance_ref
      entity_instance_idref IDREF #REQUIRED >

<!ELEMENT array_literal (binary_literal* | integer_literal* |
logical_literal* | real_literal* | string_literal* | bag_literal* |
list_literal* | set_literal* | array_literal* | type_literal* |
(nested_complex_entity_instance | flat_complex_entity_instance |
entity_instance_ref)*)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT attribute_instance (binary_literal | integer_literal |
logical_literal | real_literal | string_literal | bag_literal |
list_literal | set_literal | array_literal | type_literal |
nested_complex_entity_instance | flat_complex_entity_instance |
entity_instance_ref)>
            <!-- Used by: (nested_complex_entity_instance
nested_complex_entity_instance_subtype partial_entity_instance) -->

<!ELEMENT bag_literal (binary_literal* | integer_literal* |
logical_literal* | real_literal* | string_literal* | bag_literal* |
list_literal* | set_literal* | array_literal* | type_literal* |
(nested_complex_entity_instance | flat_complex_entity_instance |
entity_instance_ref)*)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT binary_literal (#PCDATA)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT constant_instances (nested_complex_entity_instance |
flat_complex_entity_instance)*>
            <!-- Used by: (schema_instance) -->

<!ELEMENT data (schema_instance)>
```

```
            <!-- Used by: (ISO-10303-data) -->

<!ELEMENT entity_instance_ref EMPTY>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT enumeration_ref (#PCDATA)>
            <!-- Used by: (type_literal) -->

<!ELEMENT false EMPTY>
            <!-- Used by: (logical_literal) -->

<!ELEMENT flat_complex_entity_instance (partial_entity_instance+)>
            <!-- Used by: (array_literal attribute_instance bag_literal
constant_instances list_literal non_constant_instances set_literal
type_literal) -->

<!ELEMENT integer_literal (#PCDATA)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT ISO-10303-data (data+)>
            <!-- Used by: NIL -->

<!ELEMENT list_literal (binary_literal* | integer_literal* |
logical_literal* | real_literal* | string_literal* | bag_literal* |
list_literal* | set_literal* | array_literal* | type_literal* |
(nested_complex_entity_instance | flat_complex_entity_instance |
entity_instance_ref)*)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT logical_literal (false | true | unknown)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT nested_complex_entity_instance (attribute_instance*,
nested_complex_entity_instance_subtype*)>
            <!-- Used by: (array_literal attribute_instance bag_literal
constant_instances list_literal non_constant_instances set_literal
type_literal) -->

<!ELEMENT nested_complex_entity_instance_subtype (attribute_instance*,
nested_complex_entity_instance_subtype*)>
            <!-- Used by: (nested_complex_entity_instance
nested_complex_entity_instance_subtype) -->

<!ELEMENT non_constant_instances (nested_complex_entity_instance |
flat_complex_entity_instance)*>
            <!-- Used by: (schema_instance) -->

<!ELEMENT partial_entity_instance (attribute_instance*)>
            <!-- Used by: (flat_complex_entity_instance) -->

<!ELEMENT real_literal (#PCDATA)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT schema_instance (constant_instances, non_constant_instances)>
            <!-- Used by: (data) -->
```

```
<!ELEMENT set_literal (binary_literal* | integer_literal* |
logical_literal* | real_literal* | string_literal* | bag_literal* |
list_literal* | set_literal* | array_literal* | type_literal* |
(nested_complex_entity_instance | flat_complex_entity_instance |
entity_instance_ref)*)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT string_literal (#PCDATA)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT true EMPTY>
            <!-- Used by: (logical_literal) -->

<!ELEMENT type_literal (binary_literal | integer_literal | logical_literal
| real_literal | string_literal | bag_literal | list_literal | set_literal
| array_literal | type_literal | nested_complex_entity_instance |
flat_complex_entity_instance | entity_instance_ref | enumeration_ref)>
            <!-- Used by: (array_literal attribute_instance bag_literal
list_literal set_literal type_literal) -->

<!ELEMENT unknown EMPTY>
            <!-- Used by: (logical_literal) -->
```

Example (PDML) Early-bound DTD

```
<!-- Early-bound DTD hand-generated according to PDML algorithm with
modifications necessary to comply with late-bound architectural DTD.
However, the architectural attributes which are discussed in the document
are not shown here.

By: Robin La Fontaine
21 Oct 1999

-->

<!ELEMENT pets ((pet | person)*)>

<!ATTLIST pets
            id ID #REQUIRED
            version CDATA #FIXED " ... "
            declaration_type CDATA #FIXED "schema">

<!ELEMENT pet (pet.name, pet.owner, (dog | cat | chinchilla))>
<!ATTLIST pet
      id ID #REQUIRED>
<!ELEMENT pet-ref EMPTY>
<!ATTLIST pet-ref
      refid IDREF #REQUIRED>

<!ELEMENT person (person.name)>
<!ATTLIST person
      id ID #REQUIRED>
<!ELEMENT person-ref EMPTY>
<!ATTLIST person-ref
      refid IDREF #REQUIRED>

<!ELEMENT person.name (string)>
<!ELEMENT pet.name (string)>
<!ELEMENT pet.owner (person-ref)>


<!ELEMENT dog EMPTY>
<!ATTLIST dog
      id ID #REQUIRED>
<!ELEMENT dog-ref EMPTY>
<!ATTLIST dog-ref
      refid IDREF #REQUIRED>

<!ELEMENT cat EMPTY>
<!ATTLIST cat
      id ID #REQUIRED>
<!ELEMENT cat-ref EMPTY>
<!ATTLIST cat-ref
      refid IDREF #REQUIRED>

<!ELEMENT chinchilla EMPTY>
<!ATTLIST chinchilla
      id ID #REQUIRED>
<!ELEMENT chinchilla-ref EMPTY>
<!ATTLIST chinchilla-ref
      refid IDREF #REQUIRED>
```

```
<!ELEMENT string  (#PCDATA )>
<!ATTLIST string  width CDATA  #IMPLIED
                  fixed (fixed | not-fixed) "fixed" >
```