Instructions to write the ITM XML schemas

F. Imbeaux

28/03/2012

General instructions for the IMP persons

Declaring the top element of the CPO

1. CPOs are declared in separate xsd files, that usually have the same name as the CPO (e.g. magdiag.xsd for the magdiag CPO).

2. The CPO.xsd file must include utilities.xsd (definition of general ITM data types) : <xs:include schemaLocation="utilities.xsd"/>

3. Declare then first the top CPO element, e.g. <xs:element name="magdiag">. It must have in its annotation : i) its definition, ii) the string "Time-dependent CPO" if time –dependent and "CPO" if not time-dependent, iii) the key string "machine description" as appinfo if this CPO contains some machine description information. Example :

```
<xs:annotation>
<xs:documentation>Magnetic diagnostics. Time-dependent CPO</xs:documentation>
<xs:appinfo>machine description</xs:appinfo>
</xs:annotation>
```

4. Just below the top level, the CPO must include the following elements :

   a. Datainfo (Reference in utilities) : <xs:element ref="datainfo"/>

   b. Codeparam (Reference in utilities) : <xs:element ref="codeparam"/> if the CPO is expected to be produced by a code.

   c. Time (if time-dependent CPO), declared explicitly as :

```
<xs:element name="time" type="xs:float">
<xs:annotation>
<xs:documentation>Time [s]; Time-dependent; Scalar</xs:documentation>
</xs:annotation>
</xs:element>
```

Declaring individual signals

1. MDS+ signal names limited to 12 characters. Avoid using more than 12 characters in the schemas !

2. Use minor case only for the signal names. Underscores are allowed. Avoid all carriage return in the documentation (use ";" instead). Avoid using quotes in the documentation.

3. Do not define signals in the data structure for the size of arrays that can be obtained unambiguously by a "size" instruction in a program.

4. Define precisely all signals (including sub-structure branches) in the <xs:annotation> <xs:documentation>. In addition, at the same place :

   a. Specify units in brackets [], in the documentation of each signal. E.g. [m];

   b. Specify type and dimensions of the signal in brackets (), in the documentation of each signal. E.g. Vector (ndim1) or Matrix (ndim1,ndim2).

   c. If the signal is time-dependent, add in the <xs:annotation> <xs:documentation> the string "Time-dependent".

   d. If the signal is machine description, add in the annotation <xs:appinfo>machine description</xs:appinfo>

   e. If the signal can be potentially imported by Exp2ITM, though is not a generic measurement signals (exp0D, exp1D, exp2D), add in the annotation <xs:appinfo>experimental</xs:appinfo>

5. Declare the type of each signal in the "type" attribute. NB : the time-dependence property is independent of the type declaration (see above). Declare the type as it is in a single time slice CPO. The following simple Types are allowed :

   a. xs:float : double precision real
   b. xs:integer : integer
   c. xs:string : string (NB : a string cannot be time-dependent)
   d. vecflt_type : vector (1D) of double precision real
   e. vecint_type : vector (1D) of integers
   f. vecstring_type : vector (1D) of strings (cannot be time-dependent)
   g. matflt_type : matrix (2D) of double precision real
   h. matint_type : matrix (2D) of integers
   i. array3dflt_type : array (3D) of double precision real
   j. array3dint_type : array (3D) of integers
   k. array4dflt_type : array (4D) of double precision real
   l. array5dflt_type : array (5D) of double precision real
   m. array6dflt_type : array (6D) of double precision real

n. array7dflt_type : array (7D) of double precision real

Declaring sub-structures in the CPO

1. Use grouping of signals by topic in sub-structures when possible. It avoids long flat lists of variables that may be difficult to read and search.

2. S**ub-structures must be coded by defining a complex Type at the beginning of the xsd file (the complex Type could have a very long, unique name** while the name of the substructure of this Type could be simple). Example: one defines the complexType as:

```
<xs:complexType name="dist_nucl_reac">
<xs:sequence>
List here all elements of the substructure
</xs:sequence>
</xs:complexType>
```

   the complexType is then inserted as an element in the CPO as:

```
<xs:element name="nucl_reac" type="dist_nucl_reac"/>
```

3. Commonly used complex types can be found in utilities.xsd : set of positions (R, Z), (R, Z, phi, …), generic measurement signals (exp0D, exp1D, exp2D), …

4. A substructure can be made an array of substructure. For this, set the attribute MaxOccurs to "unbounded". **Very important: if any element in this substructure is Time-dependent, the array of substructure must be declared as Time-dependent**:

```
<xs:element name="distri_vec" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Vector over all distribution functions; Time-dependent. Structure
array(ndist_spec)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
List here all elements of the substructure
</xs:sequence>
</xs:complexType>
</xs:element">
```

Advanced features (for ISIP only)

Multiple CPO occurrences
The maximum number of CPO occurrences is declared at the level of Phase4TOP.xsd, using the standard schema attribute "maxOccurs".
It is then passed to CPODef.xml as the "maxoccur" attribute. This attribute is then read by CPODef2mdspreTree to generate the MDS+ tree.

Conflicts in F90 UAL
In the coding of the GET/PUT routines for Fortran, a GET/PUT routine is created for each complex structure (unavoidable since Fortran does not allow for direct type def declaration). Then all these routines are at the same level in the f90 files, i.e. we lose the memory of the arborescence in the naming of the GET/PUT subroutines. Therefore it must be avoided to create two complex structures that have the same name at different levels of the data trees, otherwise conflict !! Generic complex types are fine since their get/put routine is defined only once (from the utilities), but beware of isolated complex structures that may have the same name !! Therefore, it is recommended to declare sub-trees by creating generic complex types belonging to the CPO.xsd file with long unambiguous names (e.g. defining a complexType scenario_configuration in scenario.xsd allows to have a sub-tree configuration with conflict).

Conflicts in Java UAL
There cannot be a child with the same name as its direct parent in Java → avoid doing this in the schemas.

Conflicts in Python UAL
A given Complex Type cannot be used simultaneously for a simple structure and arrays of structure. If there is such need, the Complex Type must be duplicated and its 2 instances reserved respectively for i) simple structure and ii) array of structure usage.
In addition, "global" is a keyword in Python, so no signal or branch in the data structure should be named "global".

Topical groups of CPOs (ISE)
For the convenience of the User, in the Integrated Simulation Editor (ISE), the CPOs are grouped by topic. Each CPO has thus a "group" information (at the level of the schemas) that indicates to which group it belongs. It is documented in the schemas by adding to the appinfo string the following syntax :
<xs:appinfo>group:information.</xs:appinfo>
The end '.' is important for the detection of this string by the xsd2CPODef7 script.
This information is thus transferred to the CPODef.xml file, from which it is then read by the interface.
Existing groups are (just made for testing purposes, not really used up to now) :
information (information on the simulation)

Visualisation (representation) information
In particular for the VisIt application, we insert in the schemas information on the possible representations for individual signals or structures (relative path in case of structures)
A TERMINER 04/07/09

Units

Units are put in brackets [] by the user in the schema under <xs:annotation> <xs:documentation>. They are identified by xsd2CPODef7.xsl and propagated in the “unit” attribute by xsd2CPODef7.xsl.

Complex types used in multiple contexts

Some complex types defined in utilities can be used in multiple contexts (e.g. an rz1d sub-structure can be time-dependent or not), so that the properties of the leaves depends on the properties of the parents. Similarly, the unit of measurement signals (exp0D, exp1D, exp2D) is defined at the level of parents.

Complex mechanisms have been set in xsd2CPODef7.xsl to propagate the information correctly from the schemas to the leaves of CPODef.xml.

For the time-dependence : the <xs:appinfo>parent-dependent</xs:appinfo> information must be given at the level of the complexType. Then all leaves of the ComplexType will have the same status (time-dependent or not) as their parent.

Example:

```
<xs:complexType name="rz1D">
                <xs:annotation>
                        <xs:documentation>Structure for list of R,Z positions (1D)</xs:documentation>
                        <xs:appinfo>parent-dependent</xs:appinfo>
                        <xs:appinfo>representation var=scalar; meshtype=curve0; link1=r; link2=z;</xs:appinfo>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="r" type="vecflt_type">
                                <xs:annotation>
                                        <xs:documentation>Major radius [m]</xs:documentation>
                                </xs:annotation>
                        </xs:element>
                        <xs:element name="z" type="vecflt_type">
                                <xs:annotation>
                                        <xs:documentation>Altitude [m]</xs:documentation>
                                </xs:annotation>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
```

For units, see the self-explaining exp1D example:

```
<xs:complexType name="exp1D">
                <xs:annotation>
                        <xs:documentation>Structure for experimental 1D signal</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="value" type="vecflt_type">
                                <xs:annotation>
                                        <xs:documentation>Signal value; Time-dependent;
Vector</xs:documentation>
                                        <xs:appinfo>unit:as_parent.</xs:appinfo>
                                </xs:annotation>
                        </xs:element>
                        <xs:element name="abserror" type="vecflt_type">
                                <xs:annotation>
                                        <xs:documentation>Absolute error on signal; Time-dependent;
Vector</xs:documentation>
                                        <xs:appinfo>unit:as_parent.</xs:appinfo>
                                </xs:annotation>
                        </xs:element>
                        <xs:element name="relerror" type="vecflt_type">
                                <xs:annotation>
                                        <xs:documentation>Relative error on signal (normalised to signal value);
Time-dependent; Vector</xs:documentation>
                                        <xs:appinfo>unit:none.</xs:appinfo>
                                </xs:annotation>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
```

Exp2ITM

Exp2ITM is dynamically generated from a Mapping_Template.xml file that is generated from the schemas. It describes all mapping characteristics for a subset of the data structure containing only the signals to be imported by exp2ITM. The Mapping_Template file is also given to local experts to fill for describing the mapping between local and ITM data formats. Two kinds of signals are considered for importation by exp2ITM, and treated slightly differently :

1. Mesurement signals, of ComplexType exp0D, exp1D, exp2D. These are common substructures described in utilities.xsd with value, relerror and abserror.
2. Individual signals not necessarily corresponding to measurements but which are considered useful to import (processed data such as fitted profiles, equilibrium data). These must be flagged at the schema level with the following property inserted in their annotation : <xs:annotation><xs:appinfo>experimental</xs:appinfo></xs:annotation>

The chain is the following (the numbers 1 and 2 refer to the two cases above) :

- Schemas → CPODef.xml (xsd2CPODef7.xsl)
    1. xsd2CPODef7 identifies directly (rule coded in it) the measurement signal types “exp0D”, “exp1D”, and “exp2D”. It propagates the information by setting an attribute “experimental” = “exp0D”, “exp1D”, “exp2D” in CPODef.xml for the leaves (value, relerror and abserror).
    2. For individual signals, the “experimental” property in the schemas is read by xsd2CPODef7 and depending whether the signal type is “xs:float”, “vect1dflt_type” or “matflt_type”, the attribute “experimental” = “expSignal/exp0D”, “expSignal/exp1D”, “expSignal/exp2D” is set in CPODef.xml. This is to differentiate from the well defined structure (value, relerror, abserror) of case 1 (useful at the Exp2ITM generation level).
- CPODef.xml → MachineMapping_template.xml (CPODef2ExpMapping.xsl)

  For both cases, a subset of the datastructure containing only the leaves having an “experimental” attribute in CPODef.xml is created. The value of the “experimental” attribute in CPODef.xml is propagated as the value of the “type” attribute in MachineMapping_template.xml. Below each of these leaves, the template for mapping description is added (coded in CPODef2ExpMapping.xsl).