TCRS – A Methodology and Tool Set for Specifying Data Content

Jesse Campos, Greg Hull
Science Applications International Corporation (SAIC)
12901 Science Drive, Orlando, Florida 32826
greg.a.hull@saic.com
jesse.j.campos@saic.com

Farid Mamaghani
SEDRIS Organization
19223 SE 45th Ct
Issaquah, Washington 98027
farid@sedris.org

Keywords:
Environmental Data, Interoperability, TCRS, SEDRIS, XML

ABSTRACT: *Specifying the requirements for environmental data at the input and output of systems or applications, and the ability to automatically evaluate and validate the data based on such requirements, is a key ingredient to successful data interoperability. This paper will describe an innovative and ongoing development, the Transmittal Content Requirements Specification (TCRS), which addresses this challenge. TCRS leverages the SEDRIS technologies of DRM, EDCS, and SRM to provide a formal methodology for the expression, and the subsequent evaluation, of environmental data requirements. TCRS is composed of several key technical components, including a process and methodology for articulating and capturing the requirements; a complete syntax for expressing the requirements; an XML encoding of the syntax; and automated tools, such as a parser and evaluator, for validating transmittals that claim conformance to a given set of requirements. This paper will provide an overview of the various technical components, and will highlight examples in the application of TCRS to real world data sets.*

1. Introduction

SEDRIS technologies provide a robust mechanism for the representation of environmental data across all domains and applications. In order to provide these broad capabilities, five technology components have been developed. These five technology components are:

- The SEDRIS Data Representation Model (DRM) that supports the full representation of any environmental data.
- The Spatial Reference Model (SRM) that supports an extensive definition of Spatial Reference Frames (SRF).
- The Environmental Data Coding Specification (EDCS) that provides complete classification and attribution for environmental data.
- An interface specification, the SEDRIS API, which allows the user to develop applications to consume and produce environmental data.
- A binary file format, the SEDRIS Transmittal Format (STF), specifically designed for the efficient storage and access of SEDRIS data.

These SEDRIS technologies provide for a complete and unambiguous data representation and interchange capability, but do not directly implement the verification of data content or the notion of "fit for use". The SEDRIS DRM and EDCS provide the framework to conduct content and applicability analysis. But in order to specify environmental data requirements, a complimentary technology, the Transmittal Content Requirement Specification (TCRS), has been developed to augment the DRM and EDCS in order to capture such requirements. The capabilities presented in this paper describe the methodology for the expression of environmental data requirements, an XML encoding for the specification of such requirements, and a toolset for data verification.

This paper begins by describing the role that TCRS plays in the interchange of environmental data. This will include inherent problems with current specification techniques and examples of initial TCRS solutions. The paper then introduces the current TCRS methodology and tools for improving interchange. The XML encoding of TCRS and examples using the syntax will be highlighted. The paper will include several examples of TCRS requirements. The future development and enhancement plans will conclude the paper.

2. Need for TCRS

2.1 Data requirements

A TCRS document is a specification of the environmental data "requirements" expressed using the SEDRIS DRM, EDCS, and SRM terminology. A requirement may be about the semantics of the data, such as which environmental objects must or must not be present. Or a requirement may be about the structure or form of representation that the data may take, such as whether data is in gridded or point sample form. Specifying data requirements allows the evaluation of data based on "fit for use" criteria for a particular application or set of applications. The specification provides an "Acceptance Test" for data, much like acceptance tests for software or hardware products. However, a common methodology and a common terminology are required to accomplish this.

2.2 Specification solutions

Traditionally two mechanisms have been used to capture requirements. The first is simply to express the requirements in English, usually through Word documents. The second is to develop highly unique, tailored verification software. The Close Combat Tactical Trainer (CCTT) Correlated DataBase (CDB) TCRS is an example of expressing requirements through a Word document [1]. Environmental data model compliancy checkers are an example of the TCRS method of writing tailored software.

2.3 Short-comings of specification solutions

The problem with requirements embedded in unique software is that they are single-point solutions. Requirements expressed in Word documents can be problematic since they are often unclear and imprecise due to human interpretation, and are not machine parsable. For example, the following could be stated as a requirement: "All trees must have a height and a stem diameter". In order to determine if a data set meets this requirement, it is necessary to specify how a tree is represented, and whether "tree" refers to a single tree, a bush, a treeline, etc. These types of ambiguities can be avoided through the use of a formal syntax that utilizes terms from the DRM and the EDCS.

3. Using TCRS

3.1 TCRS Concepts

By relying on the SEDRIS DRM and the EDCS, the TCRS concepts provide the capability to specify the broadest possible range of environmental data requirements. A TCRS *requirement*, in this sense is composed of two parts: a *domain* and a *condition*. The domain is the set of all objects for which a *requirement* is evaluated. The *condition* is the set of criteria applied to each object in the *domain*. In the previous example, the domain consists of all (software) objects that represent trees. The condition is that the (software) object representing the tree also be attributed with a height as well as a stem diameter. Note that what is missing here is information about how the trees may be represented and how they may be attributed.

Consider two other requirements, again expressed in English, but using DRM terms:

"Buildings must be represented by <Point_Feature>s with <Property_Value> components for height and width."

"Buildings represented as <Point_Feature>s must be given <Property_Value> components for height and width."

At a glance these two requirements may seem the same, but the difference lies in the distinct domains to which their respective conditions are applied. This is what is often not clear in English requirements and is the reason that TCRS syntax must have an explicit distinction between its domain and condition. The domain of the first example is all DRM objects which represent Buildings, while the domain of the second is only those <Point_Feature> objects representing buildings. Therefore a transmittal that contains buildings represented as <Polygon>s with a height and width will fail the first requirement. Note that the example is still missing an important piece; namely which EDCS entries will be used to identify the concepts of building, width, and height. The TCRS syntax must also use the syntax of EDCS, and, similarly, of the SRM. This requires the TCRS syntax to be flexible and expressive, allowing a user to define many different object sets for domains and conditions. This is done by introducing Object Matching Expressions (OME).

3.2 Object Matching Expressions

An OME is an expression composed of a set of individual criteria, where each criteria is a pass/fail test for a given object. An OME is evaluated against a SEDRIS object and *matches* the object if the evaluation is determined to be true. In the process of matching one object, other selected objects (components for example) may be involved. This set of objects *satisfies* the OME if the evaluation is determined to be true. There are 10 different criteria that can be used to build OMEs and are defined below:

- The *component* criteria evaluates to TRUE if the matching object has a satisfying component object.
- The *associate* criteria evaluates to TRUE if the matching object has a satisfying associate object.
- The *aggregate* criteria evaluates to TRUE if the matching object has a satisfying aggregate object.
- The *object* criteria without additional restrictions evaluates to TRUE since the matching object is itself. The matching object is the satisfying object.
- The *descendent* criteria evaluates to TRUE if the matching object has a satisfying object which is a descendent (recursive component relationships).
- The *ancestor* criteria evaluates to TRUE if the matching object has a satisfying object which is an ancestor (recursive aggregate relationships).
- The *field* criteria evaluates to TRUE if the matching object has a field value that is in the specified allowed set of values. The matching object is the satisfying object
- The *referenced object* criteria evaluates to TRUE if the matching object references another object that has been matched by other criteria in the OME. The referenced object is the satisfying object.
- The *indexed object* criteria evaluates to TRUE if the matching object indexes an object, through one of the DRM indexing mechanisms. The indexed object is the satisfying object.
- The *function* criteria evaluates to TRUE if the matching object passes the user implemented software tests. The implemented user function determines the satisfying object.

Each of the above criteria may add extra restrictions on satisfying objects. For example, all criteria may require that satisfying objects be of a given DRM class or that the number of satisfying objects be within a given range. Criteria can also be combined through the use of logical '*and*', '*or*', and '*not*' operators.

Further expressive power is achieved by compounding two OMEs together such that one OME applies to the set of objects that satisfy another OME. For example, one OME can match all objects of type LINEAR_FEATURE that have a component object of type CLASSIFICATION_DATA. A compound OME can be created by applying a second OME to the CLASSIFICATION_DATA object. This second OME could match CLASSIFICATION_DATA objects which have a 'tag' field of TREELINE. This compound OME would then match all LINEAR_FEATURE objects with a CLASSIFICATION_DATA component of TREELINE.

OMEs can be compounded in this way to any depth required. Doing so will often yield a larger set of satisfying objects, and in certain cases some of these satisfying objects will be further referenced or have additional criteria applied to them. For this reason, the satisfying objects can be made available to other OMEs.

3.3 Requirements and OMEs

The domain of a requirement is expressed simply as an OME. The domain is the set of all objects in a transmittal which match this OME. The condition may also use an OME to define the rules that objects from the domain must pass in order to satisfy the requirement. In addition, the condition may also state that the domain of the requirement contain a certain number of objects, although this is not considered part of an OME.

Within a TCRS there can be two kinds of OMEs, global OMEs and local OMEs. A global OME is one that can be referenced by the domain or condition of any requirement in the TCRS document, or by any other OME. A local OME is only used within the domain or condition which defines it. The domain or condition may specify either a global or a local OME. A global OME must be given a name for referencing by other OMEs, while a local OME must be anonymous.

3.4 TCRS verification

An application, called the XTCRS Checker, has been developed to evaluate the requirements expressed in a XML-encoded TCRS document. The primary function of this application is to evaluates a given data set against the requirements in a TCRS document. The application also serves as a syntax checker for verifying the requirements have been expressed according to the TCRS syntax.

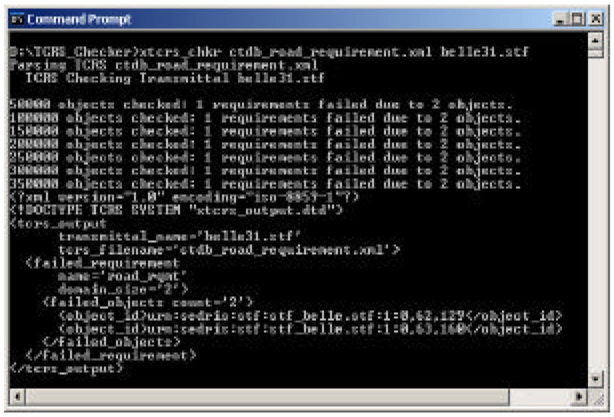
3.4.1 XTCRS Checker capabilities

The XTCRS Checker is a command line application that requires a TCRS document conforming to the XTCRS DTD and a SEDRIS transmittal in STF as inputs. Given these two inputs, the application will run and verify the STF conforms to the specified TCRS.

The XTCRS checker has additional parameters to control output and execution. An object identification string can be provided with the –s option. This will cause the XTCRS Checker to only check the object tree starting at the object specified by the object identification string. The TCRS document can be parsed and validated without actually checking a transmittal. This is done by passing the -v option to the application. The -o option allows the user to specify an output file that will be written out containing the results. The results are stored in a document that conforms to the XTCRS Output DTD. This DTD defines a structured output of the application and is provided with the XTCRS Checker. The structured output files allow for automatic parsing by other tools, such as Focus, in order to make transmittal modifications or automatic corrections. [2]

3.4.2 XTCRS Checker Output

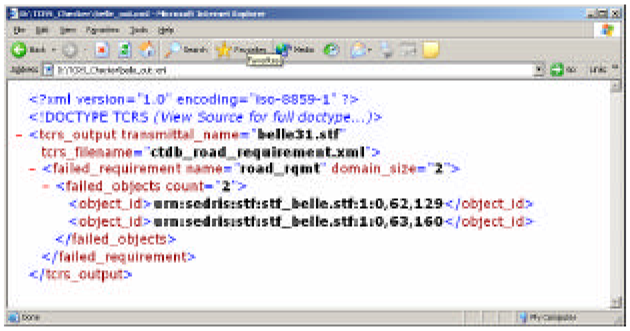
The following shows the execution of the application using the TCRS document *ctdb_road_requirement.xml* on the transmittal *belle31.stf* and no additional options. The results show that there were two objects which met the domain, and that both of them failed the requirement. The name of the requirement and the object id of the objects that failed it, are provided in the output:



```
D:\TCRS_Checker>xtcrs_chkr ctdb_road_requirement.xml belle31.stf
Parsing TCRS ctdb_road_requirement.xml
  TCRS Checking Transmittal belle31.stf

50000 objects checked: 1 requirements failed due to 2 objects.
100000 objects checked: 1 requirements failed due to 2 objects.
150000 objects checked: 1 requirements failed due to 2 objects.
200000 objects checked: 1 requirements failed due to 2 objects.
250000 objects checked: 1 requirements failed due to 2 objects.
300000 objects checked: 1 requirements failed due to 2 objects.
350000 objects checked: 1 requirements failed due to 2 objects.
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE TCRS SYSTEM "xtcrs_output.dtd">
<tcrs_output
      transmittal_name='belle31.stf'
      tcrs_filename='ctdb_road_requirement.xml'>
  <failed_requirement
      name='road_rqmt'
      domain_size='2'>
    <failed_objects count='2'>
      <object_id>urn:sedris:stf:stf_belle.stf:1:0,62,129</object_id>
      <object_id>urn:sedris:stf:stf_belle.stf:1:0,63,160</object_id>
    </failed_objects>
  </failed_requirement>
</tcrs_output>
```

Running the application with the same inputs, but adding an output file provides the results in the file *belle_out.xml*. The following displays the contents of *belle_out.xml*:



```
  <?xml version="1.0" encoding="iso-8859-1" ?>
  <!DOCTYPE TCRS (View Source for full doctype...)>
- <tcrs_output transmittal_name="belle31.stf"
   tcrs_filename="ctdb_road_requirement.xml">
 - <failed_requirement name="road_rqmt" domain_size="2">
  - <failed_objects count="2">
      <object_id>urn:sedris:stf:stf_belle.stf:1:0,62,129</object_id>
      <object_id>urn:sedris:stf:stf_belle.stf:1:0,63,160</object_id>
    </failed_objects>
   </failed_requirement>
  </tcrs_output>
```

4. XML Encoding of TCRS

This section describes the XML elements, the possible child – parent relationship of the elements, attributes, and valid attribute values used to encode TCRS documents.

4.1 Top Level XML elements

The root element of a TCRS document encoded in XML is the <TCRS> element. Only two types of child elements for <TCRS> are permitted, <requirement> and <expression>. The <requirement> element is used to define a requirement. The <expression> element is used to encode an OME. When an <expression> element is a child of a <TCRS> element, it defines a global OME. Any <expression> elements child of a <TCRS> element must have a '*name*' attribute which is unique. The <requirement> element has one mandatory attribute, '*name*,' which also must be unique, and two mandatory child elements, <domain> and <condition>. When a domain or condition is specified with a global OME, the <domain> or <condition> will have an '*expression_ref*' attribute with a value that is the '*name*' attribute of the <TCRS> level <expression>. For local domain or condition OMEs, the <domain> or <condition> element will have a child <expression> element. In addition, the <condition> element has two optional attributes, '*min_domain_size*' and '*max_domain_size*' to define restrictions on the number of objects in the domain.

The following illustrates the syntax for a TCRS with a single requirement and global OME. The requirement has a domain with a global OME referenced by *domain_expr* and a condition with a local OME:

```
<TCRS>
   <expression name="domain_expr">
      <!-- OME Goes Here -->
      <!-- OME Goes Here -->
   </expression>

   <requirement name="example_rqmt">
      <domain expression_ref="domain_expr"/>
      <condition min_domain_size="1">
         <expression>
            <!-- OME Goes Here -->
            <!-- OME Goes Here -->
         </expression
      </condition>
   </requirement>
</TCRS>
```

4.2 OME XML Syntax

In order to build an OME, the <expression> element must have one or more children which will specify either a logical operator element or one of the OME criteria elements. These elements are described in the following sections. With the exception of the <field> element, all of the logical operator elements and criteria elements are allowed to contain any of these same elements as child elements. Allowing this recursive use of elements allows the OME compounding described in section 3.2. Thus, the compounding of OMEs is captured through the encoding of the parent—child relationships of the elements described in this section.

4.2.1 Logical operator syntax

The logical operators elements are <and>, <or> and <not>. The <and> element evaluates to TRUE if all of its child elements evaluate to TRUE. The <or> element evaluates to TRUE if at least one of its child elements evaluates to TRUE. The <not> element evaluates to TRUE if any of its child elements evaluates to FALSE.

Criteria elements can contain any number of child criteria elements as well as logical operator elements. When a criteria element contains other elements, the criteria element evaluates to TRUE if all of its child elements evaluate to TRUE.

4.2.2 Component criteria syntax

The <component> element is used to represent the component criteria. Further restrictions may be placed on the satisfying component objects using attributes of the elements. Many of these attributes will also apply to other criteria elements as discussed in subsequent sections.

The '*class*' attribute is used to express the DRM class of a satisfying object from a specific set DRM classes. The value of the attribute is the name of the DRM class with underscores in place of spaces.

```
<component class="Property_Value" />
```

Likewise the following OME will match an object that has a Property Value component which has a Property Characteristic component.

```
<component class="Property_Value">
    <component class="Property_Characteristic"/>
</component>
```

The '*min_count*' attribute requires that a minimum number of satisfying objects be present in order for the criteria to evaluate to true. Similarly, the '*max_count*' attribute requires a maximum number of satisfying objects. The following example matches objects that have exactly 3 Location components:

```
<component     class="Location"     min_count="3"
max_count="3" />
```

The '*optional*' attribute is used to express that the criteria may evaluate TRUE even if there is no component object. However, if a component is present then all the child criteria elements must evaluate to TRUE. The following will match objects which either have no Classification Data components, or if they do, then the Classification Data component has a component Property Value:

```
<component class="Classification_Data"
 optional="TRUE">
    <component class="Property_Value"/>
</component>
```

The '*all*' attribute is used to express that all of the satisfying objects pass the criteria child elements. For example, the following will match objects only if all of its components have a Property Characteristic component:

```
<component all="TRUE">
    <component class="Property_Characteristic"/>
</component>
```

The '*order*' attribute applies only to ordered DRM relationships such as Vertex components of Polygons. Since the DRM specifies that ordered relationships are based on a DRM class, the '*class*' attribute is required when the '*order*' attribute is given. In the following example only the 3rd Location component of an object will satisfy the criteria:

```
<component class="Location" order="3"/>
```

The '*label*' attribute, unlike previous attributes, does not restrict the set of satisfying objects. This attribute affects the evaluation process of the criteria and the OME. When this attribute is present the satisfying object is made available for reference from other criteria within the OME. The referenced object criteria is used to accomplish this referencing as described in 4.2.10.

The '*link_obj_label*' behaves similarly to 'label'. When relationships have a link object, this attribute is used to label and reference the link object.

4.2.3 Associate criteria syntax

The <associate> element is used to represent the associate criteria. The '*class*', '*order*', '*all*', '*optional*', '*min_count*', '*max_count*', '*label*' and '*link_obj_label*' attributes are also defined for the <associate> element. The following example matches an object with a Feature_Model associate that has a Union_of_Features component:

```
<associate class="Feature_Model" >
    <component class="Union_of_Features" />
</associate>
```

4.2.4 Aggregate criteria syntax

The <aggregate> element is used to represent the aggregate criteria. The '*class*', '*min_count*', '*max_count*', '*optional*', '*all*', '*label*' and '*link_obj_label*' attributes are defined for <aggregate>. For example, to match an object which is shared by 2 or more Polygons, the following would be specified:

```
<aggregate class=“Polygon” min_count=“2”/>
```

4.2.5 Object criteria syntax

The <object> element is used to represent the object criteria. The ‘*class*’, ‘*optional*’, and ‘*label*’ attributes are defined for <object>. The most common use case for the <object> element is as the first child element of an <expression> element. This helps to clarify which object is being matched by an <expression>. The <object> element is commonly used with the ‘*class*’ attribute to match an object of a given class. The following OME will match all objects containing a Classification_Data component:

```
<expression>
    <object>
        <component class="Classification_Data"/>
    </object>
</expression>
```

Another use case for the object criteria is provided with the ‘*expression_ref*’ attribute. The value of this attribute is the name of a global OME. An object matches the object criteria if it matches the referenced OME.

4.2.6 Descendent criteria syntax

The <descendent> element is used to represent the descendent criteria. The ‘*class*’, ‘*min_count*’, ‘*max_count*’, ‘*optional*’, ‘*all*’, and ‘*label*’ attributes are defined for <descendent>.

The ‘*generations*’ attribute is used to limit the search to the given ‘depth’ (number of recursions or levels). The value is a positive integer. For example, to require an object to have a Polygon descendent no more than 3 generations below it, the following syntax would be used:

```
<descendent class=“Polygon” generations=“3”/>
```

4.2.7 Ancestor criteria syntax

The <ancestor> element is used to represent the ancestor criteria. The ‘*class*’, ‘*min_count*’, ‘*max_count*’, ‘*optional*’, ‘*label*’, ‘*all*,’ and ‘*generations*’ attributes are defined for <ancestor>. The following example requires an object to have a Model ancestor no more than 3 levels above:

```
<ancestor class=“Model” generations=“3”/>
```

4.2.8 Field criteria syntax

The <field> element is used to represent the field criteria. The <field> element has mandatory attributes of ‘*name*,’ which is the name of the objects field, and ‘*values*’, which gives the set of allowed values for the field. The value of the ‘*name*’ attribute is a field in the DRM fields structure for the object. If the field name doesn’t exist for the DRM class of the object being evaluated then the object cannot satisfy the criteria. Fields within C structures or unions in the DRM are given with the ‘dot’ notation, similar to C, and must always be specified to the primitive type. For example, a Property_Value object can have the field “meaning.code.attribute” which is of primitive type EDCS_Attribute_Code, but it cannot specify “meaning” since “meaning” is a structured type. The format of the ‘*values*’ attribute may take several forms depending on the field data type and the allowed field values.

Enumerated types can match a single enumerant or a list of white-space delimited enumerants. The enumerants can be specified either as the numeric value of the enumerant, or as the name of the enumerant without the common prefix. For example, the following field criteria for the field name “*ordering_reason*”, which is of the DRM type SE_Ordering_Reason, will match the enumerant SE_ORDRNG_REASON_FIXED_LISTED.

```
<field  name=“ordering_reason”
 values=“FIXED_LISTED” />
```

For fields of numeric types, a match can be made to a single value, a list of values, or a range of values. A range of values may be specified with inclusive bounds using parenthesis or exclusive bounds using brackets. Range values may be unbounded in one direction by omitting one of the bounds values. The following example will match the given integer field, if it is either –1 or 9999:

```
<field name=“value.u.integer_value”
 values=“-1 9999” />
```

The following example will match the given integer field, if it is greater than or equal to 100 and less than 1000:

```
<field name=“value.u.integer_value”
 values=“[100, 1000)” />
```

The next example matches a floating point field if the value is greater than 1.0:

```
<field name=“value.u.float_value”
 values=“1.0, )” />
```

EDCS data types are specified similar to enumerants, using the EDCS label without the prefix. The EDCS Attribute is required for evaluating EDCS enumerants. The first option is to specify both the attribute and the enumerant with a separating colon:

```
<field name="value.u.ee_code"
 values="BUILDING_FUNCTION:HOUSE" />
```

The second option is to provide a field criteria requiring an enumerated EDCS Attribute at the same level as the EDCS Enumerant field criteria. In this situation, the EDCS Attribute can be left out. The following example demonstrates this case:

```
<field name="meaning.code.attribute"
 values="BUILDING_FUNCTION" />

<field name="value.u.ee_code"
 values="HOUSE" />
```

Fields of type SE_String are matched as one complete string value. The field only matches if the string value of the field is identical to the values attribute.

4.2.9 Referenced object criteria syntax

The <referenced_object> element is used to represent the referenced object criteria. The '*class*' and '*optional*' attributes are defined for the <referenced_object> element. The mandatory '*object_ref*' attribute is used to specify the label of the referenced object. The value of '*object_ref*' must match the '*label*' or '*link_obj_label*' attribute of another element under the <expression> element. A satisfying object of another criteria in the OME, which has the assigned referenced label, will also satisfy the referenced object criteria.

The following example illustrates applying criteria to a link object using the <referenced_object> element. The link object between the State_Related_Features object and its component Feature_Hierarchy, is labeled using the value *state_data_obj.* The link object can then be refereneced with the object_ref attribute of the <referenced_object> element.

```
<object class="State_Related_Features">
    <field  name="state_tag"
     values="EXISTENCE_STATUS" />
    <component class="Feature_Hierarchy"
     link_obj_label="state_data_obj"/>
    <referenced_object class="State_Data"
     object_ref="state_data_obj">
        <field name="state_data.value_type"
         values="ENUMERANT_CODE" />
        <field  name="state_data.u.ee_code"
         values="DAMAGED" />
    </referenced_object>
</object>
```

4.2.10 Indexed object criteria syntax

In the DRM there are several instances where objects are referenced by an index value. The most common example is Attribute_Set_Index where the DRM *index* field value is used to index an Attribute_Set object from a base Attribute_Set_Library object. Other examples exist for Colour_Index, Model_Instance_Template_Index and several different classes indexed from Data_Table cells. In all cases, the DRM defines these component relationships to be ordered. This requires the '*class*' attribute to be used in the syntax when encoding the TCRS.

The indexed object criteria is represented with the <indexed_object> element. This element is used when a criteria is to be applied to an object being indexed through the DRM indexing mechanism. Three attributes are used to give the necessary information, *field_index*, *dt_cell_index*, and *base_object_ref.* The '*field_index*' attribute gives the name of the (matching object's) field that is used to retrieve the actual index value. The exception to this is when a component of a Data_Table or Data_Table_Library is being indexed, in which case the index value is stored in a Data_Table cell. The '*dt_cell_index*' attribute is a list of comma delimited positive integers which define the 'coordinate' of the cell based on the Data_Table's Axis components. The number of integers must match the number of Axis components of the Data_Table. The base object from which the indexing is computed must be specified with the '*base_object_ref*' attribute which gives the label of a referenced object within an OME.

The following example will match an Attribute_Set_Index object which indexes an Attribute_Set with a Property_Value component of WIDTH: (The reader is referred to the SEDRIS DRM documentation for information on the DRM Attribute Set indexing mechanism.)

```
<object class="Attribute_Set_Index">
   <associate class="Attribute_Set_Table_Group">
      <component order="1"
       class="Attribute_Set_Table"
       label="attr_set_index_base_obj"/>
   </associate>

  <indexed_object
   base_object_ref="attr_set_index_base_obj"
   field_index='index' class="Attribute_Set">
      <component class="Property_Value">
          <field name="meaning.code_type"
           values="ATTRIBUTE"/>
          <field name="meaning.code.attribute"
           values="WIDTH"/>
      </component>
  </indexed_object>
</object>
```

4.2.11 Function criteria syntax

The <function> element is used to represent the function criteria. It is provided to allow users to create their own functions for evaluating user criteria. There are 3 mandatory attributes: '*name*', which is the function name to be called, '*args*', which specifies the arguments to be passed to the function, and '*library*' which is where the function resides All functions must have the following signature:

```
SE_Boolean function ( SE_Object obj_to_evaluate,
                      Char* args,
                      char *err_string );
```

A user defined function takes an object and determines whether it passes the user criteria or not. If the object failed the user criteria, the function returns an error string.

The following example illustrates how a user would specify calling a function by the name *polygon_is_vertical* and found in the library *my_tcrs_functions_lib* in a TCRS document:

```
<function name="polygon_is_vertical"
 args="tolerance"
 library="my_tcrs_functions_lib"/>
```

4.3 Examples

The following three examples illustrate how data requirements can be expressed using the TCRS capabilities.

4.3.1 Limiting the number of polygon objects

The following requirement specifies a range of from 100 to 1 million polygons objects to be present within a data set.

```
<requirement name="example_1">
    <domain>
        <expression>
            <object class="Polygon" />
        </expression>
    </domain>
    <condition
        min_domain_size="100"
        max_domain_size="1000000" />
</requirement>
```

4.3.2 Requiring specific environmental entities

The following requirement specifies at least one race track to be present and represented as an Areal Feature.

```
<requirement name="example_2">
  <domain>
    <expression>
      <object class="Areal_Feature">
        <component class="Classification_Data">
          <field name="tag"
           values="RACE_TRACK"/>
        </component>
      </object>
    </expression>
  </domain>
  <condition min_domain_size="1" />
</requirement>
```

4.3.3 Specifying application-dependent equivalence

The following requirement specifies that environmental objects classified as either a lighthouse or as a building functioning as a lighthouse must be represented as a Point_Feature and must have a height greater than 100 feet.

```
<requirement name="example_3">
  <domain>
    <expression>
      <object>
        <or>
          <component class="Classification_Data">
            <field name="tag"
             values="LIGHTHOUSE"/>
          </component>
          <component class="Classification_Data">
            <field name="tag"
             values="BUILDING" />
            <component class="Property_Value">
              <field
               name="meaning.code.attribute"
               values="BUILDING_FUNCTION" />
              <field name="value.u.ee_code"
               values="LIGHTHOUSE" />
            </component>
          </component>
        </or>
      </object>
    </expression>
  </domain>

  <condition>
    <expression>
      <object class="Point_Feature">
        <component class="Property_Value">
         <field name="meaning.code.attribute"
          values="HEIGHT_ABOVE_SURFACE_LEVEL" />
          <field name="value_unit"
           values="FOOT" />
         <field name="value.u.float_value"
          values="(100,)" />
        </component>
      </object>
    </expression>
  </condition>
</requirement>
```

5. Future TCRS evolution and application

Expressing complex environmental data requirements through the use of XML tags can be tedious and error prone. The current TCRS technology also forces the users of TCRS to learn a new syntax, which may distract from their primary objective, the expression of their environmental data requirements. This may further deter users from using the technology to articulate their requirements in a concise and unambiguous manner.

A graphical user interface may be a more attractive solution for developers or managers who wish to express their complex requirements through a simpler, more intuitive, and more efficient method. The next step in the development of the TCRS applications is to provide such a user interface. Through such an interface, users will only be required to know the terminology that relates to the environmental data (namely, the SEDRIS DRM, EDCS, and SRM). The interaction to build a TCRS document would be managed through context-sensitive user interfaces that can provide for on-the-fly error checking as a user selects and combines DRM classes or EDCS entries. In this manner, incorrect combinations of DRM classes can be avoided, and default settings for legitimate class combinations can be utilized. In addition, range values, default values, and recurring patterns of class combinations can be stored as specific user preferences and recalled when needed.

The user can employ drag and drop sequences, and create the necessary classes and their desired conditions, without having to learn the underlying syntax. More intuitive and natural language expressions and menu choices can be used to allow the user to express the desired conditions and requirements. And upon completion such a tool will also be able to process and validate a data set against the requirements that the user has produced.

Once TCRS documents can be generated more rapidly, users can combine smaller TCRS documents to accomplish larger and more complicated tasks. For example, applications (such as database converters [3]) can invoke automated or semi-automated data transformations, if an incoming data set matches all the necessary criteria for an input TCRS, and the necessary data is present to be transformed according to an output TCRS. We envision these capabilities, tools, and other innovations and improvements will go a long way toward providing a better platform for environmental data interoperability. We also envision the techniques used for validation and evaluation of environmental data can be applied to other fields and the basic concepts of TCRS can be extended and used in other areas.

6. Conclusion

This paper has discussed the TCRS methodology and XML syntax, and has highlighted the role of the TCRS in expressing environmental data requirements. It has illustrated the use of the TCRS through examples using the concept and methodology. A new application that uses machine parsable syntax to validate SEDRIS data has been introduced, and elements of a syntax using an XML encoding have been discussed in detail. The syntax of the XML encoding was described in detail with examples given at both the atomic element levels and as compound expressions. Finally, future development and the application of this technology were explored.

7. References

[1] J. Watkins, J. Campos: "Consuming SEDRIS Transmittals – A Pragmatic Approach", Fall SIW 2001

[2] M. A. .Pigora, D. Shen, J. Campos: "Innovating with SEDRIS Tools", Fall SIW 2001

[3] K. Wertman, J. Campos: "Using STFs for CTDB Production", Fall SIW 2002.

Author Biographies

GREG HULL is a Software Engineer with SAIC. He has been working the SEDRIS project for 4 years. Mr. Hull graduated from the University of Central Florida with B.S. degrees in Computer Science and Mathematics.

JESSE CAMPOS is a Software Engineer with SAIC. He has been working the SEDRIS project for almost five years. His primary expertise is in software development as it applies to databases. Mr. Campos graduated from the University of Central Florida with a B.S. in Electrical Engineering and a Masters in Business Administration.

FARID MAMAGHANI is project manager and technical director of the SEDRIS project. He has been involved in modeling, simulation, and systems engineering for more than 20 years.