Generating EML from a Relational Database Management System (RDBMS)

- Zhiqiang Yang and Don Henshaw (AND)

The Ecological Metadata Language (EML) is an open metadata specification and provides a standard syntax (XML) for LTER metadata (Jones 2003). Implementation of the LTER Metacat, a network-wide data set catalog, demands the creation of EML documents for all LTER data (Costa 2004). Future data integration activities, such as the development of the Trends database, will also rely on complete EML documents (Servilla et al. 2006). Development of EML documents is a data specific process depending on the existing format and structure of metadata, and there are different approaches that can be used to achieve this goal. One common means for storing metadata is in structured relational tables in a relational database management system (RDBMS).

In this short document, a generalized solution for generating EML from an RDBMS is presented and based on architecture originally developed at the CAP LTER (CES 2002). Storing metadata in an RDBMS has the advantage of being able to select and output metadata in various formats including EML. The approach presented here is two-step process of generating EML from the RDBMS: 1) generate a native XML document for metadata stored in RDBMS; 2) transform native XML into the EML document (Figure 1).

Figure 1. Data flow of metadata from the RDBMS to EML, e.g., from the local RDBMS table *databases* to the EML element: dataset.

Generating native XML

Metadata stored in a site RDBMS are typically organized as attributes in relational tables without any direct correspondence in structure or naming convention with matching EML elements. Generating EML directly from these stored metadata is challenging and cumbersome, and does require a mapping of local site attributes to the corresponding element in EML. Here, a native XML document is suggested as a connection point between the RDBMS and EML. For this discussion, native XML is an XML document which fully encapsulates all the metadata contained in the RDBMS for a specific dataset. Native XML does not have a predefined schema; the schema of native XML is dependent on the database schema for metadata, but creation and use of a configuration file allows specification of RDBMS tables for inclusion into the native XML.

Most RDBMS's maintain a data dictionary, which is a set of metadata that contains definitions and representations of data elements stored in the database. For example, SQL Server has two sources to view the data dictionary: the various system tables and the INFORMATION_SCHEMA views. Users can query these two sources directly or use system stored procedures to retrieve information about the database metadata. In the

following examples, the SQL Server system stored procedures, sp_columns, sp_pkeys, and sp_fkeys, are relied upon to programmatically access table column (attribute) information, and table primary and foreign keys.

Native XML generation is highly dependent on the data dictionary. To generate native XML an entry point is needed, which usually is a table representing objects corresponding to the EML dataset element. Typically, this starting point is a table with general dataset catalog attributes such as dataset code, title, abstract, and other dataset level metadata with relational links to other tables such as personnel, keywords, or data set entities. With a given starting point, all information stored in the RDBMS related to the given dataset can be retrieved using a standard SQL statement. This process uses extensively the information in the data dictionary, and native XML is generated by retrieving all the columns in the database related to the dataset. For example: suppose there are three tables as shown in Figure 2.

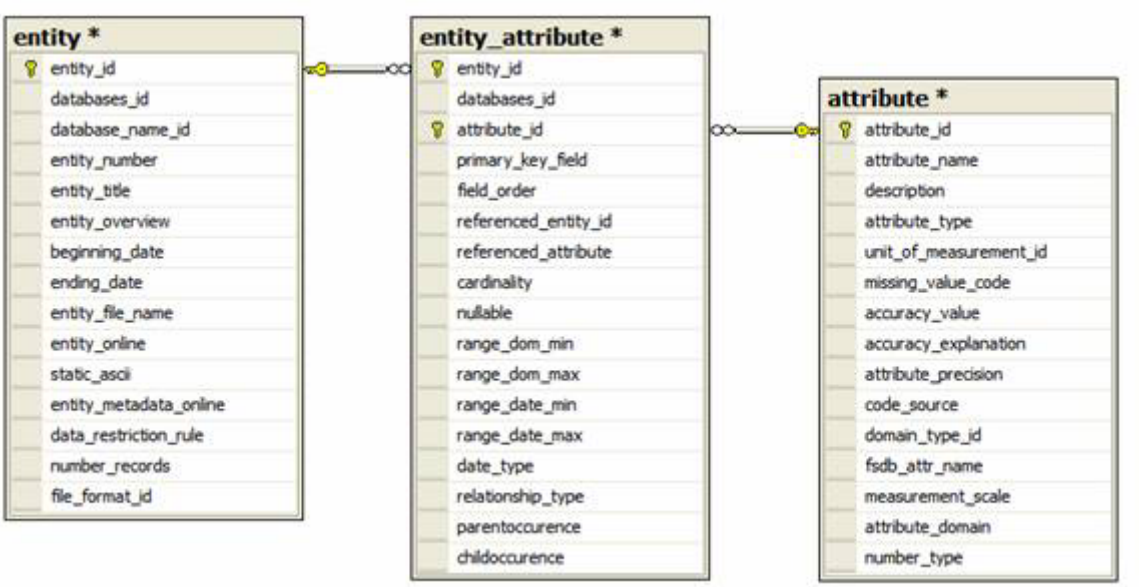

Figure 2. An example of a partial schema (from the Andrews Forest metadata schema). The *entity* table might be directly related to a parent dataset catalog table (not shown).

The native XML for an entity with entity_id = 135 would include the content from a single record in the entity table:

```
<entity entity_id="135" databases_id="122" database_name_id="1" entity_number="1"
  entity_title="Fire history and fire regimes" entity_overview="" beginning_date=""
  ending_date="" entity_file_name="DF00701" entity_online="Y" static_ascii="N"
  entity_metadata_online="Y" data_restriction_rule="" number_records="329" file_format_id="27"/>
```

Since the *entity* table is related to the *attribute* table through the *entity_attribute* pivot table, the native XML generation process will use the referential integrity information in the data dictionary to iterate through all of the records in both *entity_attribute* and *attribute* tables, resulting in a native XML similar to the example in Figure 3.

```
<entity entity_id="135" databases_id="122" database_name_id="1" entity_number="1"
 entity_title="Fire history and fire regimes" entity_overview="" beginning_date=""
 ending_date="" entity_file_name="DF00701" entity_online="Y" static_ascii="N"
 entity_metadata_online="Y" data_restriction_rule="" number_records="329" file_format_id="27"
 <entity_attribute_list>
  <entity_attribute entity_id="135" databases_id="122" attribute_id="1852"
   primary_key_field="N" field_order="5" referenced_entity_id="" referenced_attribute=""
   cardinality="" nullable="N" range_dom_min="" range_dom_max="" range_date_min=""
   range_date_max="" date_type="" relationship_type="" parentoccurence="" childoccurence=""
   <attribute attribute_id="1852" attribute_name="ASP" description="General aspect"
    attribute_type="char(2)" unit_of_measurement_id="" missing_value_code=""
    accuracy_value="" accuracy_explanation="" attribute_precision="" code_source=""
    domain_type_id="42" fsdb_attr_name="ASP" measurement_scale="nominal"
    attribute_domain="enum" number_type="">

    . . .
```

Figure 3. Native XML generated for entity_id=135 from the *entity* table, and using the *entity_attribute* table to link to the *attribute* table to list all of the attributes for that entity (entity to attribute is a many-to-many relationship, but only one attribute is shown)

This iterative process of metadata retrieval is continued until all the related information for a given entry point has been exhausted. That is, native XML from all related tables stemming from the original starting point is included.

However, problems frequently occur while programmatically iterating through the related tables, as a primary key may loop back to a previously included table. In the example, *entity_attributes* will refer back to the *entity* table creating a loop between tables *entity* and *entity_attributes*. Program code is written to prevent this loop from occurring by taking advantage of a configuration setting or file. The configuration file, a simple XML file, is established based on the data dictionary and helps guide the process of native XML generation. The configuration file is used to prevent the duplication of table information, describes the hierarchy of related tables, and lists the tables for which native XML will be generated. The configuration file for the example is shown in Figure 4.

```
<?xml version="1.0"?>
<entity>
    <entity_attribute>
        <attribute/>
    </entity_attribute>
</entity>
```

Figure 4. This example is a configuration file which specifies the tables for which native XML is to be generated.

A more complete configuration file example is included as Figure 5 and hierarchically represents many other tables associated with the attribute table from the Andrews metadata schema. This example includes unit of measurement, enumerated code

domains, place domains, taxonomic classification domains and attribute-specific methodology.

```
<entity>
  <file_format/>
  <entity_attribute>
    <attribute>
      <unit_of_measurement>
        <unit_type/>
      </unit_of_measurement>
      <attribute_enum_domain>
        <enum_domain/>
      </attribute_enum_domain>
      <attribute_place_keyword>
        <place_keyword/>
      </attribute_place_keyword>
      <attribute_taxonomic_classif>
        <taxonomic_classif/>
      </attribute_taxonomic_classif>
      <methodology_attribute>
        <methodology>
          <methodology_type/>
        </methodology>
      </methodology_attribute>
    </attribute>
  </entity_attribute>
  <database_rule/>
</entity>
```

Figure 5. A more extensive example of a configuration file based on the RDBMS data dictionary that includes the tables desired for inclusion in the native XML.

In summary, the configuration file is used to guide the process of creating native XML in conjunction with program code and referential integrity information provided by the data dictionary. The configuration file determines which tables are to be included in native XML generation allowing the exclusion of tables defined within the RDBMS schema.

Generating EML

XSL Transformation (XSLT) is used to convert native XML to a valid EML document. The key aspect of this process is to properly map the local metadata stored in the RDBMS to the corresponding EML <dataTable> elements. The mapping of a database schema to EML elements requires understanding of both the local RDBMS and the EML schema. The mapping process can become complex in the common situation where the metadata

database is not directly designed to accommodate EML metadata elements. And while the authors do not attempt to describe the XSLT language, we do provide a short example XSLT stylesheet to help illustrate this process of generating the EML <dataTable> element in Figure 6. For the example given above, the local *entity* table corresponds to <dataTable> in EML. The stylesheet illustrates direct mapping of a native XML element into EML, checks for an empty field before mapping entity_description, and uses named templates as functions or subroutines to map the native XML into EML coverage and attributeList elements.

```
<xsl:for-each select="$entities">
    <dataTable>
        <alternateIdentifier>
            <xsl:value-of select="@entity_file_name"/>
        </alternateIdentifier>
        <entityName>
            <xsl:value-of select="@entity_title"/>
        </entityName>
        <xsl:if test="not(@entity_description='')">
            <entityDescription>
                <xsl:value-of select="@entity_description"/>
            </entityDescription>
        </xsl:if>
        <xsl:if test="not(@beginning_date='')">
            <coverage>
                <xsl:call-template name="temporalCoverage">
                    <xsl:with-param name="list" select="."/>
                </xsl:call-template>
            </coverage>
        </xsl:if>
        <attributeList>
            <xsl:call-template name="attributeType">
                <xsl:with-param name="list" select="entity_attribute_list/entity_attribute"/>
            </xsl:call-template>
        </attributeList>
    </dataTable>
</xsl:for-each>
```

direct mapping from nativeXML

test for empty entity description

call named template *temporalCoverage*

call named template *attributeType*

Figure 6. A partial example XSLT stylesheet to generate the EML dataTable element from native XML.

Summary

While there are many challenges in storing metadata in a RDBMS that begin with the determination of a RDBMS schema for storing metadata and include loading and updating metadata into the RDBMS, significant benefits can be gained. Using the RDBMS to store metadata in structured and relational tables enables flexible presentation of metadata, including the generation of EML. The described approach takes advantage of features within the RDBMS for generating native XML including the use of the data dictionary for creating a configuration file and programmatically interpreting the underlying database schema. The configuration file is used to help guide

the process of populating native XML with attribute content from local RDBMS metadata tables. The XSLT stylesheets are used to complete the process of mapping the native XML into corresponding EML elements, but requires a fairly comprehensive understanding of both the local metadata tables and the EML schema. In addition to EML, other metadata document formats can also be easily generated using this described approach including PDF, HTML, Word, and other formats. Potentially, a customized script can be developed to map EML files back into the RDBMS, just the reverse process of the EML document generation described in this document, and we are exploring this possibility.

References

Center for Environmental Studies. 2002. "Xanthoria: A Distributed query system for XML encoded data", Arizona State University. Available on-line [http://ces.asu.edu/bdi/Subjects/Xanthoria/]

Costa, Duane. (2004). "EML Harvesting I: Metacat Harvester Overview and Management", LTER DataBits, Fall 2004 Issue, LTER Network Office. Available on-line [http://intranet.lternet.edu/archives/documents/Newsletters/DataBits/04fall/]

Jones, Matthew B. (2003). "A brief overview of Ecological Metadata Language", LTER DataBits, Spring 2003 Issue, LTER Network Office. Available on-line [http://intranet.lternet.edu/archives/documents/Newsletters/DataBits/03spring/]

Servilla, Mark, Brunt, James, San Gil, Inigo, Costa, Duane. (2006). "Pasta: A Network-level Architecture Design for Generating Synthetic Data Products in the LTER Network" LTER DataBits, Fall 2006 Issue, LTER Network Office. Available on-line [http://intranet.lternet.edu/archives/documents/Newsletters/DataBits/06fall/]

01001100 01010100 01000101 01010010

LTER DataBits

Information Management Newsletter of
The Long Term Ecological Research Network

01001100 01010100 01000101 01010010

DataBits: An electronic newsletter for Information Managers ----- [Spring 2007] Issue ([http://intranet.lternet.edu/archives/documents/Newsletters/DataBits/07spring/])

Featured in this issue:

Welcome to the Spring 2007 issue of Databits! Twenty-two authors or coauthors submitted articles for this issue, which is a testament to the committment of Information Managers to sharing information. The articles represent the diversity of interests within the LTER IM community and highlight a number of current topics. Most notably, there is a discussion about proposed changes in the organizational structure of the IM committee that would better integrate the GIS working group and the Technology Committee. Additionally in this issue, several articles focus on Ecological Metadata Language, describing recent developments and applications. Lastly, a number of LTER sites will be having their NSF midterm reviews in the coming months. The Baltimore Ecosystem Study was one of the first sites to be reviewed so Jonathan Walsh provided a list of some things to think about as sites prepare for these visits. We hope you find this issue of Databits informative and helpful and we thank all those who contributed articles. Enjoy!

DataBits continues as a semi-annual electronic publication of the Long Term Ecological Research Network. It is designed to provide a timely, online resource for research information managers and to incorporate rotating co-editorship. Availability is through web browsing as well as hardcopy output. LTER mail list IMplus will receive DataBits publication notification. Others may subscribe by sending email to majordomo@lternet.edu with two lines "subscribe databits" and "end" as the message body. To communicate suggestions, articles, and/or interest in co-editing, send email to databits-ed@lternet.edu.

----- Co-editors: John Campbell (HBR), Sabine Grabner (MCR)