Evaluation and Design of a Relational Database Implementation of the FGDC CSDGM Model Using Various Database Engines Along with XML Parser Design

Kit Na Goh, Idaho State University, GIS Training and Research Center, 921 S. 8th Ave., Stop 8104, Pocatello, Idaho 83209-8104

ABSTRACT

The current search methodology used for the GIS Training and Research Center's (GIS TReC) spatial library has limitations and problems which do not fully facilitate geospatial discovery and delivery. Various tools were analyzed to find a better solution by applying techniques that other researchers have used to develop a more robust and efficient search engine. The method used developed and deployed a relational database containing geospatial metadata documentation for all datasets within the spatial library and a web interface designed to efficiently query the database. This method holds great promise for helping website visitors more effectively find required geospatial data. The main purposes of this study were to: 1) develop a robust database design that is physically independent of a specific database engine and 2) populate and deploy the relational database on the Internet using a customized search application. Various Relational Database Management System (RDBMS) engines were used during the development and testing process, such as, Microsoft Structured Query Language (MS SQL) Server, Microsoft Access and International Business Machine Database 2 (IBM DB2). The database schema was developed using Unified Modeling Language Computer- aided software engineering (UML CASE) modeling. This solution is currently deployed using Microsoft Access.

KEYWORDS: IBM DB2, RDBMS, Visual Basic, Visual Studio, GIS, metadata

INTRODUCTION

To manage and support sophisticated geospatial data searches, a relational database was developed to act as the backbone against which a search interface can be executed. The relational database design was intended to improve data discovery and delivery by including all information contained within geospatial metadata documents. The Federal Geographic Data Committee (FGDC) geospatial metadata format was chosen as the base metadata standard for this project.

The FGDC metadata standard describes GIS datasets using seven major sections and three supporting sections. The seven major sections are: 1) identification, 2) data quality, 3) spatial data organization, 4) spatial reference, 5) entity and attribute, 6) distribution, and 7) metadata reference information. The three supporting sections are: 1) citation, 2) time period, and 3) contact information. In addition to this information, the relational database developed for this project also contains the name and URL of the geospatial dataset along with descriptive keywords to better facilitate searching and downloading. A customized extensible markup language (XML) Parser application was also designed to load FGDC geospatial metadata stored within XML files into a normalized relational database. This allows data from many XML files to be searched, allowing simple and complex queries to locate and find information on data to determine data availability.

In addition to improvement in search and query functionality for end-users, an additional benefit of this approach is the reduction in overall disk space usage and the removal of redundant stand-alone HyperText Markup Language (HTML)-based metadata files by importing metadata files into a relational database. The web interface will search the data based on the information in the database. It is anticipated that this will improve long-term maintenance of the archive, as shown in Figure 1.

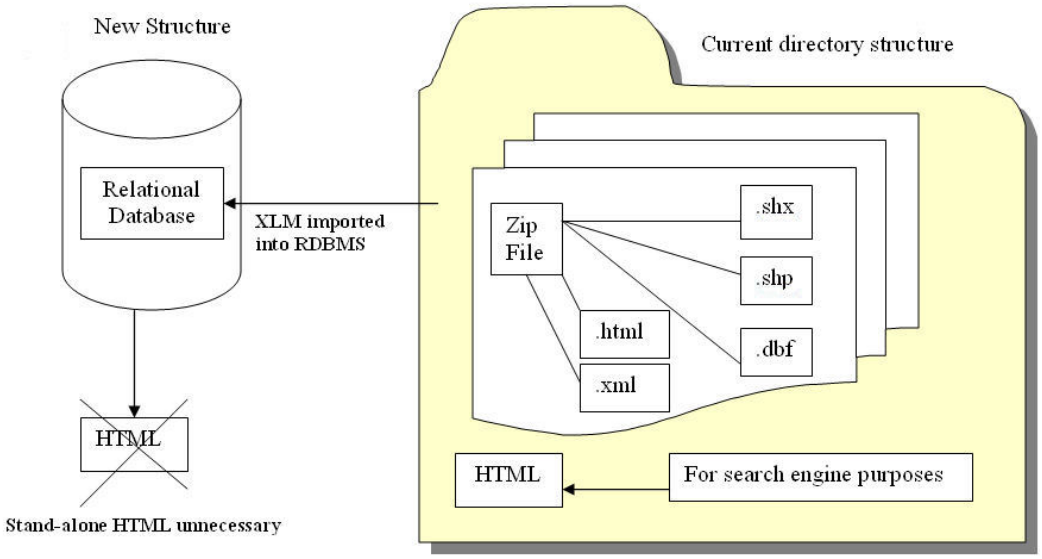

Figure 1. Reduction of HTML files improves long-term maintenance of the archive

METHODS

Database Design Using a UML CASE Class Diagram

At the outset of this project, a database schema (design) that would support geospatial metadata documents did not exist. For this reason, the first step was a careful database design normalized to third normal form. Normalization helps to remove data inconsistencies that result from data redundancies (Rob and Coronel, 2002). UML is a standard diagramming language used to design database schemas, among other things, and the software tool that was used to create UML models is Rational Rose (Boggs and Boggs, 1999). To manage and support sophisticated searches, a relational database with tables describing

GIS datasets based on the seven major sections of the FGDC metadata standard was created using Rational Rose. This database design includes the information captured within geospatial metadata documents, the name and the file path to the geospatial dataset, and descriptive keywords (Figure 2).

Based upon a careful review of both the Environmental Systems Research Institute (ESRI) profile of the content standard for digital geospatial metadata (CSDGM) and the FGDC geospatial metadata format, 50 tables representing the base metadata standard were created. These tables describe GIS datasets that include all major sections and supporting sections.

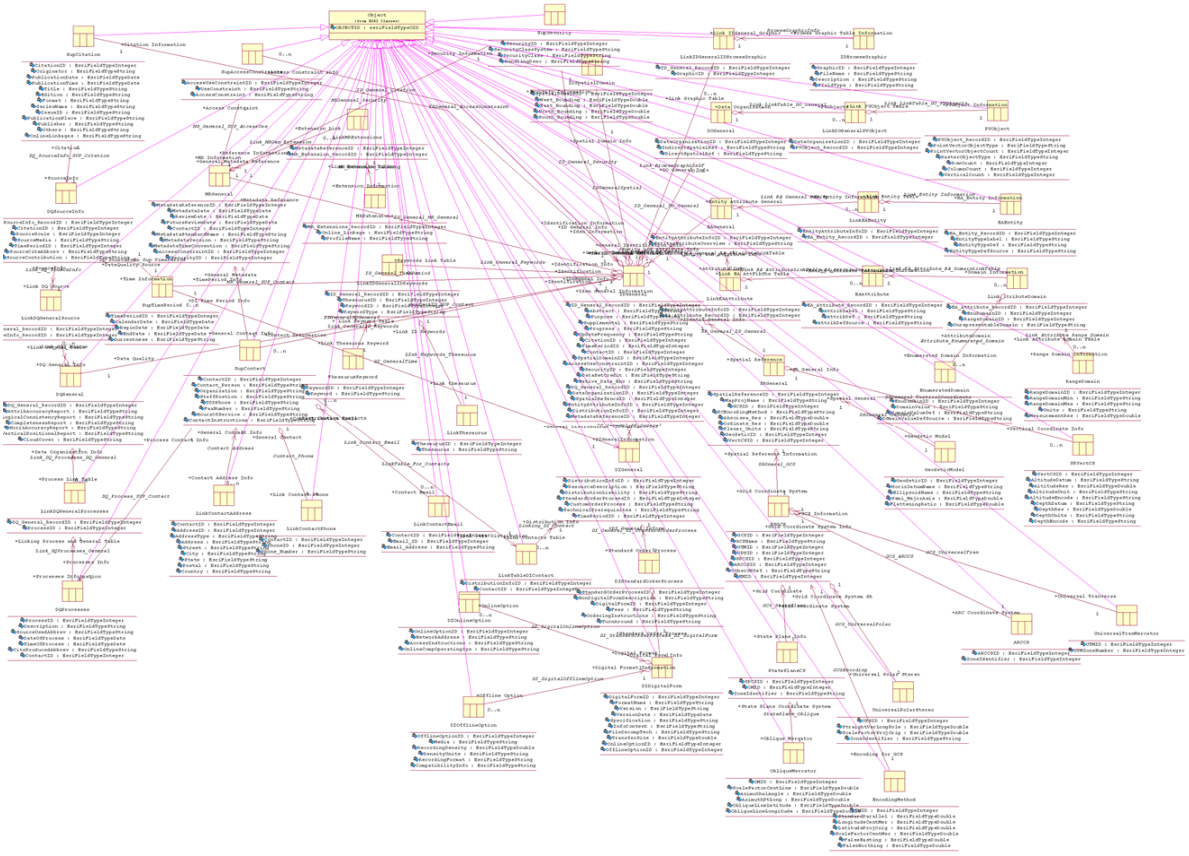

Figure 2. The UML Class Diagram for the relational database developed using Rational Rose, http://giscenter-sl.isu.edu/umlmodel/index.htm

Seven major sections are represented in the tables: identification information (***IDGeneral***), spatial reference information (***SRGeneral***), entity and attribute information (***EAGeneral***), distribution information (***DIGeneral***), data quality information (***DQGeneral***), metadata reference information (***MRGeneral***), and spatial data organization information (***DOGeneral***). The supporting sections are represented in the tables: citation information (***SupCitation***), time period information (***SupTimePeriod***) and contact information (***SupContact***).

Tables are organized into rows and columns (tuples and attributes) and some of the tables are related to other tables in the database. Using the example in Figure 3, observe that the primary key (DataOrganizationID) of the DOGeneral table uniquely identifies each row in its table. Likewise, the primary key (PVObject_RecordID) of the PVObject table is used to identify each record. LinkDOGeneralPVObject is a composite table that is used to link the tables (DOGeneral and PVObject) and thereby eliminate the possibility for redundancies. Therefore, the LinkDOGeneralPVObject table

consists of a primary key that is the combination of the foreign keys (DataOrganizationID and PVObject_RecordID).

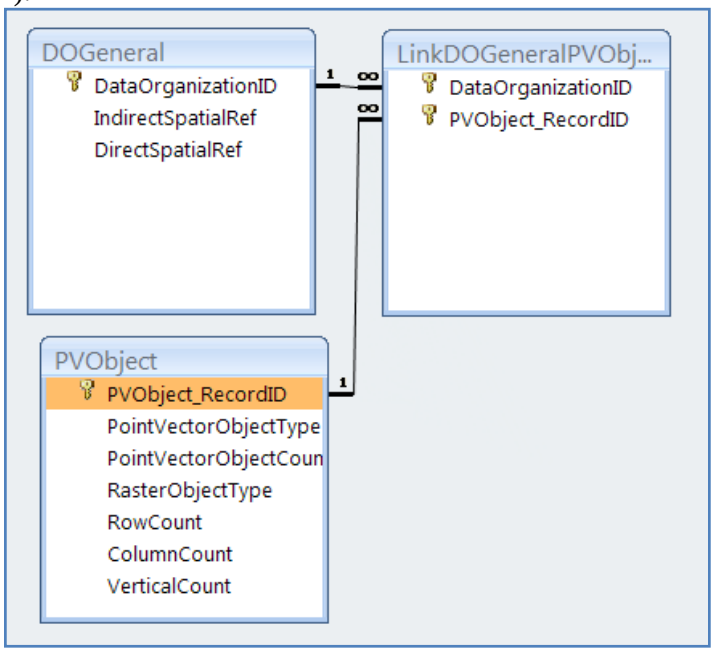

Figure 3. Sample of a relationship

After designing the database schema using the UML Class diagram model, the XML metadata interchange (XMI) file was exported from Rational Rose in order to transition into a physical database. "XMI is an Object Management Group (OMG) standard that specifies how to store a UML model in an XML file" (Perencsik et al, 2004).

The advantage of using a UML model is its ability to transition into a physical database using the UML class diagram model. Changes can be made to the model and exported using the XML file (consisting of UML XMI) that facilitates transfer of UML diagrams to other software applications (such as ArcCatalog). "When a change occurs to the model, Rose can modify the code to incorporate the change" (Boggs and Boggs, 1999), this helps to ensure that the model is resilient and flexible.

After exporting the XMI file from Rational Rose, a Microsoft Access relational database was created using the computer-aided software engineering (CASE) schema tool within ArcCatalog. "The CASE tools subsystem lets you create blueprints of the structure of the geodatabase using a graphical language – the Unified Modeling Language (UML)" (Perencsik et al, 2004). However, before running the CASE tool wizards, a semantic checker was used inside Rational Rose to make sure that the model stored in the XMI file was correctly defined and followed the set of modeling rules described by the ESRI framework. The semantic checker produces a report where it lists the errors that were found in the model. In this way, the model could be found to be free of errors before running the CASE tool wizards. The UML design is available at http://giscenter-sl.isu.edu/umlmodel/index.htm

XML Schema Testing on Various RDBMS Engines

The efficiency of using the XMI file that was exported from Rational Rose to transition into different databases (such as, MS SQL Server Express and IBM DB2) was tested.

The strategy for using the UML and CASE Schema tool from ArcCatalog to design and create the database involves using UML to define the schema and generate the XMI file. Before transitioning the

XMI file into Microsoft Access, a new personal geodatabase was first created and the CASE Schema Wizard tool was used to transition the schema into the database. Later, the same XMI file was used and transitioned into a MS SQL Server Express database by first creating a new geodatabase and repeating the process as described above. Users do not have to concern themselves with data conversion from the keyword ESRI Field Type (e.g. esriFieldTypeInteger) to standard database field types (e.g. Integer) as the CASE Schema wizard tool will perform the conversion automatically.

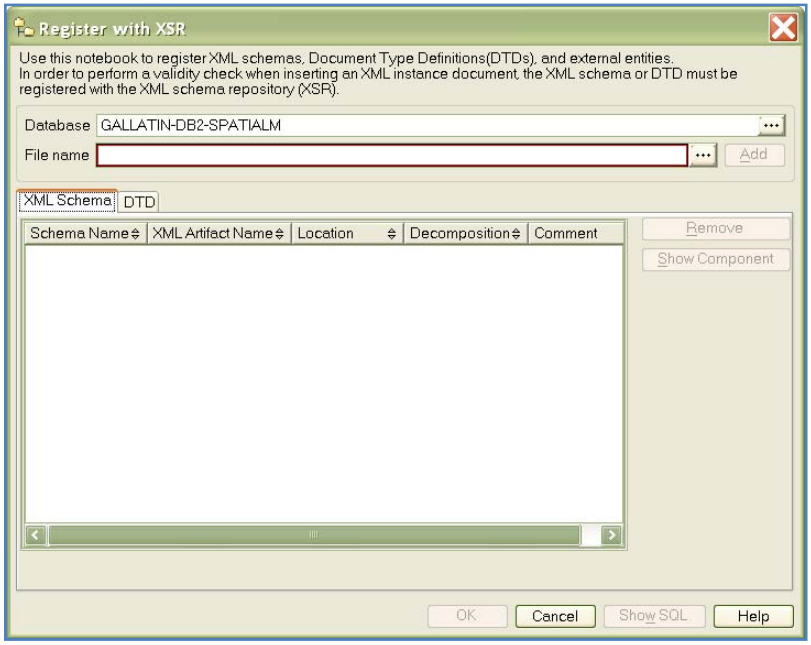

Figure 4. Register with XSR

For IBM DB2, a new database was created. Once a new database has been created, the XML Schemas, Document Type Definitions (DTD), and external entities need to be registered with the XML Schema Repository (XSR) (Figure 4). This needs to be done in order to perform a validity check when inserting the XML schema (*.XSD files). IBM DB2 allows either DTD or XSD files to be registered. In this particular instance however, the initial language used while creating the UML design was the default – "Analysis". Therefore, the UML design was saved as a Rose model (*.mdl file) that can't be registered with the XSR. If the initial language was set to XML_DTD, then the design would be saved as an *.XSD files. In this case, the file could be registered and the schema used to build the database.

Since the database design was successfully converted to each type of database in the test set, it indicates that the database design is robust and physically independent of a specific database engine. In other words, the same XMI file that encloses the UML class diagram can be transitioned into various database engines, such as Microsoft Access, MS SQL Server Express and IBM DB2.

Development of XML Parser Software

To populate the relational database, all information stored in HTML-based metadata files (approximately 30,000) needed to be imported. Parsing such metadata files can be quite difficult because of the variations of formatting one might encounter in different HTML files. Therefore, the HTML-based metadata files had to be converted to XML metadata files. There are currently only 2000 XML metadata files. This process could be done by using Microsoft Word and the U.S. Geological Survey (USGS) metadata utility "cns" to produce a well-formatted text file, which was then imported into ArcCatalog via the "import metadata tool". With the metadata stored within the standard ArcXML file format, the

process of parsing and importing metadata became greatly simplified. An "XML Parser" application was written using Visual Basic 2005 to perform this function.

Using the XMLTextReader function in Visual Basic 2005, the XML Parser application could import ArcXML formatted metadata to the relational database. The goal of building the XML Parser software was to read an ArcXML metadata document, parse it, extract the values (fields and attributes), and write the data to the appropriate relational database tables.

A graphical user interface (GUI) was designed (Figure 5) for the XML Parser program. The GUI includes basic functions (such as Open, Save, Add, Remove Files, and Open a folder) within two areas on the page: a file screen and a preview screen. The file screen displays the name of all XML files contained in the current workspace. The preview screen previews selected XML files along with their keys and values. Keys are attributes that are used for selection during data retrievals.

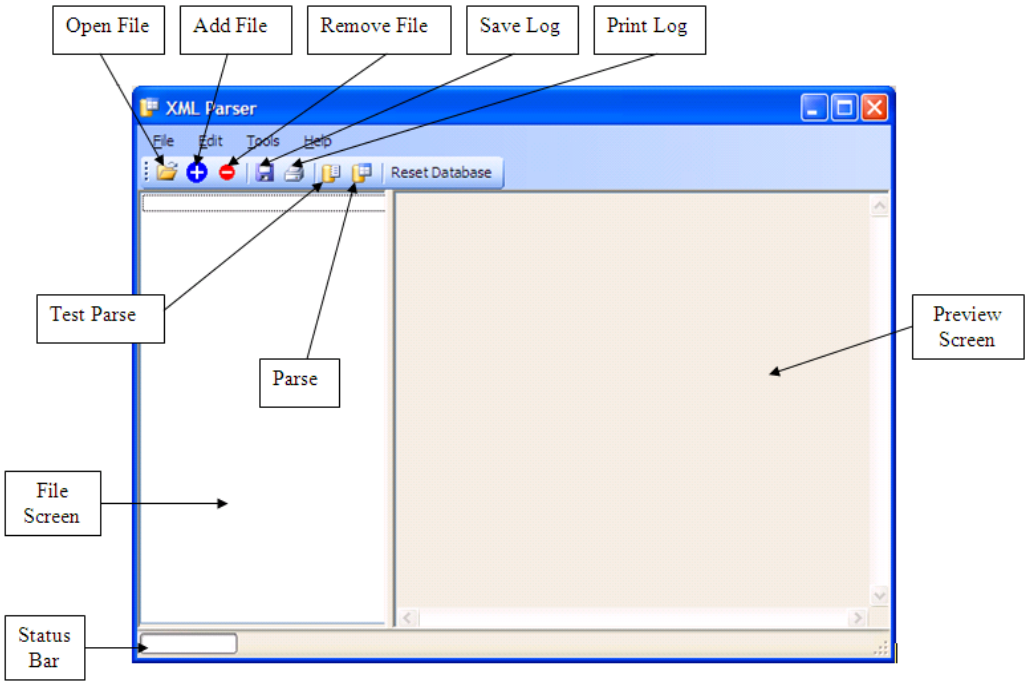

Figure 5. Screen of XMLParser software

Exploring XML Parser

Hash tables were used to store values inside the XML tags. "A hash table is a fixed-length array of pointers to link lists of nodes; each node contains a string and a pointer to the next node" (Zobel et al, 2001). These values were then inserted into the database (MetadataSpatialGDB.mdb) using the SQL SELECT statement. Active Data Object (ADO) connections were created that executed the SQL commands and inserted the values into the database.

After selecting a set of XML files for parsing, the user can parse them to the database directly by clicking the parse button  . "The numeric conversion functions examine the value stored in the argument and attempt to convert it to a number in a process called *parsing*, which means to pick apart, character by character, and convert to another format" (Bradley and Millspaugh, 2003). In this case, parsing means the process of extracting the values stored inside the XML tags. The XML Parser application will parse all files in the file index list.

When the parse button  is activated (the parse button click event), the *ReadMetaID* and *ReadOnlineLink* sub procedures are called to retrieve the MetaID (a Globally Unique Identifier (GUID)) and Online Linkages, which store the locations of the files using a Universal Resource Locator (URL). The *FieldExists* sub procedure is then called, and uses these same two attributes to determine if the XML file has already been parsed and its data already exists in the relational database. These two attributes were used instead of the GUID because if the ArcXML metadata files were created using a template (as is often done with automed software like NPS metadata tools), all ArcXML files will have the same GUID as the template. Therefore, the user would not be able to store the files. To avoid this situation and allow the use of automated software, two alternative unique attributes were used to identify the XML files and allow numerous XML files with the same GUID -- but different online linkages -- to be stored within the database.

If the record does not exist in the database, then the *ParseTheFile* sub procedure is invoked. If the record does exist, the *File Exists* message is displayed requiring the user to choose to either "update the record" or "cancel" the parse process. If the user chooses to "update", the *DeleteRecord* sub procedure is first called to delete the particular record(s) and it then inserts the updated records by calling the *ParseTheFile* sub procedure.

Inside the *ParseTheFile* sub procedure, the Select CASE function allows the parser to compare the XML tags against multiple criteria. "Select CASE is able to determine the choices by the value of selector and take appropriate actions (Schneider, 2005). "Each of the possible actions is preceded by a clause of the form ***Case valueList,*** where *valueList* itemizes the values of the selector for which the action should be taken" (Schneider, 2005). If the XML tags match one of the criteria, all the information within the tag is extracted and the values are inserted into a hash table. The tag name and the values within the tag are stored as keys and key values inside the hash table (Figure 6). There are eight CASE selections, which include the seven major sections of FGDC geospatial metadata and ESRI header information.

The sample code shown in Figure 6 shows that the XML text reader function reads the node trees and if the child element of the parent node matches the Select case criteria, such as "spref", then ReadSubTree method is used to create a boundary around that specific element. Therefore, the application can work with the information inside the "spref" element instead of the whole XML document. Next, the variable "NameValue" reads the whole string including the XML tag within the element. For example, <semiaxis Sync="TRUE"> 6378206.400000 </semiaxis> will be read into the "NameValue" (Figure 7). In this case, "semiaxis" will be stored as a key and the value within "semiaxis" (6378206.400000) will be the value for its key and added to the hash table called "spref" that was declared at the beginning of the program code, thereby identifying spatial reference information group. However, if there is any XML tag that is repeated, it will be caught in the Try…Catch statement since the hash table does not allow duplicate keys. Under the Catch statement, the count will be incremented and added as a suffix after the key's name to avoid duplication.

Seven text files were created representing the list of possible attributes included within the identification, data quality, spatial data organization (Figure 8), spatial reference, entity and attribute, distribution, and metadata reference information major section.

```
            'parse the file
            While (Reader.Read)
            Select Case nodeName
               Case "spref"
                    subReader = Reader.ReadSubtree
                    Dim Name As String
                    Dim NameValue As String = ""
                    Dim Count As Integer = 1
                    Dim Value As String = ""

                    ' unicode for quote characters
                    Const DoubleQuote As Char = ChrW(39)

                    While (subReader.Read)
                       If subReader.NodeType = XmlNodeType.Text Then
                       'Add the keys and values inside the spref Hash Table
                       Try
                         NameValue = subReader.ReadString
                         If NameValue.Contains("'") Then
                            Value = Replace(NameValue, "'", DoubleQuote & "'")
                            NameValue = Value
                         End If
                          spref.Add(subReader.Name, NameValue)
                          'If the same key exists, then add count number to
make a new key
                         Catch
                              Name = subReader.Name & Count
                              spref.Add(Name, NameValue)
                              Count += 1
                         End Try
                      End If
                    End While
                End Select
            End While
```

Figure 6. Sample Code showing Case Selection

```
<geodetic>
  <horizdn Sync="TRUE">North American Datum of 1927</horizdn>
  <ellips Sync="TRUE">Clarke 1866</ellips>
  <semiaxis Sync="TRUE">6378206.400000</semiaxis>
  <denflat Sync="TRUE">294.978698</denflat>
</geodetic>
```

Figure 7. Sample XML tag

Abstract	browset	cntper	edition	northbc	publish	sername
accconst	bottombc	cntpos	enddate	onlink	pubplace	southbc
address	caldate	cnttdd	geoform	origin	pubtime	state
address	city	cntvoice	hours	othercit	purpose	supplinf
addrtype	cntemail	country	issue	place	rightbc	theme

Figure 8. A list of attributes that may be found in the spatial data organization information major section

After parsing the file, the *InsertInfo* sub procedure (Figure 9) is called to compare the attribute names inside the appropriate text file with the keys inside the hash table. If the attributes contained in the text files are not found in the hash table, then the attribute names are added to the hash table with an empty string value. This is done to eliminate an element not found error while performing the SQL INSERT statement later on.

```
Sub InsertInfo(ByVal FileName As String, ByRef HashName As
Hashtable)
   Dim sr As IO.StreamReader = IO.File.OpenText(FileName)
   Dim De As DictionaryEntry
   Dim HashCount As Integer = HashName.Keys.Count
   Dim Counting As Integer = 0
   Dim FoundValue As Boolean
   Dim valtofind As String
   Do While sr.Peek <> -1
       valtofind = sr.ReadLine
       FoundValue = False
       For Each De In HashName
           If CStr(De.Key) = valtofind Then
               FoundValue = True
               Counting = 0
               Exit For
           Else
               Counting += 1
               'Debug.Print(CStr(De.Key))
               Continue For
           End If
       Next De
       If FoundValue = False Then
           HashName.Add(valtofind, "")
           Counting = 0
       End If
   Loop
   sr.Close()
End Sub
```

Figure 9. The InsertInfo Procedure

The *InsertDatabase* sub procedure is then called to insert the records into the MetadataSpatialGDB.mdb database. The data is inserted according to the seven major sections.

Insert Records into the Database

There are three main steps for inserting records into the database: 1) insert data to individual tables (Figure 10), 2) retrieve the primary key generated for each record (Figure 11), and 3) insert the primary keys into the link tables that have one-to-one, one-to-many, and many-to-many relationship (Figure 12).

```
CommandQuery = "INSERT INTO DQGeneral" _
& " (AttribAccuracyReport, LogicalConsistencyReport,
CompletenessReport, HorizAccuracyReport,
VerticalPositionalReport, CloudCover) " _
& " VALUES ('" & Dataqual.Item("attraccr").ToString & "', " & "'"
& Dataqual.Item("logic").ToString & "', " & "'" &
Dataqual.Item("complete").ToString _
& "', " & "'" & Dataqual.Item("horizpar").ToString & "', " & "'"
& Dataqual.Item("vertaccr").ToString & "', " & "'" &
Dataqual.Item("cloud").ToString & "')"
strSQL.CommandText = CommandQuery
strSQL.Connection = connection
strSQL.ExecuteNonQuery()
```

Figure 10. Step 1 – Insert relevant information into the individual tables

```
'Get the primary key generated
CommandQuery = "SELECT Max(DQ_General_RecordID) FROM DQGeneral;"
strSQL.CommandText = CommandQuery
strSQL.Connection = connection
reader = strSQL.ExecuteReader
strSQL.Dispose()
'Return the primary key that generated
If reader.Read() Then
  DQGeneralRecordID = Convert.ToInt32(reader(0))
End If
```

Figure 11. Step 2 - Retrieve the Primary key generated from each record

```
'Insert all the values stored in the array
If dataqualprocessid.Count > 0 Then
  For I = 0 To dataqualprocessid.Count - 1
CommandQuery = "INSERT INTO LinkDQGeneralProcesses" _ & "
(DQ_General_RecordID, ProcessID) " _ & " VALUES ('" &
DQGeneralRecordID & "', " & "'" &
dataqualprocessid.Item(I).ToString & "')"
   strSQL.CommandText = CommandQuery
   strSQL.Connection = connection
   strSQL.ExecuteNonQuery()
  Next
End If
```

Figure 12. Step 3 - Insert the primary keys into the link tables

The record is inserted by calling the sub procedures accordingly: spatial reference information (*SR_General*), data quality information (*DQ_General*), entity and attribute information (*EA_General*), metadata reference information (*MR_General*), distribution information (*DI_General*), data spatial data organization information (*DO_General*) and identification information (*ID_General*). The XML Parser will retrieve all the primary keys for each major section before inserting data into the Identification information table.

The XML Parser will then continue the parsing operation until completion of the last file in the list, at which point a message will be displayed indicating the results of file loading. The message “Complete Inserting to Database” is shown when all files have been successfully inserted into the database.

Efficiency Testing of XML Parser
Based on the response time that I set, I assumed if the XML Parser software was able to load one metadata file with all tags of data represented in less than 30 seconds then the software is considered as efficient. Testing the system was done by loading 100 XML files at once and inserting the files into the database in less than 2 minutes. The system met the performance requirements (<30 seconds per file).

Currently there are approximately 2000 XML metadata files stored in the relational database (MetadataSpatialGDB.mdb). When the rest of the GIS Center HTML-based metadata files are converted to XML metadata files, they will be inserted into the database for metadata files search.

RESULTS AND DISCUSSION

To better facilitate discovery and delivery of geospatial data for GIS TReC clients, a robust relational database containing FGDC-compliant geospatial metadata was developed and deployed. This database design is physically independent of a specific database engine and was tested by transitioning the schema into different physical databases (Microsoft Access, MS SQL Server Express, and IBM DB2). However, before running the CASE Schema tool, the semantic checker was used to make sure the model stored in the XMI file was correctly defined according to the ESRI framework.

After transitioning the schema into a database, the geodatabase rules were removed since the relational database only consists of tables and not feature classes or feature datasets. This represents a benefit for database developers as it shows that it is possible to transfer a database design to another database without manual recreation.

Each database has its own advantages. If the user only wants to store a small amount of data, then the class diagram model should be transitioned into a Microsoft Access personal geodatabase. However, if the user requires larger storage spaces to accommodate more data (>2GB), then a SQL Server Express or IBM DB2 are recommended.

The custom built XML Parser loads metadata stored within ArcXML files into a Microsoft Access database. There were three components in the parser software: the graphical user interface component, the XML parsing component, and the insert records component. The GUI component initializes the program and allows the users to select XML files to insert into the database and displays the results of the file loadings. The XML parsing component extracts the values inside the XML tags of the XML metadata files and stores them as keys and keys values in the hash tables. The insert records component writes the data into the appropriate table(s).

All code for the UML class diagram model and the executable for the XML Parser software are freely available under the L-GPL open source license and can be downloaded here: http://giscenter.isu.edu/research/techpg/nasa_tlcc/results.htm. Users can download the files, deploy the solution, and populate their own database

ACKNOWLEDGEMENTS

This study was made possible by a grant from the National Aeronautics and Space Administration Goddard Space Flight Center which was made possible through efforts of the Idaho congressional delegation.

LITERATURE CITED

Boggs, M. and W. Boggs 1999. A Tour of Rose. M*astering UML with Rational Rose (pp.34).* California: SYBEX.

Bradley, J. and A. Millspaugh 2003. *Programming in Visual Basic .Net.* New York, NY: McGraw-Hill Higher Education.

Perencsik, A., E. Idolyantes, B. Booth, and J. Andrade 2004. *ArcGIS 9: Introduction to CASE Tools.* Retrieved December 6th, 2007 from http://support.esri.com/index.cfm?fa=knowledgebase.documentation.viewDoc&PID=43&MetaID=658

Rob, P. and C. Coronel 2002. Normalization of Database Tables. *Database Systems Design, Implementation, & Management (5th ed.).* Boston, Massachusetts: Course Technology.

Schneider, D. 2005. *An Introduction to Programming Using Visual Basic 2005, sixth edition.* Upper Saddle River, NJ: Pearson Prentice Hall.

Zobel, J., S. Heinz, and E. H. Williams 2001. *In-memory Hash Tables for Accumulating Text Vocabularies.* Retrieved March 30th, 2007 from http://goanna.cs.rmit.edu.au/~jz/fulltext/ipl01.pdf