An Innovative Approach to Processing and Converting Environmental Data

William W. Ferrell
Warren Macchi
Anthony A. Barresi
Abamis IT Solutions
Orlando, FL
wferrell@abamis.com, wmacchi@abamis.com, abarresi@abamis.com

Farid Mamaghani
SEDRIS Organization
farid@sedris.org

Keywords:
SEDRIS, GIS, data conversion, environmental data, templates, data models, mappings

ABSTRACT: *A common task in the preparation, modeling, consumption, and interchange of environmental data is converting the data from a particular format and/or representation into another. It is not uncommon that during such conversions useful information is lost, needed metadata dropped, precision is reduced, and/or artifacts that were not present in the original data are introduced. This paper describes an approach to environmental data conversion that uses SEDRIS to manage these issues and to aid in the mapping between different environmental data models. Being able to describe the source and destination data models using a common terminology and representation allows for faster, more efficient, and reusable development processes. The approach described in this paper uses SEDRIS technologies, including the Data Representation Model (DRM), XML-based Transmittal Content Requirements Specification (XTCRS), Environmental Data Coding Specification (EDCS), Spatial Reference Model (SRM), and a novel template-based description of environmental data processing algorithms, to process and convert the data that is available in widely used formats from one representation into another.*

1. Introduction

Processing environmental data is a key component of current simulation applications. Without trusted and reliable sources of data and accurate processing of it, a simulation may lose its credibility, and its use may become severely limited. It is therefore vital that the processing of environmental data from its source to a simulation's components be clearly understood. This paper addresses some of the key issues that are encountered by systems engineers and software developers in the modeling and processing of environmental data.

A common (but not always appropriate) first consideration in processing environmental data is the file format of the data. These file formats impose constraints and requirements on the development of the processing software. These impositions are not always immediately obvious. To illustrate this, consider the Digital Terrain Elevation Data (DTED) format [1], which is used for storing terrain surface elevation data. The data models of the different software libraries that can read this format are often quite different. Hence, the software developer will be required to either restrict the software design to use the reader's data model or develop mappings from the reader's data model to the application's internal data model. In either case, later when a different source of elevation data is required (for example, one using the Geospatial Tagged Image File Format (GeoTIFF) [2], the application's design will require additional (sometimes major) modifications to handle the new data source.

In addition to the data model of a software library used to read a particular format, the software developer must also consider the data models used in components that need to process the environmental data. For example, consider a software component, developed by a third party, which triangulates multi-sided polygons. The data model used in the simulation application will most likely be different than the data model used by the third-party library. Hence, the developer is forced, once again, to map from/to internal data structures to/from the third-party library's data structures. Furthermore, if a developer is at some point required to replace this library with another library's implementation, the task of re-creating the data model mapping software will need to be repeated.

These challenges, and practical solutions to them, are discussed in the remainder of this paper through the perspective of software developers and systems engineers. Section 2 describes these challenges in greater detail and discusses various choices for addressing them, along with some of the implications of those choices. It also provides a summary of the SEDRIS components and an approach to address these challenges. Section 3, the central focus of this paper, discusses the application of SEDRIS and supporting technologies in a unified approach to the processing and conversion of environmental data. Section 4 concludes the paper with a summary of the results of the work to date and a discussion of future work in this area.

2. Environmental Data Challenges

Simulations use a wide variety of data sources to perform their intended functions. Many of the challenges that arise in the processing of environmental data are not unique to the simulation field. Software developers who need to read, write, and process environmental data can encounter a variety of unanticipated challenges, including the following:

- Data might be stored in several databases, possibly in multiple formats.
- Data might contain customized or extended data structures that are not fully documented.
- Data might contain far more data (whether in its detail or geographic coverage) than the simulation or the processing application can reasonably handle (processing time and/or memory constraints).
- Data might be represented or organized in a way that cannot be used directly.
- Data might require conversion into an optimized representation for space or performance reasons.
- Simulation's and/or the processing application's data models might need to be mapped to/from other data models so that data can be processed by third-party libraries.

To address these challenges, a software developer must make design decisions across the data processing pipeline. These decisions have a direct impact on the flexibility, modularity, expandability, reliability, and performance characteristics of the data processing applications. Hence, a thorough understanding of the impact of the decisions is a key step towards developing trusted and reliable applications and simulation systems.

Unfortunately, since sometimes file formats are erroneously the first but, nevertheless, an important, consideration in processing environmental data, we focus first on those.

2.1 File Formats

File formats are usually divided into two general types: text and binary encodings. In text files, the data is stored using human-readable characters. Most files using a text encoding can be examined using common text editors (such as *vi*, Notepad, or TextEdit).

An eXtensible Markup Language (XML) document is an example of a text file, but one that follows a very specific set of rules regarding its content organization and structure [3]. In an XML file the information is provided using markup and content. An example of an XML document is `<name>Sienna</name>`, where `<name>` is a start tag, `Sienna` is the content, and `</name>` is an end tag. In XML syntax, the entire string, which includes the tags and content, is referred to, in this case, as the *name* element. XML elements may contain other XML elements.

Note that without additional context it is not clear what the "name" element is actually representing. Is it a person's name, the name of car models, a material name or something else? This is because the XML syntax specification is simply a way to delineate information in a text presentation. One needs additional information to interpret the semantics of that information. Hence, without an associated schema for its interpretation, an XML file by itself is not necessarily understandable just because it is a text file. Some examples of environmental data file formats utilizing XML are the Geography Markup Language (GML) [4] and the COLLAborative Design Activity (COLLADA) Digital Asset and Exchange Schema [5].

In addition to common text editors being able to read and write text files, specialized parsers have been developed for specially formatted text files (including XML) that are widely available. Unfortunately, in the absence of editors, it is very difficult to use general-purpose editors to create large data sets, and files can easily become corrupted. Another disadvantage of text files is that they tend to be quite verbose, and therefore moderate to large datasets typically require significantly larger computing resources (e.g.,, storage, memory and processor cycles) than binary encodings of the same dataset.

A binary encoding is another way of storing data in a file (in fact, technically even text files are stored as "binary data"). A binary file format generally refers to byte streams that are not based on text encoding. One reason for using non-text files is the efficient and more natural way data can be represented in computer memory. For example, the integer value "59287" is not typically stored as five text characters in memory. Rather, it is stored as a group of bytes, such as the hexadecimal value E797 (where, in fact, the two bytes E7 and 97 represent the decimal values of 231 and 151, respectively). Another example is storage and representation of floating point numbers, such as "84103.109375", which is stored in four bytes as the hexadecimal value 47A4438E rather than twelve text characters. Because programmers develop software by manipulating data structures in computer memory, it is easier to store the data in memory directly in a file (since files are also a linear sequence of bytes). Then, to restore a program's state, all that is required is to read the data in the file directly into memory. This is faster and more efficient.

Binary encodings are generally used when compactness of the representation and efficiency of data access are the primary concerns. Some drawbacks of binary encodings include the need for specialized libraries to access the data stream; the need for specialized "editors" to manipulate or edit the data; and the difficulty in making changes to the file format. A change to a binary format may often require changes to the parser software in the file format library to account for the different organization of the byte stream.

Examples of binary file formats are OpenFlight (Presagis Inc.) [6], Shapefile [7], and the SEDRIS Transmittal Format (STF) [8]. Note that some file formats require more than one file to store all the data. For example, a Shapefile is typically composed of at least a file ending with a ".shp" extension containing the geometry information, one ending in ".dbf" containing the field attributes of each geometry/shape, and one ending in ".shx" that contains an indexing of the ".shp" file for faster access to its contents.

Because of the efficiency of a binary encoding, simulation systems sometimes use a custom file format called a "runtime format" or "runtime database." The content of these runtime databases is a distilled, highly tailored, and customized version of the data. Such specialized data sets are generally not reusable across disparate systems, since their binary format is specific to the architecture and data requirements of a particular (hardware and/or software) simulation system.

2.2 File Format Libraries

In order to access the data, developers generally use a software library that provides the appropriate read/write interfaces to the desired file format. Developers may use existing library implementations (either made available by the providers of the format, or through commercial or open source providers), or may choose to implement their own access software (because existing implementations may be incomplete, not in a specific programming language, or have licensing restrictions). In either case, several important issues in utilizing a library implementation must be considered. In the remainder of this section, the term "library" refers to any software library used in reading and writing a particular file format.

The first consideration in choosing a library is the programming language used for the interface and the implementation of the library. The interface provided by a library might not be in the same language used in its implementation. Obviously, if the interface to the library is in a different language than what the developer uses, additional work to bridge the application and the library will be required. In addition, the language in which the library is implemented will affect how easy it is to integrate the library with the developer's software. Having access to the library's source code generally can speed up the software integration and testing processes.

Memory requirements and efficiency of data access are other key considerations. Some libraries may need to load all or large parts of the data stream before data can be accessed. With such libraries and when large environmental data sets are being processed, the memory and storage requirements may exceed the available system or application resources. Speed of data access is also an important consideration for large data sets. Libraries and file formats that are not properly designed for handling large data sets may require significant computing resources to access the data even when only small sections of the data need to be read or updated. For example, some file format libraries load all the source data into memory as soon as the (file) stream is opened. If the data covers a much larger area (or has additional information/layers) than the application needs, then memory and processing time is wasted loading data that will not be used. Hence, a file format library should either allow the developer to drive the data loading process (based on the application's requirements) and/or provide filtering mechanisms or other methods for specifying areas of interest.

In addition to efficiency, a well-designed library should also provide a flexible and robust implementation. Typical processing actions should be intuitive and straightforward for the developer to implement, but the interface should also provide mechanisms for adapting the data access to follow the pattern most suitable to the developer; for example, by providing querying, data filtering, and other processing options. The library implementation should also be robust and gracefully handle various conditions, including corruption or interruption problems in the data stream, problems that the developer's software may encounter (e.g., by providing useful information in error conditions), and problems with the computing environment where the library is being used (e.g., freeing/releasing all resources when the stream is closed). Such a library encourages and allows the developer to try different approaches for data access that might be more appropriate and/or convenient to implement.

Another important consideration is the availability of support for a library. While clear, concise, and well-written documentation is a key element in understanding the use of a library, the availability of example code and a responsive community is also invaluable. Sample applications that show the developer how best to use a library are extremely useful in getting a project started. When additional questions arise, having support from either the developers of the library or from a community of users can mean the difference between rapid development cycles or days of frustration.

2.3 Data Models

File format libraries often provide an Application Program Interface (API) that the developer uses to access the environmental data stream. These APIs generally consist of a set of software data structures and functions/methods that expose a (logical) data model of the format. Because these APIs are generally not standardized, different libraries for the same format are rarely interchangeable. Hence, the effort spent in learning one implementation's API (its strengths, weaknesses, and data model) is generally not applicable to another library implementation for the same format.

In addition, the simulation system and/or the application that handles the data access for the simulation has its own data model and internal APIs used to process the system's data. Because the requirements for a file format library and the requirements for the simulation/application are usually very different, their respective data models will likely be quite different also. Hence, the developer must spend time learning about each library's API and writing software to map from/to its data model to/from the application's (or simulation system's) data model. The development of reliable and efficient software for the mapping of these data models is not a trivial task and can consume a large part of the software development effort. Mistakes made in this part of the development can lead to performance implications across the length of the environmental data processing pipeline. Some of the challenges that a developer will face when learning about environmental data models include the following:

- Multiple types of data (e.g., raster, geometry, features)
- Multiple representations for the same type of data (e.g., contours, grids, and polygons for surface elevations)
- Efficiency of data structures (e.g., being able to access slices of a large elevation data set)
- Spatial reference frames (e.g., representing position information in different reference frames and conversion of coordinate values between them)
- Object classification and attribution schemes (e.g., EDCS [9], National System for Geospatial-Intelligence (NSG) Feature Data Dictionary (NFDD) [10]).
- Associations and linkages between environmental entities (e.g., relationships between shoreline and tidal state).
- Extensibility (e.g., handling new objects, object attributes, object representations).

It is clear that systems engineers and software developers are faced with a variety of sometimes daunting learning tasks. Therefore, reducing the number of data models (and the associated approaches and terminologies) a developer must learn will reduce the complexity of the development task. The above list also highlights the necessity of having clear and consistent documentation for the different data models that will be utilized. The use of standardized data models can help in this matter by providing the developer with knowledge, expertise, and consistent approaches that are reusable across different applications and designs.

2.4 Data Manipulation/Reformatting

In most applications the required environmental data will be retrieved, refined, and assembled from multiple data sources, file formats, environmental domains, and spatial coverage. Hence, the applications and tools throughout the pipeline must process the data to create an amalgamated and coherent view of the spatial area for the simulation exercise.

As noted earlier, one of the first steps in preparation for the processing of environmental data is to map the external data model(s) to the internal data model. The challenges that may occur in this stage range from converting simple data type and unit conversions to highly incompatible data models. For example, the source data model might provide the surface elevation as a regular grid of 32-bit integer values in feet and using an offset and a scale factor for each grid point. But the internal data model requires the elevation to be represented using a regular grid of floating point values in meters. In this case, a relatively simple conversion step can provide the mapping. However, if the source data model represents the elevation surface as polygons and includes underground structures (such as tunnels), the mapping would require more complex considerations.

Besides the mapping between external and internal data models, other common operations include processing and querying the environmental data. For complex data processing needs, developers sometimes use software libraries developed by a third party. For example, triangulation of polygons is a common operation that is not trivial in the general case. Therefore, a developer may utilize an external library that performs this function and integrate the library with the system. This integration step is similar to what the developer must do to map a file format library's data model to the internal data model, with the additional step that requires routing the data from the application to the third-party library and back.

An important consideration is how a developer can determine when the data needs to be processed before it can be used in the next stage. While intuitively one understands the statement "if a polygon has more than three sides, then it needs to be triangulated," implementing such a statement in software is much more complex due to variations in conditions, detecting those conditions, and the amount of low-level detail that a programming language requires. In general, for a given data set, the data of interest can be

- Present in a directly usable form (e.g., three-sided polygons)
- Present but in an unusable form that can be derived into a usable form (e.g., triangulation of polygons)
- Present but in an unusable form that cannot be (relatively easily) derived into a usable form (e.g., concave polygons that the triangulation algorithm cannot handle)
- Or be absent entirely (e.g., no surface data exists)

The ability to identify these conditions is critical to successful execution of applications that need to deal with environmental data. While an application may not be able to handle some of these situations, being able to identify them and generate an appropriate log of the condition can be extremely useful, both during the design phase and during the data processing.

If data is present but not in the required form, it can be passed to independent processing components that know how to reformat or transform the data. What is needed in this situation is a mechanism that (1) lets the developer recognize when the data is not in the appropriate form, and (2) helps the developer identify the algorithm(s) that can transform the data. It is important to note that identifying and invoking the algorithm in step (2) requires a mechanism for specifying and declaring its functionality; for example, by expressing that if the data is in form X, the algorithm can transform it to form Y. For step (1), the developer can benefit from a similar mechanism in the data processing pipeline that can identify the situations when the data can proceed along the pipeline and when it has to be transformed.

Because of the complexity and cost in creating software that can process any source of environmental data, most developers restrict their software to process only a limited set of source data (e.g., only DTED Level 1 or only non-compressed 16-bit GeoTIFF for terrain elevation). However, if a formal mechanism exists that can recognize those data organizations that the application cannot process directly, along with a mechanism to identify an algorithm that knows how to transform the data, then a more flexible and capable data processing approach can be created.

2.5 SEDRIS Approach to Environmental Data

The SEDRIS [11] approach to the representation and exchange of environmental data allows producers to describe their data in a way that is natural to them yet unambiguous to the consumers of the data. With this approach, standardized technologies have been developed that apply across all environmental domains and applications (e.g., ocean, terrain, urban, atmosphere, sensors, and space).

The STF is a binary file format for the encoding and transmittal of SEDRIS data. The format is optimized for file size and for random access of any part of the data content. A logical unit of data is called a *transmittal* and typically consists of two or more files. One file (with extension ".stf") acts as the root of the transmittal and contains information about the transmittal. The other subordinate files (also with extension ".stf" but numbered sequentially) store the actual data content.

The data is organized in a hierarchy of objects, where each object is an instance of a class from the SEDRIS DRM. There are a little over 300 DRM classes that cover the gamut of environmental domains and the relationships between environmental entities. For example, the root of the hierarchy is an instance of the DRM class `Transmittal Root`, which may be composed of instances of other DRM classes, such as `Image Library`, `Model Library`, and `Environment Root`. In turn, a `Model Library` object, for example, is composed of `Model` objects, which within their own hierarchies may be composed of such primitives as polygons, vertices, colors, etc. Associations between the DRM classes allow the creation of explicit relationships between object instances.

To allow for representational polymorphism, the DRM factors the common representations of the environmental objects, but does not include individual classes for such objects (such as trees, buildings, etc.). Instead it allows the identification of those representations by relying on the EDCS [9]. For example, the EDCS includes entries and corresponding definitions for a large array of environmental concepts and attributes. The label and/or code of these entries can then be attached to a `Model` to identify it as that environmental object. An online registry [12] allows users to browse and search all available EDCS concepts and also submit new concepts for registration.

The SRM [13] provides the framework required for the specification of spatial positions in a variety of Spatial Reference Frames (SRFs). An associated API allows for the conversion of coordinate values between different SRFs. Supported reference frames, datums, and coordinate systems span a wide range, which include all of the common reference frames such as Geodetic, Geocentric, Universal Transverse Mercator (UTM), and Polar Stereographic, along with other representations such as Military Grid Reference System (MGRS).

The SEDRIS API utilizes the DRM, EDCS, and SRM technology components to create instances of DRM classes and their relationships that can represent a wide range of environmental data. This DRM hierarchy can then be stored in an STF transmittal for interchange with other applications or subsequent updates and modification using the same API. In effect, the SEDRIS API is the file format library for the STF transmittal file format. All SEDRIS technology components are available as open source software development kits (SDKs), have been standardized through ISO/IEC (the International Organization for Standardization and the International Electrotechnical Commission), and are freely available for download [14].

A particularly relevant technology is the XML-based Transmittal Content Requirements Specification (XTCRS) [15]. The XTCRS is a formal language that is used to describe the content requirements of environmental data sets. The requirements specified in this language can be used to validate a SEDRIS transmittal (or a part of it) and verify that the constructs and the content of the transmittal meet the specific criteria. This data content specification technology forms the basis for the approach to environmental data processing and conversion described in the following section.

In addition to standards and SDKs, a set of tools for creating, viewing, and editing environmental data in the STF format, along with tools for converting data in several file formats to and from STF are also available [16]. The conversion tools allow a developer to write software based on the SEDRIS data model and then, by converting the source data to STF files, to read the source data without having to learn the data models of many other file formats.

3. An Innovative and Unified Approach

Instead of developing new software that maps data between different data models, the innovative approach discussed in this paper utilizes SEDRIS technologies to provide a unified framework for the processing and conversion of environmental data. In particular, the framework provides a clear, consistent, and flexible approach for specifying the input and output requirements of a system's components and algorithms.

3.1 Processing Environmental Data

When processing environmental data, a reasonable first step is to determine whether or not the data is in a form that can be easily digested. If the input data can be expected in one of several representations (for example, when numerical values are represented as integers versus floats, or when a surface is represented as a grid of regular points vs. polygons), the developer is faced with several choices. These include, designing the software to

- Be flexible enough to handle all possible representations
- Handle a limited set of representations and reject others
- Or handle a limited set of representations, but pipe the other representations through processing components that can map these representations into the set the developer's software can handle

The first approach is usually the preferred approach, but various practical limitations may not make this feasible. These may include project resource constraints (such as budget and schedule) or the required expertise for handling the more complex or unfamiliar representations. Hence, in order to quickly develop "working" software, developers might choose the second approach and hope that when more expertise/resources are available the additional flexibility will be added (i.e., converge to the first option). Unfortunately, often such software is not designed in a way that is easy to adapt to handling new representations; therefore, the software becomes brittle and more difficult to maintain after each modification.

To realize the last approach, the developer will need to design the software to recognize when the data is in a representation that can be handled, and if not, to be able to invoke a proper component that can map the representation into one the developer's software can handle.

As noted earlier, the SEDRIS XTCRS, and its associated SDK, provides the means for software to determine whether a SEDRIS transmittal (or a part of the transmittal hierarchy) satisfies a set of requirements. Therefore, the processing component can use an appropriate XTCRS to evaluate the input data stream and determine if there is data that matches the representations that can be handled by the developer's software. When the input data does not match one of the representations that can be handled, the software can then query a set of available processing components and invoke the appropriate one to convert the data into a different representation that the software can handle.

The process for querying those components that can handle other/unfamiliar representations can be accomplished using the same methodology. That is, each processing component can be described by both an input and an output XTCRS that the software can query and validate the input data against. Obviously the output XTCRS is one that is compatible with the representation that the developer's software can handle, but by specifying the input and output requirements, it is possible to create a catalog of well-defined processing components.

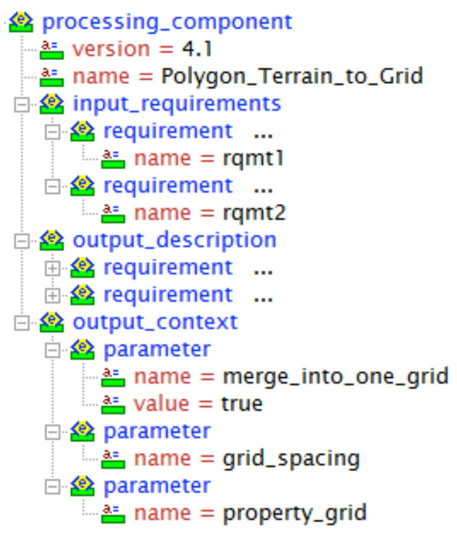
This ability to create a catalog of environmental data processing components using a comprehensive and unambiguous language (XTCRS) can be an effective approach to the reuse of these components. In addition, by encapsulating and cataloging the various algorithms and components, the developer's expertise can be focused on specific areas of the processing of environmental data without the need to learn and re-engineer a monolithic data processing pipeline.

Another advantage of using the SEDRIS DRM as an internal data model is the use of the file system to temporarily store (potentially large) environmental data fragments. For example, if the processing pipeline requires references to multiple sources of data that need to be conflated into an integrated data set, then the SEDRIS API can provide an efficient mechanism for managing the data. This is because the SEDRIS API has been designed for high performance access by caching frequently used data and for memory efficiency by automatically writing to the file system (and releasing resources for) the data that is no longer being referenced.

When an application that processes environmental data uses the SEDRIS DRM as its internal data model, then each of the environmental data processing components can be described by a template composed of

- An XTCRS describing the input data requirements
- An XTCRS describing the output data organization
- A set of variable name/value pairs to be applied to the output XTCRS to guide/restrict the processing algorithm

An example of this template is shown in Figure 1. This template was used to describe the `Polygonal Terrain to Gridded Terrain` processing component of the STF to Compact Terrain Database (CTDB) Converter. This processing component is a complex algorithm that uses a Fast Fourier Transform to identify the optimal grid spacing of the vertex spatial pattern. By describing the input requirements of this algorithm, we were able to reuse this processing component in the STF to DTED Converter. The STF to DTED Converter was thus augmented with XTCRS descriptions of the different input data organizations it can handle. If the input data does not contain a grid but matches the organization expected by the `Polygonal Terrain to Gridded Terrain` algorithm, it is forwarded to this component so that its output can then be routed back for conversion to the DTED format.



```
processing_component
  version = 4.1
  name = Polygon_Terrain_to_Grid
  input_requirements
    requirement ...
      name = rqmt1
    requirement ...
      name = rqmt2
  output_description
    requirement ...
    requirement ...
  output_context
    parameter
      name = merge_into_one_grid
      value = true
    parameter
      name = grid_spacing
    parameter
      name = property_grid
```

Figure 1 - Sample processing component template

3.2 Converting environmental data

Most environmental data file formats have been developed to address the needs of a particular application area. In some cases, there are overlaps between application areas or, due to technological innovations, new requirements arise that trigger the development of additional file formats. For example, both the OpenFlight and COLLADA formats store 3D graphics information, but OpenFlight has an emphasis on 3D visualization while COLLADA is geared toward the interchange of three-dimensional (3D) models between 3D modeling tools.

Some file formats allow for extensions that allow them to be used beyond their original intent. For example, the Tagged Image File Format (TIFF) was designed to store raster images such as logos, scanned documents, and pictures. However, due to the format's popularity and flexibility, the GeoTIFF specification was developed to allow the Geographic Information System (GIS) community to store geographic information such as aerial imagery and elevation models. GeoTIFF is not a separate file format (from TIFF), but rather it adds a set of specifications that describe the rules for storing those specific raster data types in a TIFF file. In particular, the specification defines a set of tags used to geo-reference the raster data stored in the file.

While the variety of file formats is not a problem in itself (since it is driven by the requirements of specific communities), it does create significant and complex problems when the data from different file formats needs to be assembled into a coherent data set. As described earlier, the libraries used to access the data in these file formats define their own data models. However, because the scope of each file format is limited, the respective data models of these libraries are not suitable for directly creating relationships between the data elements in the different formats. Hence, the application designer is faced with the task of creating an additional data model (and likely a related custom file format) to support the establishment and storage of these relationships.

An alternative to creating these custom data models is to use a data model that not only can represent the data stored in the different file formats, but also allows for the specification of relationships between these data elements. This is the approach taken in SEDRIS. The SEDRIS DRM has the capability to represent the data contained in popular environmental data formats while also providing the relationships between the DRM objects that are used to represent the data in those formats. For example, in the DRM, a point feature can have an association relationship to the geometry description of that same feature; or an elevation grid representing a surface can have an association to the polygonal representation of that surface.

While having a comprehensive data model such as the DRM is critical, it would not be sufficient without data in this data model. Tools to convert between common environmental data file formats to and from the STF file format have been developed over the years, most of which are available freely from the SEDRIS web site. Some of the popular file formats currently supported or in the works include CTDB, DTED, GeoTIFF, Gridded Binary (GRIB), OpenFlight, Shapefile, and Vector Product Format (VPF).

The conversion of environmental data from a file format with a limited data model into the SEDRIS DRM is generally a straightforward task. However, the reverse (the "STF to other format" side of the equation) is not always as easy. Part of the complication is due to the polymorphic representations that the DRM allows. Since most of the "other format to STF" converters are written by different developers and also represent that specific format's unique data model, the DRM hierarchy and constructs that are most appropriate to the task or are familiar to the developer are used. Therefore, the design of a converter that can accept any STF as input and converts it to another format is potentially faced with a variety of DRM organizations at its input.

As described earlier, often when developers are faced with data that may be available in a variety of representations, they generally design for the most common or preferred representations (at least initially). The developers of the "STF to other format" converters are no different and have sometimes followed a similar approach. Therefore, some of these converters are designed to handle specific representations and are unable to deal with all the DRM organizations that may be generated by the other converters.

To address the limitations in some of the "STF to other format" converters, we use the same technique described earlier for processing environmental data. That is, templates for describing the input requirements of these converters are being developed, and mechanisms are being added to the converters to search for processing components when the data is not in a form that can be processed directly. In this manner, the converters can be more explicit about the organization of data they support, and suitable processing algorithms can be developed to increase their flexibility.

Another benefit of describing the input requirements of the conversion tools is that developers producing SEDRIS transmittals in the STF format can use these requirements to automatically generate compatible DRM organizations. This in turn increases the availability of data in a common data model and organization that can be converted into other environmental data file formats.

4. Conclusion and Future Work

The unified approach described in this paper provides a framework to not only describe the environmental data, but also the requirements of the algorithms that work on such data. Using the SEDRIS DRM as a common data model allows for a reduction of the problem space produced by the variety of data models exposed by the different file format libraries.

A benefit of the technique is that developers can create components specific to their level of expertise in a particular aspect of environmental data processing, and these components can be clearly described for reuse by other developers. Using these techniques, we were able to reuse processing algorithms and components currently used in the SEDRIS file format converters. In addition, by clearly describing the data organizations handled by the processing algorithms used in these converters, the understanding of their limitations is also increased so that they can be further improved.

Future work will formalize the specification of environmental data processing components and provide the means for the creation of a searchable catalog of algorithms and processing components matching specific input/output requirements.

5. References

[1] DTED (Digital Terrain Elevation Data) Specification, http://dds.cr.usgs.gov/srtm/version2_1/Documentation/MIL-PDF-89020B.pdf

[2] GeoTIFF Specification, http://www.remotesensing.org/geotiff/spec/contents.html

[3] XML (Extensible Markup Language), http://www.w3.org/TR/REC-xml/

[4] GML (Geography Markup Language) ISO 19136:2007, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32554

[5] COLLADA, http://collada.org

[6] OpenFlight Specification, http://www.presagis.com/products_services/standards/openflight

[7] ESRI Shapefile Technical Description, http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[8] SEDRIS Transmittal Format Binary Encoding, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39412

[9] EDCS (Environmental Data Coding Specification), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30810

[10] NFDD (NSG Feature Data Dictionary), http://www.gwg.nga.mil/documents/asfe/NFDD_v1.8.pdf

[11] SEDRIS, http://www.sedris.org

[12] EDCS Registry, http://edcsreg.sedris.org

[13] SRM (Spatial Reference Model), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54166

[14] SEDRIS Standards, http://standards.sedris.org

[15] J. Campos, G. Hull, F. Mamaghani: "TCRS – A Methodology and Tool Set for Specifying Data Content", Spring SIW 2004.

[16] SEDRIS Tools, http://tools.sedris.org

[17] J. Campos: "How to Produce & Consume Transmittals", SEDRIS Technology Conference, January 2004.

Author Biographies

WILLIAM W. FERRELL is a software developer at Abamis IT Solutions. Mr. Ferrell has developed software for a variety of application domains, including simulation systems, GIS data analysis, hardware control systems, and mobile devices. Mr. Ferrell is a member of the SEDRIS core team, where he is a key maintainer and developer of the SEDRIS SDKs and tools. He is a member of IEEE and Association for Computing Machinery (ACM).

WARREN MACCHI is the president of Abamis IT Solutions, an Orlando, Fla., corporation that specializes in environmental and geographical information data analysis, conversion, and modeling services. Mr. Macchi has been developing software for modeling and simulation, 3D visualization, and environmental data processing for almost 15 years. He is also a core team member of the SEDRIS program, where he contributes in the development of the SEDRIS standards and software development kits. Mr. Macchi is a member of IEEE, ACM, and Simulation Interoperability Standards Organization (SISO), and holds a B.S. and M.S. in computer science from the University of Central Florida.

ANTHONY A. BARRESI is a developer and creative designer at Abamis IT Solutions. Mr. Barresi develops software and user interfaces for interactive systems used in desktop and mobile applications. He also develops 3D models and virtual environments to be used in simulation and gaming and entertainment systems. Mr. Barresi holds a B.A. in digital media Internet and interactive systems from the University of Central Florida.

FARID MAMAGHANI has worked in the field of interactive networked simulation since 1984. He is one of the original designers of the Defense Advanced Research Projects Agency (DARPA) Simulation Networking (SIMNET) computer image generation system and, over the years, has worked as systems engineer and project manager on a variety of networked simulation projects. He is one of the people responsible for the establishment of the SEDRIS project, and provides management oversight and technical direction for the project. He is also involved in a number of other simulation and training programs and has extensive background in the design, development, and management of large software and simulation projects.