XML as an Enabler for Metadata in Scientific Dataset Processing

by George K. Thiruvathukal

Abstract

This white paper describes the potential for XML to facilitate the processing of scientific data sets. XML, the Extensible Markup Language, is emerging rapidly as a framework for information processing, interchange, and presentation. Scientific data sets are represented typically as flat file structures, primarily due to performance considerations. In this document, I will describe how the two can be brought together to enable efficient processing while simultaneously supporting the rich processing potential of XML. What kind of rich processing potential is enabled?

Introduction

What is the scientific computing process? As this writer sees it, scientific computing is characterized by one or more applications that create (typically) large data sets. These applications are connected together in some pipeline, wherein the data sets from one phase are processed in a subsequent phase.

The pipeline itself is used in an iterative fashion, especially during the software development phase. In the existing MDMS work, we refer to the notion of a run, which is a particular instance of execution. In this paper, I refer to the notion of a run as a single pass through the pipeline. This allows every step of the pipeline to introduce, modify, or remove data sets with clean bookkeeping. More on this in the design section.

I will expand this introduction when working on a paper.

Filesystem Based Metadata

The approach I describe here is totally based on the file system. I believe this will result in a much simpler metadata system that also has the potential to be used in a distributed context and even behind firewalls, etc. This is very much a tangent, so I will come back to this thought later.

As I see it, metadata is data about the data. If you look at every environment where metadata is used, it is always kept as close to the actual data as possible. For example, databases use table metadata as part of the table definition itself, etc. I believe our approach to keeping metadata in the database is a useful starting point but really does not lead to the rich experience that is possible if data and its metadata are kept together.

Abstract Filing: An Abstract Data Type for Managing Files

Long ago, the field of computer science embraced the notion of abstract data types as a useful mechanism for introducing new features into a programming language. The ADT approach is found in many of the popular scientific libraries, such as MPI and MPIO. I believe that our metadata system must be available as an abstract data type that can be used transparently by applications.

What facilities should be provided:

1. Interface to the workflow model.
2. Ability to create datasets transparently. Users should be able to create one FileSet with whatever name desired and never directly invent names for the actual data sets created in loops, or wherever desired.
3. Ability to discover datasets created from other phases within a run and process them, accordingly.
4. Ability to tie program information (e.g. iteration count, dimensions, etc.) to the metadata itself. While processing a FileSet, metadata can be written at any time, and it can follow any hierarchy desired by the user.

The FileSet concept initially will be specific to standard I/O. I want to demonstrate these principles, initially, in a non-parallel context. I see no reason why the ideas cannot be generalized to parallel I/O, and this is more than likely a project for undergraduate/graduate research students (Greg, David?)

The FileSet Abstract Data Type

FileSet is the abstract data type name. FILE * is a pointer to a standard file.

Creating a File Set:

int newFileSet(FileSet *fs, char *name)

Create a FileSet. The name, supplied by the user, will be used to create a filesystem resident directory named name, if the directory does not exist.

Error Codes:
FS_ERROR_BAD_METADATA - User has specified a directory name that exists but does not contain valid metadata.

FS_ERROR_NO_METADATA - User has specified a directory naem that exists but does not contain metadata.

FS_ERROR_NO_ACCESS - Permission problem.

int nextRun(FileSet *fs)

Bump the run count. This updates the metadata file (Execution.xml). Any phase in the pipeline may call this. Once done, however, any subsequent phase will be assuming the current run ID.

Error Codes:
None known.

int nextPhase(char* phaseName)

This indicates what phase of the pipeline we are presently in. The name phaseName must be present in Phases.xml.

int recordEvent(FileSet *fs, char *description, char *keys[], char *values[])

Events may be "recorded" at any time. The information here can be used for any purpose. When written, the event information is recorded as a collection of key/value pairs. Event information is specific to a phase, run, and possibly a specific iteration.

Why would one want this in the metadata? Possibly for debugging. Possibly to record a feature of interest (e.g. "found new maximum pressure value"; keys and values would contain any parameters of interest.)

FILE* getDataSet(char* name)

Based on the current <run, context, iteration>, get a FILE* for an existing file.

Errors:
FS_ERROR_NOT_EXISTS

FILE* newDataSet(char *name)

Create a new file name and return a handle to the open file. Note that this function can be called any number of times to create a file name. The FileSet ADT will ensure that the file is given a unique (mangled) name on disk.

Errors:
FS_ERROR_ALREADY_EXISTS

void newContext(char *contextName)

Create a new context for iteration. This can be useful if I/O is performed in loops where you might want to write files in an inner loop.

void revertToPreviousContext()

Return to the previous context. This method will result in an error if called at the top level context.

void revertToNamedContext(char *contextName)

Return to a named context. This can be useful if you have nested contexts and have to break out of a deeply nested loop and need to return to an "outer" context.

void nextIteration(FileSet fs, char *userIterationVariableName, int userIterationCount)

Advance to the next iteration. The user may supply *userIterationCount* to and userIterationVariableName to provide details for the actual loop index name and its value. Internally, the XML document will automatically number the iterations and name them after the context.

A Glimpse of the Metadata

A FileSet is implemented in a Unix/Windows Directory. The directory contains two subdirectories, which will have further subdirectory structure broken out as needed.

```
FileSet XYZ
  - .md
                Execution.xml
                Phases.xml
                Run1.xml
                Run2.xml
                etc.

  - .d
                This file will contain mangled file and directory names that are generated
                as desired by users.
```

Execution.xml

```
<Execution>
<Run id="1" date="07-02-2001" run_file="Run1.xml"/>
<Run id="2" date="07-09-2001" run_file="Run2.xml"/>
</Execution>
```

Run1.xml

```
<Run id="1">
 <Context name="top">
  <Iter val="0" user_index="i" user_index_value="1">
    <DataSet name="rho" file_name=".d/app_compute_1_rho_top_0"/>
  </Iter>
  <Iter val="1" user_index="i" user_index_value="2">
    <DataSet name="rho" file_name=".d/app_compute_1_rho_top_1"/>
  </Iter>

  ...
  <Iter val="4" user_index="i" user_index_value="5">
     <DataSet name="rho" file_name=".d/app_compute_1_rho_top_4"/>
     <DataSet name="rho_trace" file_name=".d/app_compute_1_rho_trace_top_5"/>
  </Iter>

  ...
 </Context>
</Run>
```

It is possible that a subsequent phase of execution changes the metadata. Suppose there is a phase that prepares JPEG versions of the "rho" data sets. Suppose that a second phase (another executable program, named render2D) modifies the metadata.

```
<Run id="1">
 <Context name="top">
  <Iter val="0" user_index="i" user_index_value="1">
     <DataSet name="rho" file_name=".d/app_compute_1_rho_top_0"/>
     <DataSet name="rho_jpeg" file_name=".d/app_render2D_1_rho_jpeg_top_0"/>
  </Iter>
  <Iter val="1" user_index="i" user_index_value="2">
     <DataSet name="rho" file_name=".d/app_compute_1_rho_top_1"/>
     <DataSet name="rho_jpeg" file_name=".d/app_render2D_1_rho_jpeg_top_1"/>
  </Iter>

  ...

  <Iter val="4" user_index="i" user_index_value="5">
     <DataSet name="rho" file_name=".d/app_compute_1_rho_top_4"/>
     <DataSet name="rho_jpeg" file_name=".d/app_render2D_1_rho_jpeg_top_4"/>
     <DataSet name="rho_trace" file_name=".d/app_compute_1_rho_trace_top_4"/>
  </Iter>

  ...
 </Context>
</Run>
```

Question: How to encode the phase name into the DataSet? I think phase=phase-name.