

Runtime Generation of Data Processors on Local User Computers

Jigarkumar Patel, Sergiu M. Dascalu, Frederick C. Harris, Jr.
Department of Computer Science & Engineering, University of Nevada
Reno, USA
{jspatel, dascalu, fred.harris}@cse.unr.edu

Abstract—Data interoperability in scientific research is a major challenge. Heterogeneous data file formats, data structure formats, and data storage schemas are prevalent in the scientific research community. This poses a great challenge for collaboration between researchers due to data interoperability issues. In our prior work, we have addressed these challenges by proposing a web-enabled approach for generating data processors initially intended for environmental science researchers. This paper takes the solution one step further, and describes the software that enables generating and running data processors on local user computers. The paper details the design of the software solution and presents the steps that the users need to follow for running processors for a variety of data conversion needs. The solution presented is powerful and flexible as it can be applied to a large variety of data conversion needs and to practically any type of data-intensive scientific research. It also facilitates collaboration as scientists can share their custom created data processors, thus increasing efficiency in research activities. Researchers can benefit from the software engineering principles used in creating and generating data processors, and can take advantage of the reusability of numerous data processors create by end users who do not necessarily have software development expertise.

Keywords—data processor; data conversion; software tool; data interoperability; research data

I. INTRODUCTION

Data interoperability is a major challenge across different disciplines. The decision of using specific data storage solutions, data file formats, and data structures is dependent on data sensors, data loggers, data collection and curation methods, research requirements, the experience and technical expertise of the research group involved, and institutional data policies. There have been several efforts made in the direction of standardizing data and metadata formats, for example [1, 2, 3]. However, the overhead of standardization sometime hinders the research process, so in many cases the scientists end up adhering to the file formats that best suit their needs. In addition, interoperability among models is a complex research challenge, stemming from the heterogeneity of the models themselves, the diversity of data types, structures, and formats used as input and output for models, the semantics of variables and data involved in the models, and the transformations necessary to transfer information from one model to another. Other aspects such as the diverse technology (e.g., programming languages, operating systems, APIs,

architectures) used in developing the simulation software that runs the models further contribute to the complexity of the model and data interoperability challenge [4]. Various solutions exist for dealing with model coupling [5, 6] and several major initiatives have been focused on addressing issues pertaining to model and data interoperability, including [7, 8, 9, 10]. An informative overview of such initiatives is presented in [11]. Nevertheless, due to the intrinsic complexity and variety of interoperability needs, many of such issues are still to be resolved. In short, the combination of all the factors mentioned above presents a substantial collaboration challenge for scientists involved in data-intensive research.

Funded by a Nevada-wide NSF EPSCoR grant [12] and by another NSF EPSCoR collaborative grant that involves teams from Idaho, Nevada, and New Mexico [13], the authors of this paper have tackled model and data interoperability on several directions – specifically, via a web-service workflow-based approach that has led to the creation of the Demeter framework [4, 14, 15], through a standalone solution that emphasizes visual object-based scenario definition, code generation, dataflow scenario execution, and 3D data visualization [16, 17], and by proposing a data processor-focused approach that aims to support user friendly, highly flexible definition and manipulation of environmental datasets (both grants mentioned above are for projects involving climate change research and education).

Pertaining to the last direction of work (centered on data processors), in our earlier paper [18] we have focused on data interoperability challenges. There, we discussed general issues related to data file formats, data filtering, and data scaling. In a more recent paper [19] we have expanded the description of our approach by presenting the methodological steps needed for data processor definition. Part of this, we have designed and described the toolkit we entitled WEDMIT (Web-Enabled Data and Model Interoperability Toolkit). This toolkit supports a new, innovative way of defining and processing datasets that allows users without computer expertise or skills to flexibly define new data structures, apply a variety of operations on them, create data processors, and run them to apply various transformations, including format conversions, on observational or simulated datasets. Although initially developed for supporting environmental science research, the flexibility and generality of WEDMIT makes it useful for any data-intensive area of research.

Based on the work described in [18] and [19], we present in this paper details of how a data processor is generated and runs on local user machines. While WEDMIT is a web-enabled software environment, parts of data processor definition and generation need to be performed on local computers, as explained later in the paper. Our proposed solution is significant because it is novel and contributes to addressing existing challenges pertaining to model and data interoperability without forcing scientists to make radical changes to their research practices. The proposed data processor generation approach fits well with existing research data and models and aims to handle a variety of time consuming and complex data processing tasks. Notably, it does not take more than few minutes of the users' time to generate executable software that automatically performs such tasks.

The paper in its remaining parts is organized as follows: Section 2 provides background information pertaining to model and data interoperability; Section 3 gives an overview of the steps needed to create a data processor using the WEDMIT approach; Section 4 provides detailed descriptions on generating and running data processors on local user computers using this approach; Section 5 discusses WEDMIT's advantages and its limitations; and Section 6 presents our concluding remarks and outlines several possible directions of future work.

II. BACKGROUND

Larger scientific problems are broken into smaller problems for mathematical and computational simplicity. These small problems are often represented as mathematical models, which are then translated into computational simulation models (or, in short, just models). To simulate larger systems and get an understanding of more complex phenomena, two or more models are often combined, a process which is known as model coupling. The model coupling can be parallel, where two or models run in parallel while interacting with each other, or sequential, where output from one model is fed as (part of) input into the another model. The key challenge in the entire process is the passing of data between the models. Scientific models are notoriously varied, being designed by many different researchers, and using many different data types, structures, file formats, scales, units, and semantics [4, 18].

Several solutions exist to address model and data interoperability challenges [5, 6, 7, 8, 9, 10, 11]. However, many of them are limited to providing a framework, environment, API, or set of standards and/or guidelines that require researchers not only good knowledge of their resources but also using these resource to develop simulation software from the scratch. To the best of our knowledge, there is no current solution similar to WEDMIT, which is a web-enabled, flexible, and user-oriented approach and supporting toolkit for generating and running data processors and assisting model and data interoperability. The architecture of the WEDMIT web-based software was previously described in [18]. In this ongoing project, we have addressed so far web-enabled creation of data structures, data structure operations, data processors, and data processor definitions. The approach and its associated tool also supports user access to stored data processors and models, which makes it possible to persistently

save user generated content for later reuse and sharing with other users. While at this time WEDMIT supports data interoperability, in future releases it will aim to provide operational support for model coupling. Presenting the current state of the project, this paper is focused on how to generate and run data processors created on local user computers.

III. CREATING DATA PROCESSORS USING THE WEDMIT APPROACH

As discussed in previous sections, researchers use different data formats and file formats. The specific data and file formats needed in a given research activity are dictated by the simulation models that the researchers use. Researchers often use model coupling techniques to benefit from knowledge across multiple domains and/or disciplines. In the simpler version of a model coupling scenario (involving two models), the output of one model is used as input into the other model. This coupling process often involves some level of data massaging. Many models, without being part of the coupling process, also require data to be prepared in specific formats. To perform model simulations, researchers typically spend a lot of their valuable time to prepare (using available data) datasets in the format required by these simulations. Such tedious preparation activities could involve a combination of manual data filtering, some computer programming, and not infrequently some much needed help from others (particularly from people with good computer knowledge and/or software development skills). To avoid these repetitive error prone methods we have suggested the WEDMIT approach and developed its associated toolkit [18].

For the reader's convenience we summarily recap here the steps needed to create data processors with WEDMIT. The first step in the creation of a new data processor is to define a new *data structure*, which can be done by using several pre-defined data types and providing names for the data structure's *elements* (or *columns*). For example, a newly created data structure for a river hydrology is shown in Figure 1. Notably, the design of the new data structure is entirely up to the user (typically, a researcher), which provides great flexibility in terms of what datasets the user can work with. Usually, in WEDMIT's terminology a data structure describes the composition of an existing or future dataset (in our case, a river's hydrology dataset) or defines the organization of a model simulation's output. The second step in the process is to define *data operations* on data structures by means of providing filtering conditions (such as threshold or range comparisons) on columns, sorting the items in a column, applying pre-defined functions (such as math functions) on the columns, and splitting the output data structure in multiple files. This part of the process of defining a new data processor is shown in Figure 2. The final step in the process is to complete the definition of a *data processor* based on an available WEDMIT data structure and the data operation set associated with it. Part of a data processor's definition, in this step the user also needs to specify the processor's input file format and its desired output file format. This step is illustrated in Figure 3 where the user selected NetCDF as input file format and Text as output file format in a data processor designed to be applied on the newly created river hydrology dataset.





























	Data Type	Name	Description
1	Floating Point Number	Stage	 
2	Floating Point Number	Temperature	 
3	Floating Point Number	Surface Temperature	 
4	Floating Point Number	Bottom Temperature	 
5	Floating Point Number	Salinity	 
6	Floating Point Number	Surface Salinity	 
7	Floating Point Number	Bottom Salinity	 
8	Floating Point Number	Discharge	 
9	Floating Point Number	Filtered Discharge	 
10	Floating Point Number	Specific Conductance	 
11	Floating Point Number	Surface Specific Conductance	 
12	Floating Point Number	Bottom Specific Conductance	 
13	Floating Point Number	Turbidity	 
14	Date/Time	Measurement DateTime	 


Figure 1. Example of river hydrology data structure.

1. Filter

Column:

Condition:

Value:

 Save









Stage	>	-2	 
Temperature	>=	2	 
Temperature	<=	45	 
Measurement DateTime	<=	2010/01/01	 
Measurement DateTime	>=	2001/01/01	 

Figure 2. Sample filtering data operations.

Input File Format	Data Structure Operation
NetCDF Files	Data Quality Assurance
Input File Format	<input type="text" value="NetCDF Files (nc)"/>
Operation	<input type="text" value="Data Quality Assurance"/>
Output File Format	<input type="text" value="Text Files (txt)"/>
Comment	<input type="text"/>

Figure 3. Example of data processor definition.

As noted in [18], a data processor applies desired transformations on an input data file of a specified file format (whose data structure and set of operations are part of the processor’s definition). By applying these transformations an output file is generated in the format also specified in the definition of the processor. Thus, the data processor concept is powerful and flexible, as it can be used in many kinds of data processing activities, needed in a broad range of research and development areas.

IV. CREATING DATA PROCESSORS USING THE WEDMIT APPROACH

The data processor creation (or design) process is entirely based on using the WEDMIT web application, whereas independent from the web the data processor itself is generated and run on the users’ local machines. This entire process, involving code generation and execution, is illustrated in Figure 4.

The first step in the process is to download a generic data processor which, in WEDMIT terms, is an executable file with an associated set of class libraries that need to reside on the user’s computer. This generic data processor is available on the WEDMIT web application download area, and the users need to download and save it on their computer only once. Based on the user designed data processor, which as shown in Section 3 includes a data structure, data structure operation set, and data processor input/output file formats, WEDMIT generates an XML-based data processor definition (or, in short, a data processor). As shown in Figure 4, the next step in the process is to download on the user’s local computer the XML-based data processor definition of the processor. This definition is actually a serialized representation of a data processor object that contains all the information about data structure columns, data structure operations, and input/output file formats (an example is shown in Figure 5). The users are required to place the downloaded XML file under the XML subdirectory of the generic data processor folder. The next step in the process is to update the generic data processor configuration file as illustrated in Figure 6. The generic data processor encapsulates the running of the actual data processor. It is designed to run any user created data processor, hence the user must specify the data processor that he or she wants to run. The name of the data processor is the name of the data processor definition XML file without file extension. For code generation purposes, the generic processor will look under the XML subdirectory to find the user created data processors at runtime. Users also must specify the full path to the input file and the output file with appropriate file extensions. It is worth noting that the configuration can be easily extended to a desktop GUI. However, to ensure portability for the time being we decided not to use any GUI framework that would be bound to a specific operating system. The use of a cross platform GUI framework would require separate installation on each required machine, whereas the XML-based configuration that we use enables cross-platform compatibility for the generic processor, without requiring any additional set of libraries.

After the above configurations are completed, the generic processor is ready for use, and the users can invoke the generic processor from their console. The generic processor runtime

sub-process is highlighted in Figure 4 using grey rectangles and the output of running the generic processor is shown in Figure 7. Procedurally, the generic processor first reads the configuration file and loads the data processor's XML file. At this time, the XML-based data processor is de-serialized into a programming object. Classes for the input data structure, output data structure, and data processor are generated next (an example of an automatically generated class for the input data processor is shown in Figure 8, while Figure 9 shows the data processor class generated as a part of the same sub-process). Because this is vital in reading and writing data structures, the auto generated class for the data structure incorporates the column order, as shown in Figure 8 (although the programming object itself has no notion of class fields order). The auto generated data processor class includes all the code required to run the user created data processor as well as several comments to guide the user, as displayed in Figure 9. Next, the generic processor compiles the generated classes into a dynamic class library. This class library essentially covers start-to-end data processor requirements for the user. This compiled DLL is loaded into the generic data processor at runtime, and it carries out the user created data processor operations. At the end of the processor's run the user can retrieve the resultant file using the specified output file path.

In our case, the river hydrology dataset with the structure shown in Figure 1 will be processed using the operations presented in Figure 2 and following the specifications of the data processor described in Figure 3. The result will be, as indicated in the processor's definition, a Text file that contains the processed river hydrology data.

Notably, the class generation and compilation process is not required every time. To speed up the process, once the generic processor has the dynamically compiled library available, it will skip the code generation and compilation part in subsequent runs of the same processor. It is also plausible that the user might want to make some changes to a data processor design and generate a new XML definition file for it. In this scenario, the generic processor must regenerate all the classes associated with that particular data processor. To help with this, there is a configurable option for the generic data processor, as shown in Figure 5. Certainly, it is quite possible that the data processing tools provided by WEDMIT might not be able to fulfill a user's all specific requirements. In such cases the user has the possibility to change source code of the auto generated data processor classes. For this reason and to take advantage of object-oriented design principles, the generated classes are separated in multiple files where the users need to work with only a single source code file. The generic processor will not overwrite this user modified source code file, however it will automatically recompile it and generate a new DLL class library. The overall design and main relationships among the classes of the generic data processor code are shown in Figure 10. This class diagram doesn't include dynamically generated data processor classes based on the XML-definition of the user-designed data processor.

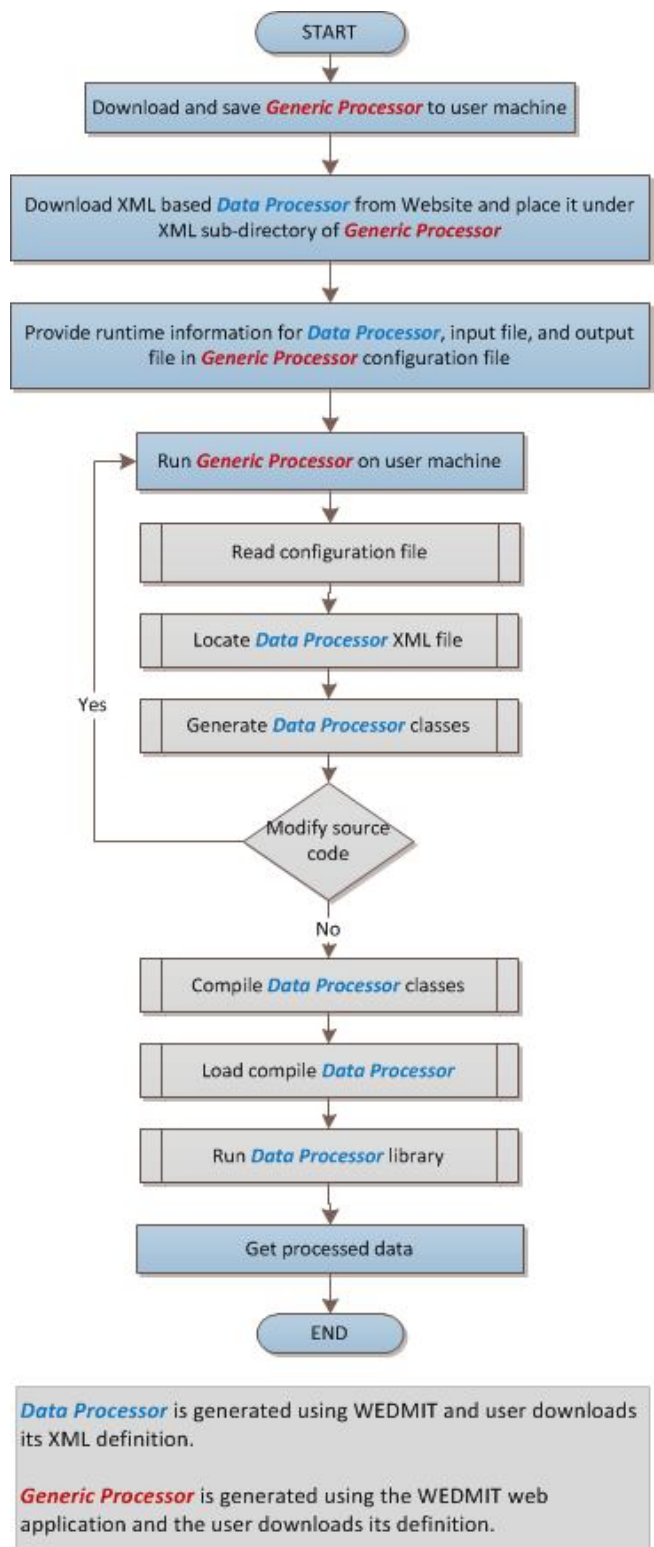


Figure 4. Using data processors on local user computers.

```

<?xml version="1.0" encoding="utf-16"?>
<SerialProcessor
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="CodeGenerator">
  <Name>Quality Control Processing</Name>
  <Input>
    <Name>NetCDF Files</Name>
    <Extension>nc</Extension>
    <IsGridded>>false</IsGridded>
  </Input>
  <Output>
    <Name>Text Files</Name>
    <Extension>txt</Extension>
    <IsGridded>>false</IsGridded>
  </Output>
  <Structure>
    <Name>River Hydrology Data</Name>
    <VariableName>RiverHydrologyData</VariableName>
    <Columns>
      <Column>
        <Name>Stage</Name>
        <VariableName>Stage</VariableName>
        <Index>1</Index>
        <DataType>
          <Name>Floating Point Number</Name>
          <SystemName>float</SystemName>
        </DataType>
      </Column>
      ...
    </Columns>
  </Structure>
  <Operations>
    <FilterOperations>
      <FilterOperation>
        <ColumnIndex>1</ColumnIndex>
        <Condition>&gt;</Condition>
        <CompareValue>-2</CompareValue>
      </FilterOperation>
      ...
    </FilterOperations>
    <SortOperations>
      <SoftOperation>
        <ColumnIndex>14</ColumnIndex>
        <Ascending>>false</Ascending>
      </SoftOperation>
    </SortOperations>
    <FunctionOperations />
    <IgnoreColumnOperations />
  </Operations>
  <ID>5</ID>
</SerialProcessor>

```

Figure 5. Sample XML data processor definition.

V. DISCUSSION

The proposed WEDMIT approach offers a new solution for dealing with data interoperability issues. The first part of the solution is a web-based application, which makes it possible to access the toolkit from different computers located practically anywhere. The second part relies on the concept of generic data processor, which in essence is a software resource that has to reside and run on a user's local computer. The generic processor has been designed and developed using the Mono/.Net framework, thus making it portable enough to run on computers operating on Windows, Linux, or Mac. The generic processor was designed with extensibility as the

```

<configuration>
  <!-- Don't modify runtime section -->
  <runtime>
    <assemblyBinding xmlns="urn:schemas-
microsoft-com:asm.v1">
      <probing privatePath="libs"/>
    </assemblyBinding>
  </runtime>

  <!-- Users can modify following sections -->
  <!-- Only change value fields. Don't change key
field. All values must be quoted -->
  <appSettings>
    <!-- Name of the processor without XML extension.
Place XML file associated with processor in XML
directory. -->
    <add key="Processor" value="Processor4"/>
    <!-- If you want to regenerate code because of
change in XML file. -->
    <add key="GenerateCode" value="true"/>
    <!-- If you want to recompile the code because of
changes you made to class files. It will also
recompile the library-->
    <add key="RecompileCode" value="true"/>
    <!-- Full path to input file with extension
csv,nc,xml,xlsx,txt -->
    <add key="InputFilePath" value="C:\input.nc "/>
    <!-- Full path to output file with extension
csv,nc,xml,xlsx,txt-->
    <add key="OutputFilePath" value="C:\output.txt "/>
    <!-- <add key="OutputFilePath"
value="C:\VMC\DataProcessor\Sample Data\Nevada
Portal Sample Result.xml"/> -->
  </appSettings>
</configuration>

```

Figure 6. Generic data processor XML-based configuration file.

```

C:\VMC\DataProcessor\DataProcessor\bin\Debug>DataProce
Starting application...
Start time: 9:22:56 PM
Reading configuration file...Done.
Generating source code..
Generating input class code..Done.
Generating output class code..Done.
Generating processor class code..Done.
Generating mapping class code..Done.
Writing Input class code..Done.
Writing Output class code..Done.
Writing mapping class code..Done.
Writing user custom processor code..Done

```

Figure 7. Console output for the generic data processor run.

primary goal in mind. As such, new file formats can be easily added by implementing a base interface with a small number of methods. It also provides a set of strongly typed data object classes for data processing, which reduces significantly the code (that is, the programming effort) needed to load and populate data objects. Researchers necessitating complex data processor solutions can leverage the ready-to-use code created by others via the WEDMIT approach, where most of the “heavy lifting” (intensive programming work) has been already done for them. In such cases, the researchers are encouraged to modify existing data processors to fit their needs. Saving a considerable amount of time, all changes are automatically compiled and incorporated into the newly generated executable files. The only requirements for implementing the changes are a basic knowledge of programming and a text editor for source code modification.

Even though WEDMIT has several notable benefits, our solution still has several drawbacks. Specifically, at this time it

cannot handle complex data processor scenarios via the web application. Furthermore, it currently has a limited set of pre-defined functions and the filtering functions themselves are limited to comparisons based on range and threshold conditions. Moreover, the researchers cannot incorporate their existing programs with the WEDMIT-generated code unless they were developed using Mono/.Net framework.

VI. CONCLUSIONS AND FUTURE WORK

Data interoperability is a complex challenge and researchers would certainly welcome any help they can get. Researchers with non-technical background might face even greater challenges dealing with interoperability issues. WEDMIT, our proposed approach for helping with data interoperability, provides an easy to use, user-centric solution aimed at addressing most of related software development challenges with minimal programming needs. This is a new approach with great potential for applications across many areas of scientific

```
//Auto generated code
using System;
using WEDMIT.Formats;
using WEDMIT.Processor;
namespace UserProcessor{

public partial class Input5 : Record{

public Input5 (){//Constructor

this.ColumnIndex<Input5>(i=>i.Stage);
this.ColumnIndex<Input5>(i=>i.Temperature);
this.ColumnIndex<Input5>(i=>i.SurfaceTemperature)
;
this.ColumnIndex<Input5>(i=>i.BottomTemperature);
this.ColumnIndex<Input5>(i=>i.Salinity);
this.ColumnIndex<Input5>(i=>i.SurfaceSalinity);
this.ColumnIndex<Input5>(i=>i.BottomSalinity);
this.ColumnIndex<Input5>(i=>i.Discharge);
this.ColumnIndex<Input5>(i=>i.FilteredDischarge);
this.ColumnIndex<Input5>(i=>i.SpecificConductance
);
this.ColumnIndex<Input5>(i=>i.SurfaceSpecificCondu
ctance);
this.ColumnIndex<Input5>(i=>i.BottomSpecificCondu
ctance);
this.ColumnIndex<Input5>(i=>i.Turbidity);
this.ColumnIndex<Input5>(i=>i.MeasurementDateTime
); } //End of constructor

public float Stage {get;set;}
public float Temperature {get;set;}
public float SurfaceTemperature {get;set;}
public float BottomTemperature {get;set;}
public float Salinity {get;set;}
public float SurfaceSalinity {get;set;}
public float BottomSalinity {get;set;}
public float Discharge {get;set;}
public float FilteredDischarge {get;set;}
public float SpecificConductance {get;set;}
public float SurfaceSpecificConductance
{get;set;}
public float BottomSpecificConductance {get;set;}
public float Turbidity {get;set;}
public DateTime MeasurementDateTime {get;set;}
}}
```

Figure 8. Automatically generated input data structure class.

```
//User is allowed to modify this code.
using System;
using System.Collections.Generic;
using System.Linq;
using WEDMIT.Formats;
using WEDMIT.Processor;
namespace UserProcessor
{
public partial class Processor5 :
DefaultProcessor<Input5,Output5,#InputDataFormat#
<Input5>,#OutputDataFormat#<Output5>> ,
IProcessor<Input5,Output5,#InputDataFormat#<Input
5>,#OutputDataFormat#<Output5>> {

public Processor5():base(){
//Set some of the values that might be unique to
your requirement
}

public override void Process (){
/* If you want to override Process then comment
following line and write your own code for
process. * */
base.Process (); //Leave this line uncommented
for default process
}

public override Input5 MapInputRecord (object[]
values){ /* In the case where you want to
write your own code to map values to object then
write your code here. This function is called
only when you have Default Process is used. */
return base.MapInputRecord (values);
}

public override bool FilterOperation(Input5
Record)
{/* Check for filter conditions. Return false if
it doesn't meet filtering criteria else return
true by default. You can add more conditions as
per your need without affecting processor. By
default all conditions are combined with &&(AND)
operator. You can combine them as per your need.
*/
bool condition1 = (Record.Stage > -2);
bool condition2 = (Record.Temperature >= 2);
bool condition3 = (Record.Temperature <= 45);
bool condition4 = (Record.MeasurementDateTime <=
DateTime.Parse("2010/01/01"));
bool condition5 = (Record.MeasurementDateTime >=
DateTime.Parse("2001/01/01"));
if(!(condition1 && condition2 && condition3 &&
condition4 && condition5)) return false;
return true;
}

public override void SortOperation()
{/* Sort output dataset/records. By default
records are sorted descending hence use of
OrderByDescending. If you want to order Ascending
then just user OrderBy method.*/

OService.NewDataSet (OService.DataSet.Order
ByDescending (c=>c.MeasurementDateTime) .ToL
ist());
}}}
```

Figure 9. Automatically generated input data processor class.

research. As a web-enabled application, WEDMIT supports reusability and fosters collaboration – the research community can certainly benefit from sharing a variety of user tailored data processors.

While WEDMIT provides an effective solution for dealing with certain data interoperability challenges, it still has significant room for improvement. For example, currently the users must download the generic processor and run it on their local computers, but in the future we would like to enable researchers to run it on the web (from a web server). This is currently under consideration, the main aspects under scrutiny being potential security issues and potentially very large data files that might inundate the server. Part of our future work it to also deploy this web-based application on the Nevada Climate Change Portal [20, 21, 22], and thus make it easily accessible to the public. Furthermore, adding an ability to read/write new file formats is currently available to web developers only. In the future, a useful extension would be to allow web-application users to contribute source code or compiled library files to support new file formats.

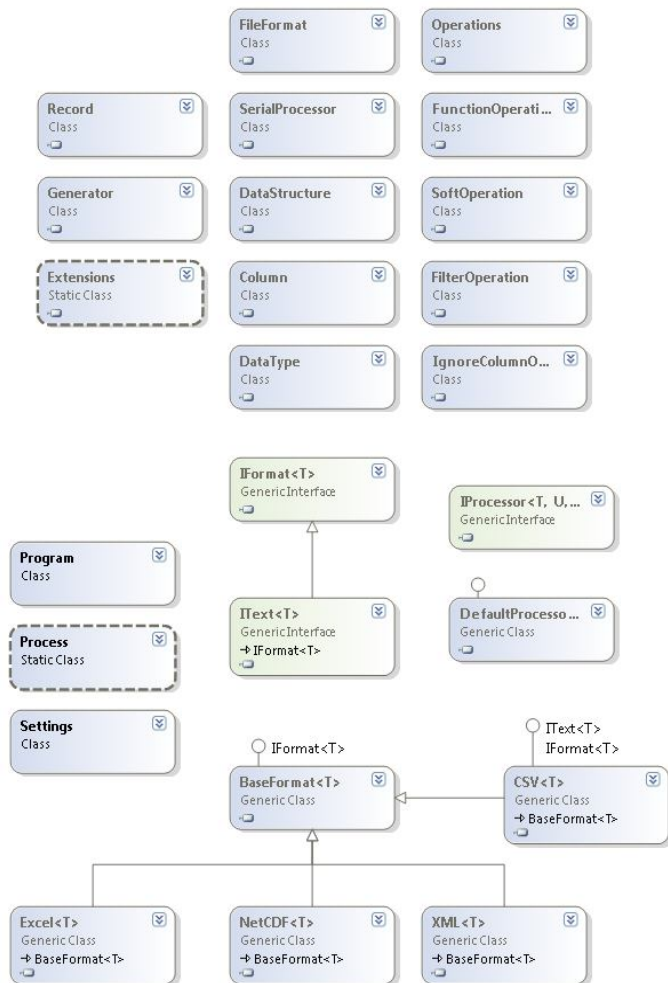


Figure 10. Class diagram of the generic data processor.

ACKNOWLEDGMENT

This work was made possible through the support provided by the National Science Foundation under Cooperative Agreements No. EPS-0814372 and No. EPS-0919123.

REFERENCES

- [1] ISO 19115, Geographic Information Metadata. Accessed Feb. 28, 2013 at http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020
- [2] Dublin Core Metadata Initiative. Accessed February 28, 2013 at <http://dublincore.org/>
- [3] CSDGM – Content Standard for Digital Geospatial Metadata. Accessed February 22, 2018 at <http://www.fgdc.gov/metadata/csdgm/>
- [4] S. Dascalu, E. Fritzinger, S. Okamoto and F.C. Harris, Jr., “Towards a software framework for model interoperability,” in Proceedings of the 9th IEEE International Conf. on Industrial Informatics (INDIN 2011), Lisbon, Portugal, IEEE Computer Society, July 2011, pp. 705-710.
- [5] T. Bulatewicz, Support for Model Coupling: An Interface-based Approach. PhD thesis, University of Oregon, 2006.
- [6] T. Bulatewicz and J.E. Cuny, “A domain-specific language for model coupling,” in Proceedings of the 38th Winter Simulation Conference (WSC-06), 2006, pp. 1091-1100.
- [7] OpenMI. Accessed February 28, 2013 at <http://www.openmi.org/>
- [8] CSDMS - Community Surface Dynamics Modeling System. Accessed Feb. 28, 2013 at http://csdms.colorado.edu/wiki/Main_Page
- [9] ESMF – Earth System Modeling Framework. Accessed February 28, 2013 at <http://www.earthsystemmodeling.org/>
- [10] The Kepler Project. Accessed February 28, 2013 at <https://kepler-project.org>
- [11] H.R.A. Jagers, “Linking data, models, and tools: an overview,” in Proceedings of the International Congress on Environmental Modeling and Software (IEMSS-2010), Ottawa, Canada, July 2010.
- [12] Nevada Climate Change Portal (NCCP)/About the Project/Funding. NSF EPSCoR Coop. Agr. No. EPS-0814372. Accessed Feb. 15, 2013 at <http://sensor.nevada.edu/NCCP/The%20Project/Funding.aspx>
- [13] Collaborative Research: Cyberinfrastructure Developments for the Western Consortium of Idaho, Nevada, and New Mexico. NSF EPSCoR Cooperative Agreement No. EPS-0919123. Accessed Feb. 15, 2013 at <http://nsf.gov/awardsearch/advancedSearchResult?PIOrganization=Nevada%20System%20of%20Higher%20Education&>
- [14] E. Fritzinger, S. Dascalu, D.P. Ames, K. Benedict, I. Gibbs, M.J. McMahon Jr., and F. C. Harris Jr., “The Demeter framework for model and data interoperability,” in Proceedings of the International Congress on Environmental Modeling and Software (IEMSS-2012), Leipzig, Germany, July 2012, pp. 1535-1543.
- [15] S. Okamoto, E. Fritzinger, S. Dascalu, F.C. Harris Jr., S. Latifi, and M.J. McMahon Jr., “Towards an intelligent software tool for enhanced model interoperability in climate change research,” in Proceedings of the World Automation Congress (WAC-2010), Kobe, Japan, September 2010, IEEE Computer Society, pp. 1/1-6.
- [16] S. Okamoto, R.V. Hoang, S.M. Dascalu, F.C. Harris Jr., and N. Belkhatir, “SUNPRISM: An approach and software tools for collaborative climate change research,” in Proceedings of the 13th Intl. Conference on Collaboration Technologies & Systems (CTS-2012), Denver, CO, May 2012, pp. 583-590.
- [17] S. Okamoto, SUNPRISM: A Software Framework for Climate Change Research. PhD thesis, University of Nevada, Reno, 2011.
- [18] J. Patel, S. Okamoto, S.M. Dascalu, and F.C. Harris Jr., “Web-enabled toolkit for data interoperability support,” in Proceedings of the 21th International Conference on Software Engineering and Data Engineering (SEDE-2012), Los Angeles, CA, June 2012, pp. 161-166.
- [19] J. Patel, S. Okamoto, S.M. Dascalu, and F.C. Harris Jr., “A Web-enabled approach for generating data processors,” in Proceedings of the International Conference on Information Technology: New Generations (ITNG-2013), Las Vegas, NV, April 2013, in press.

- [20] The Nevada Climate Change Portal (NCCP). Accessed February 15, 2013 at <http://sensor.nevada.edu/NCCP/>
- [21] S. Dascalu, "Imagine a million file cabinets of climate data: The Nevada Climate Change Data Portal," invited talk, the Nevada Climate Change Seminar Series. University of Nevada, Las Vegas, September 7, 2011. Available at http://digitalscholarship.unlv.edu/climate_change/6/
- [22] M.J. McMahon Jr., S.M. Dascalu, F.C. Harris Jr., S. Strachan, and F. Biondi, "Architecting climate change data infrastructure for Nevada," in Advanced Information Systems Engineering Workshops CAISE-2011, Lecture Notes in Business Information Processing, LNBIP-83, C. Salinesi and O. Pastor, Eds. Springer, 2011, pp. 354-365.