INL/CON-12-26599
PREPRINT

A Proposed Data Fusion Architecture for Micro-Zone Analysis and Data Mining

5th International Symposium on Resilient Control Systems

Kevin McCarty
Milos Manic

August 2012

The INL is a
U.S. Department of Energy
National Laboratory
operated by
Battelle Energy Alliance

A Proposed Data Fusion Architecture for Micro-Zone Analysis and Data Mining

Kevin McCarty
University of Idaho
Idaho Falls, Idaho USA
kmccarty@ieee.org

Milos Manic
University of Idaho
Idaho Falls, Idaho USA
misko@uidaho.edu

***Abstract* – Micro-zone analysis involves use of data fusion and data mining techniques in order to understand the relative impact of many different variables. Data Fusion requires the ability to combine or "fuse" date from multiple data sources. Data mining involves the application of sophisticated algorithms such as Neural Networks and Decision Trees, to describe micro-zone behavior and predict future values based upon past values. One of the difficulties encountered in developing generic time series or other data mining techniques for micro-zone analysis is the wide variability of the data sets available for analysis. This presents challenges all the way from the data gathering stage to results presentation. This paper presents an architecture designed and used to facilitate the collection of disparate data sets well suited for data fusion and data mining. Results show this architecture provides a flexible, dynamic framework for the capture and storage of a myriad of dissimilar data sets and can serve as a foundation from which to build a complete data fusion architecture.**

I. Introduction

With the advent of ever greater computing capability, there is a corresponding, if not greater, capability to capture and store large amounts of miscellaneous data. While capturing data may be a relatively simple task, migrating the data from raw output to a final, analyzable form usually requires a number of additional steps, each of which can prove difficult to design and implement [1].

The task gets even more complex when the multiple data sets are involved [1,2]. It is useful to test differing data sets against a given data mining algorithm to ensure the efficacy of the algorithm is not due to "curve fitting" or some special property of the underlying data. This also requires any software for data mining have certain dynamic, configurable properties in order to handle diverse data sets.

Finally, because the data sets tend to be large and number of intermediate steps extensive, there must to be a way to automate the entire data collection and analysis process in order for it to be of any practical use [3].

All these requirements result in several major obstacles that must be overcome. The first is data import. Systems can consist of multiple data sources (such as sensors) that output streams of data in a format, not generally suited for storage in a database. The data must be recognized, scrubbed and transformed for it to be of use [4]. A typical data source configuration is depicted in Fig. 1.

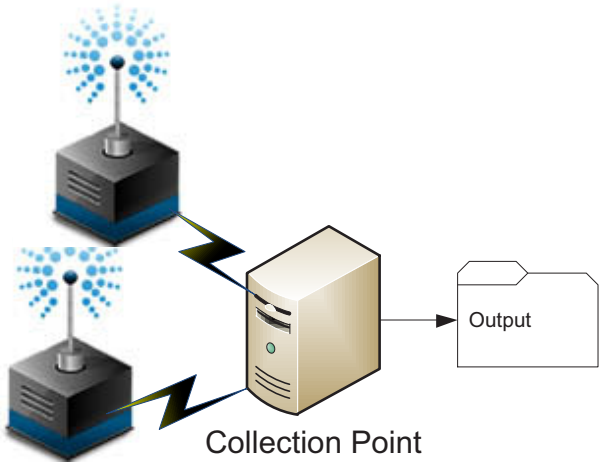

Fig. 1. Sensors transmit data to a collection point

In addition, a wide variety of input sources often leads to an equally diverse set of file formats to handle. Some data might be text-delimited, others free-form, still others jagged or fixed-length. The possibilities can be as wide-ranging as the data sources themselves [5]. Good software engineer techniques are needed to address to significant variation in requirements without leading to unmaintainable or extensible software [6, 7].

Next, there is the Extraction, Transform, Load (ETL) process which transforms raw data into something that can then be mined, either in relational or multi-dimensional formats. Extraction involves migrating data from its source, such as a file on an FTP server to a more flexible data repository, such as a table in a relational database. Transform involves removing bad or missing data values moving from less suitable to more suitable data types, and normalizing or denormalizing when appropriate [3]. The transform process also may involve translating the data, supplying missing values or applying derived calculations. There is also no reason for output to reside in a specific location. The Load process involves migrating the data from its current location to a final repository, which can be located virtually anywhere deemed appropriate, potentially in multiple locations simultaneously as depicted in Fig. 2.

Fig. 2. Moving data from a collection point to external repositories

There are a number of commercial ETL tools such as SQL Server Integration Services (SSIS), IBM Data Storage and AdeptiaETL suite that exist for this purpose, but they proved impractical on either a cost basis or the level of customization required or both.

Once the data is in a destination repository, data mining and data fusion can begin [8]. Data mining is an automated technique where special algorithms explore data sets looking for interesting relationships and trends. Popular data mining techniques include Decision Trees, Neural Network Algorithms, Hard/Soft Clustering and many others [3]. Data fusion aids in data mining by applying temporal, spatial or other relationships across unlike data sets in order to discover dependencies and key influencers [8].

Finally, because the results from a data mining algorithm can be difficult to interpret by atypical human observer, there needs to be some sort of visual display to put the results in context. Such a display can include charts, grids, color variations, dials or other dashboard elements to emphasize important results and put them in a form that can be easily understood.

Object-oriented architectures and methodologies have been proposed to handle complex system such as the one described [5, 6, 9]. Such systems attempt to reduce complexity by hiding implementation details from consumers. They also, via abstraction, try to reuse and extend functional components in ways that avoid code duplication and tight coupling of components. Such systems are easier to debug, extend and maintain and tend to have smaller code bases [6, 10].

This paper presents a software architecture in development designed to reduce the complexity required to import disparate source files and provide ETL processes used to generate relational datasets and datamarts for use in data fusion and data mining with particular emphasis on Time Series data although it is easily extensible to any type of structured or semi-structured data. While initially a standalone architecture, this design approach is adaptable for use in commercial ETL tools or as a hybrid system combining the power of both commercial and customization approaches.

II. Problem Statement

Any system designed to handle differing file types, import processes, source and destination types, servers and databases and other repositories will require a lot of customization. The challenge is to be able to handle a wide range of individual scenarios without becoming overly complex. Even with a relatively small number of distinct processes the number of permutations in code can quickly become very difficult to maintain or extend [11]. What's needed is an architecture that can support a diverse and flexible set of imports, processes and destinations without becoming overly complex. It must also be extensible enough to accommodate additional data without require major changes in the existing codebase.

Once the data is in the repository it must be transformed for analysis to include creating and implementing data fusion elements across data sets, This should also be a flexible and dynamic process as the transformations and data mining techniques, such as Time Series, should be interchangeable as needed.

The architecture proposed combines the following attributes

1. A externally configurable class library for the import process
2. A class library and callable database objects to manage the ETL processes and data fusion
3. Custom and commercial data mining techniques
4. Custom and commercial display tools for results

III. Implementation

The development environment for this project is Visual Studio 2010. The programming language is C#. Achieving requirement 1 (import) of the architecture required the use of polymorphic techniques in support of the import class library. This was due to the diversity of the data sets that needed to be mined and variable locations where data originated. Among the types initially supported were:

1. Delimited-text files such as csv format with different delimiters
2. Fixed-length formatted files
3. Free-form files where data elements occurred in regular patterns but without traditional formatting
4. Database objects

Polymorphism is a facet of object-oriented design which provides the ability to generate objects which share similar abstract functions and characteristics but differ only in their specific implementation. An object of a base class can then "morph" into any of the inherited classes through a simple assignment operator. Consumers of that object do not need to be aware of the specific implementation class or methods, rather they can still treat the object in its base form using calls to abstract methods which are automatically overridden by the actual inherited class.

Take the example of an "animal" object. Whether a dog, bird or person, animals perform many of the same functions, such as eating, breathing or sleeping. A particular implementation of "eat food" will vary from animal to animal, but the basic function, "eat food" does not. The animal object can expose a method "eat food" which is then overridden by the bird object that eats seeds from a feeder; the dog object that eats dog-food from a bowl and a person object that dines at a restaurant.

As a result a consuming program, without knowing what kind of animal it has or how it eats, can presume the object in question knows how "eat food" appropriately to its type. The consumer can also only has to invoke a single "eat

food" method to cause the animal to engage in eating behavior. This allows any necessary state-awareness on the part of an overall system to move from a global awareness imposed on a system to a local one imposed on an object, leading to fewer interdependencies among individual components and greater system resiliency.

In addition to abstract methods overridden by descendants, an object-oriented program can also expose basic methods common to a set of classes. So in polymorphism, objects inherit the base class functionality and implement the necessary abstract methods, providing differentiation to distinguish them from other inherited objects. Such an approach offers a number of advantages to the architecture.

1. Creating a standardized import interface, which is relatively easy to describe and consume. Despite a high degree of flexibility and complexity in the underlying process, the consumer requires very few lines of code to implement an import.
2. While the "import" process may vary greatly from data source to data source, the complexity of the individual import can be hidden within the derived class or classes. This leads to more loosely coupled and tightly cohesive code components.
3. Extending the import process to include new data sources requires only deriving a new base class and will not affect the existing implementation.
4. Use of abstract and derived classes means less reliance on conditional statements and code duplication that tend to make software more difficult to maintain and debug.

As an example, consider 3 datasets. 1 is simple comma-delimited data with headers. The second is tab-delimited but only has headers in certain files. The third file is comma-delimited with headers, but the data is only valid for files that start with a date. Creating a coherent file parser would involve coding a number of functions:

1. A function to parse the file into lines with specialized exceptions for the header line in each file and non-valid lines in files 2, 3.
2. A function to parse each resulting line to an import record. Each file would need a different parser with exceptions for the headers and any non-data lines.
3. A function to check data elements for type and validity.
4. A function to parse the resulting data records into the proper SQL import command.

Even a rudimentary implementation of such an application will require a significant construction of IF/CASE statements interspersed within each parser to account for the many variances among the import data sets. An execution branch in this example might have numerous permutations to account for, each of which must be coded and then maintained in the event of further changes. Add additional varied datasets and the number and variety of execution branches grows exponentially more difficult to manage.

The component of the proposed architecture to manage data imports to a data fusion database or warehouse follows these basics steps:

1. A data source is identified and linked to an Import Type object
2. The Generic Import Type object configures import criteria and the destination connection.
3. The Generic Import Type object creates a suitable Import Source Type object containing rules for import and parsing the data stream.
4. The Import Source Type streams the data in and parses it into an Import Set, which consists of collections of records. Each record has a collection of fields, field types and field values.
5. The Import Set then moves records to an output destination..

The initial set of classes deal with the data itself. A class was created to stored field or column data along with type information. A collection of fields constitutes a record. Each record contains a method to output itself into a string that can be used to generate a SQL insert command or stored procedure call. These classes are not abstract because the potential variability of configurations is quite small. An abstract collection of records is an ImportSet (IS). Each ImportSet contains a connection to an output repository along with a method to iterate through its records, retrieve the insert command and output each record. The ImportSet is abstract in order to support different output repositories such as a Microsoft SQL Server database, Oracle Server or simple text file.

The next real abstraction created deals with the problem of different file formats. Currently, this project deals only with file-based data sources, but other data sources, such as a remote database are likely candidates in the future. Implementation consists of a series of abstract objects:

ImportType (IT) – specifies the import source in general form, and handles configuration of source/destination options.

ImportSourceType (IST) – handles specific file formats such as comma-delimited and any special rules for parsing a stream into an ImportSet.

ImportSet (IS) – handles moving from memory to a data respository.

Combining these objects allows for the creation of a relatively simple but flexible input engine that hides much of the underlying complexity of the import process. Typical consumption and usage of the objects in a program is demonstrated by the following pseudocode:

```
Function InputSomeData(SourceName) Returns null Begin
        IT = new ImportType
        IT.Configure(SourceName)
        IT.DoImport
End
```

```
Function ImportSource.DoImport Returns null Begin
        IST = new ImportSourceType
        IS = new ImportSet
        IST.ParseSourceToImportSet(IS)
        IS.MoveToDestination()
End
```

In order to avoid excessive hard-coding of server, database and other names within classes, a configuration utility is under development that will allow an XML file to be used for additional ad-hoc configuration. Use of XML will allow for greater extensibility with a reduced need to recompile.

IV. Results

A program to both consume and demonstrate the architecture was built using the Windows Presentation Foundation (WPF). The core architecture was built around the following base classes:

1. Field
2. Record

And the following abstract classes:

1. ImportSourceType
2. ImportSet
3. GenericImportType

The abstract classes contain a combination of virtual, abstract and standard methods and properties. Several helper functions were also created to determine the correct objects and apply known configuration information. Since most of the import files consist of delimited text data, the first extension to this architecture was for handling standard delimited file types. A DelimitedFileType base class was created. This extension required roughly 110 lines of code and is able to handle any standard delimited file type.

Test 1 – A Simple File

The first actual file type to import was a series of windmill measurements saved out to a directory as tab-delimited files called CAESWind data. These files were relatively simple to process because they all consisted of a single header line and used standard formatting with several important execptions:

1. CAESWind data contains header information which is inconsistent.
2. CAESWind data contains a mix of date and numeric data.
3. CAESWind data contains bad dates which must be ignored.

Since CAESWind files are tab-deliminated, a CAESTabDelimintedTextType class inherited from DelimitedFileType was created. Aside from configuration information (which will be eliminated with the configuration utility), the new class required approximately 50 lines of code to implement, including curly braces, comments, exception handling and whitespace. The WPF application used to import CAESWind data requires only 3 lines of code for implementation.

Test 2 – A Moderately Complete File

The next extension was for some hydro plant data called IFHydro. IFHydro is also tab-deliminated with a complete different set of problems:

1. Header data is inconsistent.
2. Some files have no data at all.
3. Some records reflect a sensor off-line, with all zero values in the data
4. Some records are fragmented with incomplete data

Extending the DelimitedFileType to accommodate the IFHydro data required almost exactly the same number of lines of code despite a somewhat more complex set of requirements.

Test 3 – A Highly Complex File

The next extension involved data from a military facility. Called the Mates dataset, it was comma-delimited but also a freeform file. This hybrid and inconsistent combination created a number of issues:

1. Header data is nonexistent.
2. Data records run both vertically and horizontally in the file.
3. Records come in groups within the file. Group separators consist of a series of specific link breaks and values
4. Certain data elements pertain to a single record while other data elements pertain to all members of a data group.

	A	B	C	D	E
203	Point System Nam	MATES.CIED.AC01.PURGE			
204	Trend Every:	COV			
205	Date Range:	1/26/2012 00:00:00 - 2/1/2012 23:59:59			
206	Report Timings:	All Hours			
207	No data in range specified.				
208					
209	**********				
210	Point System Nam	MATES.CIED.AC01.SAT			
211	Trend Every:	Trend COV (3.000)			
212	Date Range:	1/26/2012 00:00:00 - 2/1/2012 23:59:59			
213	Report Timings:	All Hours			
214	1/26/2012	0:55:51	49.27	-N-	NONE
215	1/26/2012	0:56:35	52.38	-N-	NONE
216	1/26/2012	0:56:46	55.41	-N-	NONE
217	1/26/2012	0:56:57	58.52	-N-	NONE
218	1/26/2012	0:57:08	61.6	-N-	NONE
219	1/26/2012	0:57:21	64.68	-N-	NONE

Fig. 3. Mates Data Sample

Even with some exception handling, comments and white space, the overall implementation of the Mates Import took roughly an hour to complete, requiring about 130 lines of

C# code. Hence the 3 extensions to the architecture to support 3 distinctly different file import types, from raw data to a record in a SQL Server database table, with varying levels of complexity was accomplished using less than 300 lines in total. In addition, no changes were necessary to the WPF import application or any of the underlying classes, maintaining full backward compatibility within the class library.

V. Conclusion and Future Work

Object-oriented programming was introduced with the goal of simplifying software development by hiding or encapsulating object complexity behind simpler to use methods and properties [6]. Polymorphism was designed to allow a developer to minimize code duplication and maximize flexibility in an architecture [5, 6]. In this initial implementation, the import of a number of very different datasets using a relative few lines of code was accomplished. In addition, the results indicate the overall size of the class library need only grow slightly to accommodate new input as well as destination formats.

The architecture clearly demonstrates both simplicity and flexibility in addressing the problem of diverse data imports in handling the Mates, IFHydro and CAESWind datasets and is expected to perform similarly on additional varied future datasets. There remains considerable work to be done, however. It is quite likely that certain data imports will be so large as to make it impractical for the import set to cache records in memory as it currently does. It will be necessary to modify the ImportSet class to handle serialization to a file or even a database for large raw datasets and then stream the preprocessed records back for import.

In addition, the architecture needs to be extended to support the other tiers of the data fusion process: Data Mining and Display. For these, the architecture will have to extend to support a series of pluggable data mining algorithms, in particular, a proposed hybrid combination of Time Series and other advanced algorithms in order to improve overall predictive capability. Combined with generic display elements such as dials, grids, charts, etc. this architecture will provide a significant capability for combining and exploring datasets of widely varying originations, sizes and attributes. There is also currently only limited ETL which will have to be extended to accommodate the more stringent requirements for data mining models. These proposed extensions are under development and will be addressed in future work.

Acknowledgements

Work supported by the U.S. Department of Energy under DOE Idaho Operations Office Contract DE-AC07-05ID14517, performed as part of the Center for Advanced Energy Studies, and the Instrumentation, Control, and Intelligent Systems (ICIS) Distinctive Signature of Idaho National Laboratory, and Bish's RV.

VI. References

[1] Y. Qing, Z. Jing and W. Haiyang, "Business Process-Oriented Software Architecture for Supporting Business Process Change," in *International Symposium on Electronic Commerce and Security*, 2008.

[2] F. Bushmann, "Introducing the Pragmatic Architect," *IEEE Software,* pp. 10-11, Sep-Oct 2009.

[3] J. Han, M. Kamber and L. Pei, Data Mining Concepts and Techniques, 3rd Ed., Morgan Kaufmann, 2011.

[4] K. McCarty and M. Manic, "An Adaptive Architecture for Hydroinformatics Design and Implementation," in *International Symposium on Hydroinformatics and Ecohydraulics*, Concepcion, Chile, 2009.

[5] X. Cui, S. Yanchun and X. Sai, "Architecture Design for the Large-Scale Software-Intensive System: A Decision-Oriented Approach and the Experience," in *Engineering of Complex Computer Systems*, 2009.

[6] R. S. Pressman, Software Engineering, A Practitioner's Approach, 6th Ed., McGraw Hill, 2005.

[7] G. Buchgeher and R. Weinreich, "An Approach for Combining Model-Based and Scenario-Based Software Architecture Analysis," in *Fifth International Conference on Software Engineering Advances (ICSEA)*, 2010.

[8] K. McCarty and M. Manic, "A Temporal-Spatial Data Fusion Architecture for Monitoring Complex Systems," in *International Conference on Human Systems Interation*, 2010.

[9] M. Waterman, J. Noble and G. Allan, "How Much Architecture? Reducing the Up-Front Effort," in *AGILE India*, 2012.

[10] F. Buschmann, "Software architecture and reuse-an inherent conflict?," in *Software Reuse: Advances in Software Reusability*, 1994.

[11] O. Barais, "A framework to specify incremental software architecture transformations," in *EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005.