2009

# CoDoSA: A Lightweight, XML-Based Framework for Integrating Unstructured Textual Information

John R. Talburt
*University of Arkansas at Little Rock*, jrtalburt@ualr.edu

Eric D. Nelson
*University of Arkansas at Little Rock*, ednelson@ualr.edu

# CoDoSA: A Lightweight, XML-Based Framework for Integrating Unstructured Textual Information

**John R. Talburt**
University of Arkansas at Little Rock
jrtalburt@ualr.edu

**Eric D. Nelson**
University of Arkansas at Little Rock
ednelson@ualr.edu

**ABSTRACT**

One of the most fundamental dimensions of information quality is access. For many organizations, a large part of their information assets is locked away in Unstructured Textual Information (UTI) in the form of email, letters, contracts, call notes, and spreadsheet. In addition to internal UTI, there is also a wealth of publicly available UTI on websites, in newspapers, courthouse records and other sources that can add value when combined with internally managed information. This paper describes a system called Compressed Document Set Architecture (CoDoSA) designed to facilitate the integration of UTI into a structured database environment where it can be more readily accessed and manipulated. The CoDoSA Framework comprises an XML-based metadata standard and an associated Application Program Interface (API). It further describes how CoDoSA can facilitate the storage and management of information during the ETL (Extract, Transform, and Load) process to integrate unstructured UTI information. It also explains how CoDoSA promotes higher information quality by providing several features that simplify the governance of metadata standards and enforcement of data quality constraints across different UTI applications and development teams. In addition, CoDoSA provides a mechanism for inserting semantic tags into captured UTI, tags that can be used in later steps to drive semantic-mediated queries and processes.

**KEYWORDS**

XML, framework, UTI, unstructured information, embedded metadata, data architecture

## INTRODUCTION

Many organizations are realizing that a significant portion of their information assets reside in unstructured formats, much of which is textual. Unstructured Textual Information (UTI) is largely an artifact of inter-personal, often informal, communication that does not follow a systematic pattern that would allow it to be directly ingested into a structured database environment. Depending upon the nature of the organization, UTI may reside in many forms such as emails, letters, contracts, proposals, medical records, or call-center notes. By some estimates, a typical organization has 2 to 10 times more unstructured data than structured data (Inmon and Nesavich, 2008).

Because UTI can take a variety of forms, organizations face many challenges in trying to integrate this information into their operational and analytical environments. In particular the key entities and entity attributes in UTI must first be located and extracted in a process known variously as "named entity recognition," "entity identification," or "entity extraction" (Wikipedia, 2009). Although there is a wealth of research about the extraction process itself such as Arlotta, L., Crescenzi, V., Mecca, G., & Merialdo, P. 2003; Chiang, C., Talburt, J., Wu, N., Pierce, E., et.al. 2008; Talburt, J., Wu, N., Pierce, E., & Hashemi, R., 2007; Talburt, J., Wu, N., Pierce, E., Chiang, et.al. 2007; and many others, the focus of this paper is on providing a framework that enables the extraction process and promotes information quality by simplifying the enforcement of metadata rules and standards across applications.

Several researchers have developed conceptual frameworks around ontology (Ding, Y., Embley, D., and Liddle, S., 2006; Embley, D., Campbell, D., Jiang, Y., et.al., 1999) and others have promoted the value of standardization across applications (Doan, A., Ramakrishnan, R., Chen, F., et. al., 2006). The CoDoSA Framework is unique in its approach to UTI information management in that it provides a simple, script-driven method for separating the information design role from the developer role as a way to promote discipline and governance in coding and metadata standards. In this respect, CoDoSA is similar to the Unstructured Information Management Architecture (UIMA) developed by Ferrucci. D. & Lally, A. (2004). Though UIMA supports XML metadata descriptors, the metadata structures are still instantiated at the application level and are primarily intended to promote inter-process communication rather than data standardization across all applications.

## MOTIVATION

The extraction of information from UTI can be a complex process. Each type of document often requires its own extraction logic. Even within the same class of documents, there may be enough differences in style and patterns of information that it must be decomposed into even smaller sub-classes, each with its own extraction logic. The result is a proliferation of UTI extraction processes that must be developed and maintained, often with each application having a different output format. This approach also requires a second process to transform the various extraction outputs into a consistent format that can be loaded into the database environment as illustrated in Figure 1.

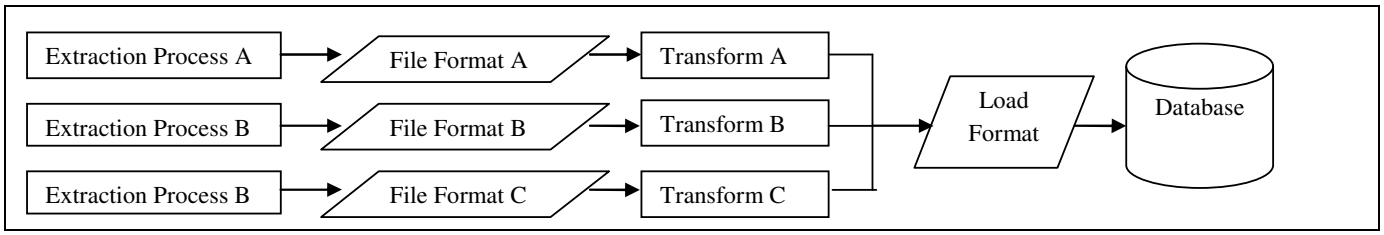


**Figure 1: Typical UTI ETL Process**

Without a great deal of discipline and governance in coding and metadata standards, a new file format and a concomitant file transform can be added to the system each time a programmer implements a new extraction process. The effect is to double the amount of coding and maintenance effort each time an extraction application is written. Moreover, if the ultimate load format changes, then appropriate changes must be made in each of the individual transforms. In short, a system of this design is a recipe for creating information quality problems.

The Compressed Document Set Architecture (CoDoSA) is a lightweight XML framework and application program interface designed to mitigate the problem illustrated in Figure 1. It does this by providing a common interface and a shared metadata standard across all extraction processes. In the application of CoDoSA to this scenario, each extraction process would implement the CoDoSA interface (API) and use the same predefined metadata template called a "skeleton." Each DocSet created through the API retains a copy of the skeleton as metadata embedded into the DocSet along with the actual output values extracted from the UTI. Because all of the DocSets created using the same skeleton share the same metadata structure, they can all be easily transformed into a standard load format using a single transform that also implements the CoDoSA API. This is illustrated in Figure 2.

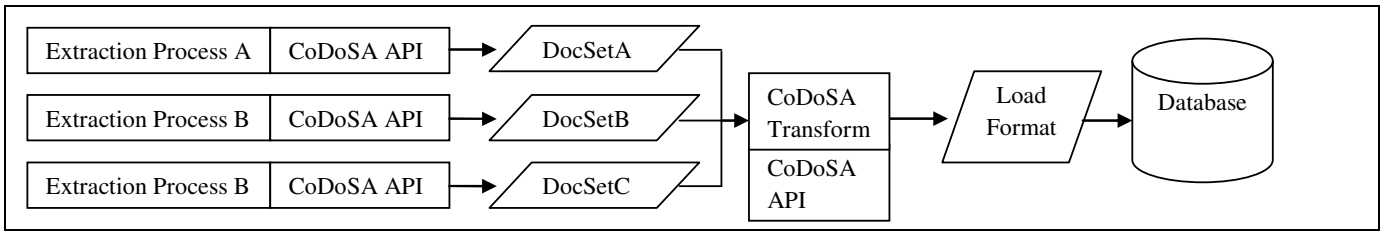


**Figure 2: A UTI ETL Process Using the CoDoSA Framework**

## FRAMEWORK COMPONENTS

XML has gained wide acceptance because of its flexibility and expressive power, however its primary application has largely been in the interactive transactional environment such as those done over the Internet or other client-server applications. CoDoSA provides a way to leverage the descriptive power XML in the batch processing environment while retaining the performance characteristics of traditional flat-file processing.

Compressed Document Set Architecture (CoDoSA) is a data management framework that facilitates the storage and manipulation of information in the form of mark-up documents. CoDoSA has two principal parts

1. A standard for encapsulating into a single dataset, called a "document set" or "DocSet," that comprises two segments
    a. XML document schema (metadata), and
    b. Multiple instances of documents conforming to the metadata schema stored in a compressed, tag-delimited format (CTF).
2. An API that defines a series of interfaces that implement methods for
    a. Establishing the metadata layout of a DocSet
    b. Writing to a DocSet
    c. Reading from a DocSet
    d. Appending to a DocSet

#### CoDoSA Logical File Structure

A DocSet comprises two segments

1. Metadata Segment, a well-formed XML document that includes a
    a. <document_Set> Section that provides information that applies to the entire DocSet such as creation date, owner, contact, etc.
    b. <document_Schema> Section that gives the metadata the describes the layout common to all documents stored in the DocSet such as item names, cardinality, format, etc.
2. Document Segment that contains a series of document instances in a special compressed, tag-delimited format called CTF. Each CTF document instance conforms to the metadata description in the <document_Schema> Section.

These segments are illustrated in Figure 3.

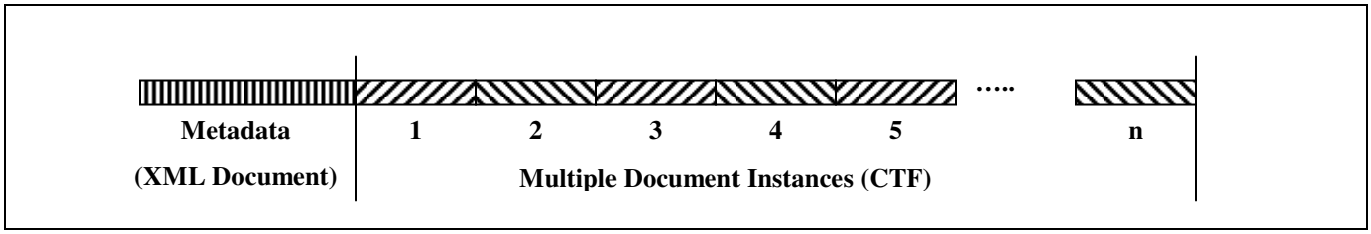


**Figure 3: Segments of a CoDoSA Document Set**

Even though the idea of embedded metadata in digital systems is common and even patented for some applications such as digital cameras (Fuller, C., Gorkani, M., and Humphrey, R, 2004), the use of embedded metadata for general file processing has not been widely adopted with the exception of image files e.g. Tagged Image File Format (TIFF) Standard. Apparently due to the legacy of the unbridled determination to save every byte of storage, the concept of embedded metadata has never been popular in commercial file processing despite the enormous amount of manual effort routinely expended in reconciling externally defined file layouts with the actual dataset they purport to describe.

Figure 4 shows an example Data Type Definition of a CoDoSA Metadata Segment that was used in support of an actual UTI ETL system in which the DocSets were produced by "web crawlers" (agents).

```
<CoDoSA_Metadata>
        <document_Set>
                <creation_Date>          system date
                <docSet_Identifier>      value provided by Open_DocSet()
                <instance_Delimter>      valid values: LF(default), CR, LC
                <docSet_Description>
                <docSet_Contact>
                <source_Description>
                <source_URL>
                <source_Date>
                <agent_Type>
                <agent_Name>
                <agent_Developer>
                <semantic_Domain>        default domain for semantic labels
                <append_Date>            system date at time of append
                <append_Description>     value provided by Open_DocSet_Append()
        <document_Schema>
                <simple_Item>
                        <identifier>
                        <description>
                        <tag>            system generated
                        <min>            valid: 0 to 99, default: 0
                        <max>            valid: 1 to 99, default: 1, min<=max
                        <semantic>       valid: see IA Semantic Label Standard
                        <format>         valid: see IA Format Standard
                        <default>        default value if missing
                        <type>           data conversion type
                <complex_Item>
                        <identifier>
                        <description>
                        <tag>            system generated
                        <min>            valid: 0 to 99, default: 0
                        <max>            valid: 1 to 99, default: 1, min<=max
                        <semantic>       valid: see list, Appendix A
                        <simple_Item>
                        <complex_Item>
```

**Figure 4: XML Document Type Definition for CoDoSA Metadata**

### DocSet Skeletons

A DocSet Skeleton is a particular instance of the DocSet Metadata Document Type Definition. The ability to define both complex and simple items allows DocSets to have hierarchical structures, best visualized as a tree structure. Figure 5 shows a file DocSet structure "Marriage" for storing information extracted from a published marriage announcement.

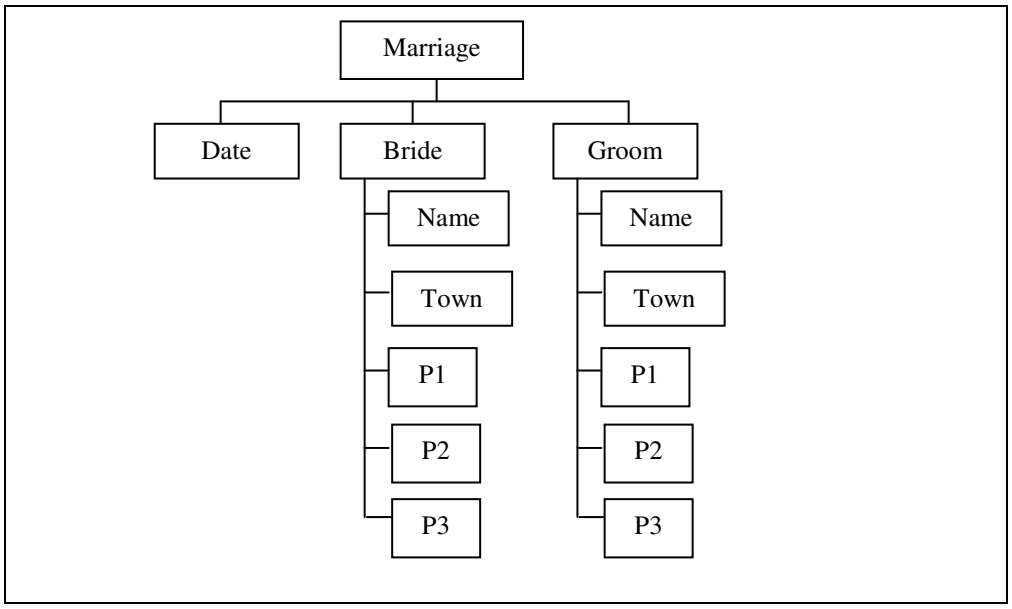


**Figure 5: Marriage Announcement Metadata Structure**

At the top level of the document "Marriage" there is one simple item "Date" (of marriage) and two complex items "Bride" and "Groom". The items "Bride" and "Groom" both have five sub-items, "Name" (of bride or groom), "Town" (home town), "Parent 1", "Parent 2" and "Parent 3."

Figure 6 shows an instance of the CoDoSA metadata (a "CoDoSA Skeleton) that describes the structure shown in Figure 5. Note that the DATE item is defined as a <simple_Item> and that it is a required item because the <min> and <max> values of item cardinality are both one, i.e. one and only one DATE item must be present. The metadata for DATE also has a <format> constraint of MMDDCCYY. Note that the BRIDE and GROOM items are defined as <complex_Item>. All of these values including the <description> are provided by the skeleton designer. Different UTI domains would call for different skeletons.

The BRIDE and GROOM items are defined as a <complex_Item> and both are required although the PARENT sub-item is optional have a 0:3 cardinality constraint. Also note that the NAME, TOWN, and PARENT sub-items also have <semantic> tags, "CN_Full", "AD_City", and "CN_Full" respectively. The values for the semantic tags are taken from the organization's standard information taxonomy.

When metadata tags are missing, then their values are provided by default according to the definition shown in Figure 4. For example, if <min> and <max> are not provided, the default is 0:1.

Figure 6 shows a DocSet Skeleton that implements the tree structure shown in Figure 5. Since skeleton (metadata) items can be complex and have multiplicity, references to an item must be fully qualified so that they are unambiguous. For example in Figure 6, a reference to the second parent of the bride would be "BRIDE.PARENT[2]"

Figure 6 shows a DocSet Skeleton to implement the marriage data tree structure shown in Figure 5.

```
<CoDoSA_Metadata>
	<document_Set>
		<creation_Date></creation_Date>
		<dataset_Identifier> </dataset_Identifier>
		<dataset_Description> </dataset_Description>
		<dataset_Contact> </dataset_Contact>
		<source_Description> </source_Description>
		<agent_Type> </agent_Type>
		<agent_Name> </agent_Name>
		<instance_Delimiter>LF</instance_Delimiter>
		</document_Set>
	<document_Schema><document_Name>MARRIAGE</document_Name>
		<simple_Item><identifier>MDATE</identifier>
			<description>Date of marriage</description>
			<format>DATE_MMDDCCYY</format>
			<min>1</min><max>1</max></simple_Item>
		<complex_Item><identifier>BRIDE</identifier>
			<description>Bride's Information</description>
			<min>1</min><max>1</max>
			<simple_Item><identifier>NAME</identifier>
			<description>Bride's name, unparsed</description>
			<semantic>CN_Full</semantic>
			<min>1</min><max>1</max></simple_Item>
			<simple_Item><identifier>TOWN</identifier>
			<description>Bride's Town of Residence</description>
			<semantic>AD_City</semantic>
			<min>0</min><max>1</max></simple_Item>
			<simple_Item><identifier>PARENT</identifier>
			<description>Bride's Parent Name, unparsed</description>
			<semantic>CN_Full </semantic>
			<min>0</min><max>3</max></simple_Item>
			</complex_Item>
		<complex_Item><identifier>GROOM</identifier>
			<description>Grooms's Information</description>
			<min>1</min><max>1</max>
			<simple_Item><identifier>NAME</identifier>
			<description>Grooms's name, unparsed</description>
			<semantic>CN_Full</semantic>
			<type>STRING</type><min>1</min><max>1</max></simple_Item>
			<simple_Item><identifier>TOWN</identifier>
			<description>Grooms's Town of Residence</description>
			<semantic>AD_City</semantic>
			<min>0</min><max>1</max></simple_Item>
			<simple_Item><identifier>PARENT</identifier>
			<description>Groom's Parent Name, unparsed</description>
			<semantic>CN_Full</semantic>
			<min>0</min><max>3</max></simple_Item>
		</complex_Item>
	</document_Schema>
</CoDoSA_Metadata>
```

**Figure 6: CoDoSA Skeleton for Marriage Announcement**

### CoDoSA Document Segment

The Document Segment is a series of tagged documents that represent specific instances of the document schema defined in the Metadata Segment. Whereas the document schema is defined in terms of the XML syntax, the document instances are represented in CTF, a compressed, tag-delimited format.

Document instances are separated (delimited) by either a line feed character or a combination of a line feed and carriage return character depending upon which was specified by the <instance_Delimiter> element of the metadata segment.

Each document instance is numbered with a Document Sequence Number starting a 1. Document Sequence Numbers are automatically generated by the system and inserted into each document instance written to the Document Segment.

#### *CTF - COMPRESSED, TAG-DELIMITED FORMAT*

CoDoSA is characterized as a "lightweight" XML framework because specific instances of a document are stored in a special compressed format called CTF that stands for Compressed, Tag-delimited Format. Since each document must follow the structure of the DocSet metadata skeleton, it is not necessary to repeat the use of the full XML tags in each document instance. This serves to significantly decrease the size of the DocSet and overcome the objection that XML is too verbose for manipulating large datasets. In CTF an item value is represented by a character string that

- Starts with a left bracket ([)
- Ends with a right bracket (])
- Has a 1- or 2-character tag identifier immediately following the opening left bracket.

A CTF item value is represented by a string of CoDoSA characters immediately following the item tag with trailing blanks omitted, and ending with the matching right bracket, i.e., matching in the sense of being at the same nested level. An example of a CTF complex item would be

[T[U321][Vxyz]]

Where

- "[" denotes the start of the item value
- "T" is a 1-character tag that identifies the element complex item
- The string "[U321][Vxyz]" represents the value of the element with tag "T".

Note that in this example, the value of the complex item "T" comprises two CTF simple items, [U321] and [Vxyz] identified by tags U and V respectively.

#### *DOCUMENT SEGMENT EXAMPLE*

As an example, suppose that there is a web crawler "Dallas_Herald_v2.1.java" that is used to extract the named entities from web-posed marriage announcements. Furthermore suppose that the crawler implements a CoDoSA API to create its output as a DocSet using the tree structure described in Figure 3 and DocSet metadata description given in Figure 4.

(Announcement 1) Dallas Herald, May 20, 2008 - The marriage of Jane Smith of Tyler to Ray Johnson of Dallas was held at the First Methodist Church. Attending were the bride's parents John and Linda Smith, and the groom's father Frank Johnson and his mother Judy Williams and her husband Fred Williams.

(Announcement 2) Dallas Herald, May 21, 2008 - Mary Jones of Dallas and Herman Jacobs of Fort Worth were married at the Steven's Chapel in Plainview. The bride was given away by her mother, Lucy Jones.

In this scenario, the DocSet created by processing these announcements using the skeleton from Figure 6 is shown in Figure 7.

```
<CoDoSA_Metadata>
        <document_Set>
                <creation_Date>20080917</creation_Date>
                <dataset_Identifier>marriage_example.txt</dataset_Identifier>
                <dataset_Description>Example of a CoDoSA doc set</dataset_Description>
                <dataset_Contact>John Doe x22436</dataset_Contact>
                <source_Description>Dallas Herald Online, 8/22/2001</source_Description>
                <agent_Type>WebBot</agent_Type>
                <agent_Name>Dallas_Herald_v2.1.java</agent_Name>
                <instance_Delimiter>LF</instance_Delimiter>
                </document_Set>
        <document_Schema><document_Name>MARRIAGE</document_Name>
                <simple_Item><identifier>MDATE</identifier>
                        <description>Date of marriage</description>
                        <tag>B</tag><format>DATE_MMDDCCYY</format>
                        <min>1</min><max>1</max></simple_Item>
                <complex_Item><identifier>BRIDE</identifier>
                        <description>Bride's Information</description>
                        <tag>C</tag><min>1</min><max>1</max>
                        <simple_Item><identifier>NAME</identifier>
                        <description>Bride's name, unparsed</description>
                        <tag>D</tag><semantic>CN_Full</semantic>
                        <min>1</min><max>1</max></simple_Item>
                        <simple_Item><identifier>TOWN</identifier>
                        <description>Bride's Town of Residence</description>
                        <tag>E</tag><semantic>AD_City</semantic>
                        <min>0</min><max>1</max></simple_Item>
                        <simple_Item><identifier>PARENT</identifier>
                        <description>Bride's Parent Name, unparsed</description>
                        <tag>F</tag><semantic>CN_Full </semantic>
                        <min>0</min><max>3</max></simple_Item>
                        </complex_Item>
                <complex_Item><identifier>GROOM</identifier>
                        <description>Grooms's Information</description>
                        <tag>G</tag><min>1</min><max>1</max>
                        <simple_Item><identifier>NAME</identifier>
                        <description>Grooms's name, unparsed</description>
                        <tag>H</tag><type>STRING</type><min>1</min><max>1</max></simple_Item>
                        <simple_Item><identifier>TOWN</identifier>
                        <description>Grooms's Town of Residence</description>
                        <tag>I</tag><semantic>AD_City</semantic>
                        <min>0</min><max>1</max></simple_Item>
                        <simple_Item><identifier>PARENT</identifier>
                        <description>Groom's Parent Name, unparsed</description>
                        <tag>J</tag><semantic>CN_Full</semantic>
                        <min>0</min><max>3</max></simple_Item>
                </complex_Item>
        </document_Schema>
</CoDoSA_Metadata>
[A1][B05202008][C[DJane Smith][ETyler][FJohn Smith][FLinda Smith]][G[HRay Johnson][IDallas][JFrank
Johnson][JJudy Williams][JFred Williams]]
[A2][B05212008][C[DMary Jones][EDallas][FLucy Jones][G[HHerman Jacobs][IFort Worth]]
```

**Figure 7: A MARRIAGE DocSet with Two Document Instances**

There are some essential differences between the MARRIAGE DocSet shown in Figure 7 and the MARRIAGE Skeleton shown in Figure 6. First is the inclusion of the CTF Document Segment appended to the end of the CoDoSA_Metadata Segment. In this version of the framework, even though the CoDoSA_Metadata is itself a well-formed XML document, it only forms a "header" for the entire DocSet dataset.

A second difference is that in the MARRIAGE DocSet the items in the <document_Set> Section have been populated with instance values such as a <creation_Date> value of "20080917" and a <agent_Type> value of "WebBot". These and other values in this section were written to the DocSet by the application "Dallas_Herald_v2.1.java."

Another important difference is the insertion of <tag> items and values in the <document_Schema> Section of the DocSet. For example, a new item <tag> with a value of "B" was added to the MDATE item. The <tag> values are inserted by the CoDoSA API when the agent invokes (opens) the MARRIAGE Skeleton prior to writing to the DocSet. The <tag> values are generated by the API and are the tags used in the CTF markup to relate instance values to their metadata definitions. In this case "B" is the CTF markup tag for the MDATE item, and can be seen in the two document instances.

### DocSet Build Process

The process for creating a DocSets for a particular operation has the following steps

1) A Data Architect designs a DocSet Skelton "S" for the application
2) The Data Architect creates and deploys the skeleton with appropriate documentation to a Skelton Library
3) The Application Developer writes his or her extraction code so that the application
    a) Invokes Open_DocSet(S, D) when the program starts. The API call loads the Skeleton S from the library, validates its structure, and inserts CTF tags into the <document_Schema>. It also opens a path to the output DocSet D.
    b) Invokes a series of Write_DocSet_Item(V, P) calls that populate the <document_Set> Section of D, e.g. Write_DocSet_Item("20080917", "creation_Date")
    c) For each document the application processes, the program must
        i) Invoke Open_Document() to make an empty instance X of the tree structure defined in the skeleton S.
        ii) Perform a series of Write_Logical_Item(V, P) calls that populate the instance X with a value V for an item with path P, e.g. Write_Logical_Item("Tyler", "BRIDE.TOWN") in Figure 7.
        iii) Invoke Write_Document() to convert the populated document instance X into a CTF markup instance and write the CTF instance to the output DocSet D
    d) Close_DocSet() after all documents have been processed and written to D.

Reading a DocSet is basically the reverse with methods to open the DocSet and read document instances sequentially through a series of Read_Document() and Read_Logical_Item(P) calls. However, in addition to reading instance values, the Reader API also includes a Read_Metadata_Item(P) that allows an application to request metadata values such as the <min> cardinality for an item. This functionality allows an application to implement more complex business rules than those enforced by the API.

## CoDoSA FRAMEWORK AND INFORMATION QUALITY

The implementation of the CoDoSA Framework provides several benefits, especially when trying to manage the ETL process for unstructured to structured integration that involves the coordination of many processes. Perhaps the most significant is that CoDoSA provides a convenient way to better manage information through the design principle of "Separation of Concerns". These include:

1) Promoting better information architecture through the separation of physical and logical data structure concerns
    a) Each DocSet carries its own metadata description. The application developer writes to a pre-defined logical schema and does need to know the physical layout.
    b) The ability to embed DocSet-level information such as a description, owner, and a creation date, promotes better data management, auditing, and tracking.

c) The framework supports complex (composite) data items and multiple instances of the same item without the overhead of a DBMS implementation

2) Promoting better information quality through the separation of process and data architecture concerns

   a) The data architecture standard is enforced in the DocSet Skeleton designed by the data architect. All processes using the same skeleton can only instantiate the items as defined in the skeleton

   b) All processes using the same skeleton are subject to the same information quality rules. The API enforces these rules and provides the process with a return code to signal when a violation has occurred such as incorrect data type, invalid formats, invalid characters, or item optionality or cardinality violations.

3) Reducing the number of job set-up errors and process re-runs by eliminating some of the primary sources of setup errors including:

   a) Basing the setup on an incorrect or out-of-date record layout or guessing at the layout when none is available.

   b) Introducing typographical errors in record layouts during the transcription from paper to electronic format.

   c) Errors in multi-step processes where the output layout from one step does not correspond with the input layout expected by the following step.

4) The Compressed-Tag Format used to store document instances is well-suited to support unstructured information integration that are typically "sparse item applications", i.e. a large number of item types are defined, but in any one document, only a few will actually have a value.

5) Simplifying and shortening program development time by providing a standard data model that is defined by an XML script (skeleton) rather than implemented through interface coding or a fixed database schema. Clearly the overall goal of decoupling the business rules layer from the implementation layer of an application can be accomplished through means other than CoDoSA such as using class/interfaces as wrappers for database connections. However almost all of these techniques rely upon some level of coding as opposed to simply creating an XML script. In this sense the CoDoSA API is a "meta-interface" because the true configuration of the interface is determined by an underlying script. No coding changes to the CoDoSA API are required to implement a different skeleton or to changes in an existing skeleton.

## CoDoSA FRAMEWORK AND SEMANTIC-MEDIATED PROCESSING

Another feature of the CoDoSA framework is that its embedded metadata provides support for semantic tagging that can in turn support semantic-aware processes. Ever since Berners-Lee, T., Hendler, J., and Lassila, O. (2001) proposed the concept of the Semantic Web, there has been a growing interest in using explicit semantic tags to replace the complex and often inaccurate inference of semantics from context. Embley, D. (2004) and Maedche, A., Naumann, G., and Staab, S. (2002) have proposed the semantic approaches to information extraction.

The XML definition for the CoDoSA metadata as shown in Figure 2 includes a <semantic> tag. For organizations that have, or want to develop, a taxonomy or ontology, the standard keywords can be included in the DocSet skeletons associated with individual items. Embedding these keywords into the DocSets can provide support for downstream, semantically-enabled processes such as information retrieval and process automation.

## AN INDUSTRY APPLICATION OF CoDoSA

This section shows code fragments taken from a system to extract property (real estate) related information from HTML pages. The system was a developed as a pilot project by a large data management firm and implemented the CoDoSA framework and API as a way to coordinate the work of five programmers who were writing site-specific web crawlers.

An advantage of CoDoSA is that it allows the cleaning, validation, and standardization of item values to be deferred until after the individual DocSets have be collected and aggregated (CoDoSA Transform Process of Figure 2). By moving these responsibilities from the application developer to the transform application, develop time is decreased and data anomalies are handled in a more uniform and consistent manner. As a simple example, one site may describe the number of bathrooms in a residential dwelling with the value "2-1/2" while another may use "2.5". Since values are handled as strings in the DocSet, both values can written to the same logical item (e.g NbrOfBaths) then later transformed to a common standard by a single transform process.

### Writing a CoDoSA Document

Figure 8 is a fragment of Java code that shows how a CoDoSA document is opened and closed using the Open_DocSet() and Close_DocSet() methods, respectively. Note that a document instances can be appended to an existing DocSet using the Open_DocSet_Append() method. In this application, the HTML pages to be processed have already been downloaded into a local file.

```
cdsWriteApi writer = new cdsWriteApi();
public void createCoDoSA(File file, String skeleton, String output, boolean append,
String appendDes) {
    try {
        int rc;
        if(append)
            rc = writer.Open_DocSet_Append(skeleton, output, appendDes);
        else
            rc = writer.Open_DocSet(skeleton, output);
        if(rc != 0) return;

        processPages(file);

        writer.Close_DocSet();
        System.out.println("File " + file.getName() + " created.");
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

**Figure 8: Code Fragment Using Open and Close Methods**

After the DocSet is opened for writing, new document instances can be created using the Open_Document() method. The Open_Document method basically creates an in-memory structure using the skeleton as the data model. Once a new document instance is opened, item values can be written to the document (leaves of the tree) by their logical name using the Write_Logical_Item method. The Write_Logical_Item method validates that the logical name is defined in the DocSet Skeleton. Note that in the current implementation, document instance must be written and read sequentially, however logical items within a document may be written or read in any order. After all items have been written to a document, the Write_Document() method closes the document instance and writes a CTF encoded version of the in-memory tree to the DocSet. This sequence of operations is illustrated in Figure 9.

```
Public void processPages(File file){
    String elem = "", mid = "";
    try {
        // open the file of HTML Pages
        BufferedReader f1 = new BufferedReader(new FileReader(file));
        // start reading and processing the HTML

        while (morePages){
           // start a new document instance
           writer.Open_Document();

           // after processing HTML page and extracting the elementary school district
           int rc = writer.Write_Logical_Item(elem, "school.elementary");
           if (rc != 0) System.out.println(writer. getErrorMessage(rc));

           // after processing HTML page to extract the middle school district
           int rc = writer.Write_Logical_Item(mid, "school.middle");

           // write out the complete instance
           writer.Write_Document();
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

**Figure 9: Code Fragment Writing Logical Items**

#### Reading a CoDoSA Document

A CoDoSA document is read by the cdsReadApi class. The first step is to open the file stream with the Open_DocSet. After opening, the method Read_Document reads successive document instances from the DocSet into memory. Element values can be read from the document by their logical names using the Read_Logical_Item method. As in the Write_Logical_Item() method, items must be referenced by their qualified element name. Figure 10 shows a segment of Java code for reading the logical item *school.Elementary*. Finally the file stream is closed with a call to the Close_DocSet method. Also note that a cdsWriteApi was declared this is so that it can be used to output the textual error message.

```
public void propertyMaster(String filename){
    String elem = "";
    cdsReadApi reader = new cdsReadApi();
    cdsWriteApi writer = new cdsWriteApi();

    int rc = reader.Open_DocSet(filename);
    if (rc != 0){
        System.out.println("() return = " + rc + ", " + writer.getErrorMessage(rc));
    }

    while(rc == 0){
        rc = reader.Read_Document();
        if (rc == 0){
            int value = 0;
            elem = reader.Get_Logical_Item("school.Elementary");


        }
    }
    reader.Close_DocSet();
}
```

**Figure 10: Code Fragment Reading Logical Items**

## CONCLUSION AND FUTURE WORK

The working prototype of the CoDoSA framework was used successfully used to coordinate the work of 5 programmers developing web crawlers to extract UTI from public websites. In that application, the framework proved to be an effective means to promote rapid development and a means to enforce discipline and governance in coding and metadata standards across a large set of applications. Future plans call for refining the current Java implementation and making into an Open Source development project available to the information management and information quality research communities.

## REFERENCES

1. Arlotta, L., Crescenzi, V., Mecca, G., and Merialdo, P. (2003) Automatic annotation of data extracted from large web sites. *Sixth International Workshop on the Web and Databases (WebDB 2003)*, pp. 7-12, San Diego, California, June 2003.
2. Berners-Lee, T., Hendler, J., and Lassila, O. (2001) The Semantic Web. *Scientific American,* vol. 36, no. 25, pp. 34-43, May 2001.
3. Chiang, C., Talburt, J., Wu, N., Pierce, E., et.al. (2008) A case study in partial parsing unstructured text. *Fifth International Conference on Information Technology*: New Generations, Las Vegas, NV: IEEE Press, pp. 447-452.
4. Ding, Y., Embley, D., and Liddle, S. (2006) Automatic creation and simplified querying of semantic web content: An approach based on information-extraction ontologies. *The Semantic Web – ASWC 2006*, Springer, pp. 400-414.
5. Doan, A., Ramakrishnan, R., Chen, F., DeRose, P., Lee, Y., McCann, R., Sayyadian, M., and Shen, W. (2006) Community information management. *IEEE Data Engineering Bulletin*, March 2006.
6. Embley, D. (2004). Towards Semantic Understanding -- An Approach Based on Information Extraction Ontologies. *Fifteenth Australasian Database Conference (ADC2004)*, Dunedin, New Zealand. CRPIT, 27.
7. Embley, D., Campbell, D., Jiang, Y., Liddle, S., Lonsdale, D., Ng, Y., and Smith, R. (1999) Conceptual-model-based data extraction from multiple-record web pages. *Data and Knowledge Engineering*, Vol. 31, No. 3, pp. 227-251, November 1999.
8. Ferrucci. D. and Lally, A. (2004) UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, June 2004.
9. Fuller, C., Gorkani, M., and Humphrey, R (2004) Embedded metadata engines in digital capture devices. United States Patent 6833865.
10. Inmon, W. and Nesavich, A. (2008) *Tapping into Unstructured Data,* Prentice Hall, p. 20.
11. Maedche, A., Naumann, G., and Staab, S. (2002) Bootstrapping an ontology-based information extraction system for the web. In P.S. Szczepaniak, J. Segovia, J. Kacprzyk, L.A. Zadeh (eds.), *Intelligent Exploration of the Web. Series - Studies in Fuzziness and Soft Computing*. Springer/Physica-Verlag.
12. Talburt, J., Wu, N., Pierce, E., & Hashemi, R. (2007). Entity identification using indexed entity catalogs. In H.R. Arabnia & R.R. Hashemi (Eds.), *2007 International Conference on Information and Knowledge Engineering* (pp. 338-342). Las Vegas, NV: CSREA Press.
13. Talburt, J., Wu, N., Pierce, E., Chiang, C., Heien, C., Gulley, E. & Moore, J. (2007). Entity Identification in Documents Expressing Shared Relationships. N. Mastorakis, S. Kartalopoulos, D. Simian, A. Varonides, V. Mladenov, Z. Bojkovic, E. Antonidakis (Eds.), *11th World Scientific and Engineering Academy and Society International Conference on SYSTEMS* (Vol. 2, pp. 223-228). Agios Nikolaos, Crete: WSEAS Press.
14. Wikipedia entry "Named entity recognition" retrieved January 20, 2009.