PMML and UIMA Based Frameworks for Deploying Analytic Applications and Services

David Ferrucci[1], Robert L. Grossman[2] and Anthony Levas[1]

1. Introduction - The Challenges of Deploying Analytic Applications and Services

It is convenient to divide data into structured data, semi-structured data and unstructured data. By structured data, we mean data that is organized into fields or attributes. Examples include database records. Semi-structured data has attributes but does not have the regularity of structured data. Data defined by HTML or XML tags are examples of semi-structured data. Unstructured data lacks attributes or fields and includes text data, signals, images, video, audio or similar data. Of course, data may be a combination of one or more of these types. For example, the content of a message can be unstructured text and the metadata semi-structured XML tags.

By an analytic application, we mean an application that analyzes data. Examples include statistical or data mining models, extracting features from images or signals, parsing text, and analyzing text using natural language processing or language models.

Today, the process of developing analytic applications presents several challenges.

1. **Deployment.** Analytic models for structured data are commonly built using a statistical or data mining application, but deployed in operational systems where the models must be often be re-coded using C++, Java or other language. This slows down the deployment of new analytic applications and the updating of current ones.

2. **No standard architecture.** There is no generally accepted standard architecture for analytic applications, which increases the cost and time required to develop them. In particular, a standard architecture for developing and deploying analytic applications would facilitate code reuse.

[1] IBM T.J. Watson Research Center
[2] Open Data Group and University of Illinois at Chicago

DM-SSP'06, August 20, 2006,
Philadelphia, Pennsylvania, USA.

3. **The difficulty of cleaning, preparing, integrating and transforming data.** The cost of developing analytic applications is usually dominated by the cost of collecting, cleaning, transforming and preparing the data prior to its modeling or scoring. If, as is often the case, the data is from two or more separate sources or systems, then the data must also be integrated, complicating the process even further.

In this white paper, we describe an approach to developing and deploying analytic applications that addresses each of these challenges.

During the past several years, the Data Mining Group's Predictive Model Markup Language or PMML has emerged as an interchange format for analytic models for structured data, while IBM's Unstructured Information Management Architecture (UIMA) has emerged a component based architecture for developing analytic applications for unstructured data.

In this paper, we give quick introductions to PMML and UIMA, describe some of the common structure of PMML and UIMA, and describe two different ways that applications combining both PMML and UIMA can be built.

To illustrate the core ideas, we describe a case study that shows how an analytic application can be built that automatically transforms messages at one security level into messages at a lower security level.

2. Example Scenarios

In this section, we describe two simple scenarios using PMML based and UIMA based models.

Categorization. Given a text message, there is a standard trick to create a state vector from it. One simply counts the number of times each word in a dictionary occurs, forms a state vector of the counts, and normalizes the state vector appropriately. This is often called a bag of words model. Given the state vector, standard statistical models for clustering or probabilistic clustering can then be used to categorize the message. There are PMML based models for bag of word models and several different clustering models that can be used in this way to encapsulate the data required for categorizing messages.

Processing XML tagged data to lower classification level. Consider an XML tagged message that is classified at a classification level C1 and contains specific information about a specific individual in a specific location. By coarsening the location data and replacing specific information about individuals with more generic information, such as "report of hostile activity within the last three hours in named area of Interest A", the message, in many cases, has a lower level of classification, say C2, and can be more broadly distributed. UIMA Annotators can be used to do the various translations required in this example.

The examples in this section are described in more detail in Section 8.

3. PMML-Based Analytic Services – Key Concepts

PMML is declarative and data-centric in the sense that a markup language is used to describe analytic models in a platform and application independent fashion. In contrast, UIMA is process-centric. In PMML, you create models; in UIMA, you create engines.

PMML's approach to developing and deploying analytical applications is based upon a few key concepts:

- **View analytic models as first class objects**. We think of analytic models as being described using the Predictive Model Markup Language or PMML. Applications or services can be thought of as producing PMML or consuming PMML. A PMML XML file contains enough information so that an application can process and score a data stream with a statistical or data mining model using only the information in the PMML file.

- **Support data preparation.** As mentioned above, data preparation is often the most time consuming part of the data mining process. PMML provides explicit support for many common data transformations and aggregations used when preparing data. Once encapsulated in this way, data preparation can more easily be re-used and leveraged by different components and applications.

- **Facilitate using different architectures for learning models and scoring models.** Broadly speaking most analytic applications consist of a learning phase that creates a (PMML) model and a scoring phase that employs the (PMML) model to score a data stream or batch of records. The learning phase usually consists of the following sub-stages: exploratory data analysis, data preparation, event shaping, data modeling, & model validation. The scoring phase is typically simpler and either a stream or batch of data is scored using a model. PMML is designed so that different systems and applications can be used for producing models (PMML Producers) and for consuming models (PMML Consumers).

- **View data as event based.** Many analytic applications can be naturally thought of as event based. Event based data presents itself as a stream of events that must are transformed, integrated, or aggregated to produce the state vectors that are inputs to statistical or data mining models. The current version of PMML provides implicit support for event based processing of data; future versions are expected to provide explicit support.

	Learning	Scoring
PMML	In learning, a PMML producer processes a data set to produce a PMML model.	A PMML consumer i) inputs a PMML model and then ii) using the model can process one or more data records to produce scores.
UIMA	A Collection Processing Engine (CPE) can be used in learning or training to analyze a collection or group of documents to produce an Analysis Engine (AE). CPEs can also be used much more generally for a variety of other tasks.	An Analysis Engine (AE) analyzes documents based on the model produces by the CPE in the training phase. This is also called an "Applier."

4. Overview of PMML

In this section, we give a brief introduction to PMML [Grossman:2002].

PMML is an application and system independent interchange format for statistical and data mining models. More precisely, the goal of PMML is to encapsulate in an application and system independent fashion a model so that two different applications (the PMML Producer and Consumer) can use it. PMML is developed by a vendor led working group, which is part of the Data Mining Group [DMG].

PMML consists of the following components:

1. **Data Dictionary.** The data dictionary defines the fields that are the inputs to models and specifies the type and value range for each field.

2. **Mining Schema.** Each model contains one mining schema that lists the fields used in the model. These fields are a subset of the fields in the Data Dictionary. The mining schema contains information that is specific to a certain model, while the data dictionary contains data definitions that do not vary with the model. For example, the Mining Schema specifies the usage type of an attribute, which may be active (an input of the model), predicted (an output of the model), or supplementary (holding descriptive information and ignored by the model).

3. **Transformation Dictionary.** The Transformation Dictionary defines derived fields. Derived fields may be defined by normalization, which maps continuous or discrete values to numbers; by discretization, which maps continuous values to discrete values; by value mapping, which maps discrete values to discrete values; or by aggregation, which summarizes or collects groups of values, for example by computing averages.

4. **Model Statistics.** The Model Statistics component contains basic univariate statistics about the model, such as the minimum, maximum, mean, standard deviation, median, etc. of numerical attributes.

5. **Model Parameters.** PMML also specifies the actual parameters defining the statistical and data mining models per se. Models in PMML Version 3.0 include regression models, clusters models, trees, neural networks, Bayesian models, association rules, sequence models, support vector machines, rule sets, and text models.

Figure 1 illustrates the relationship of the Data Dictionary, Mining Schema and Transformation Dictionary. Note that inputs to models can be defined directly from the Mining Schema or indirectly as derived attributes using the Transformation Dictionary.

PMML also provides some additional infrastructure that is helpful when for modeling, including:

- A variety of different data types including, arrays of values, sparse arrays of values, and matrices.
- Support for missing values, and several different ways of working with them.

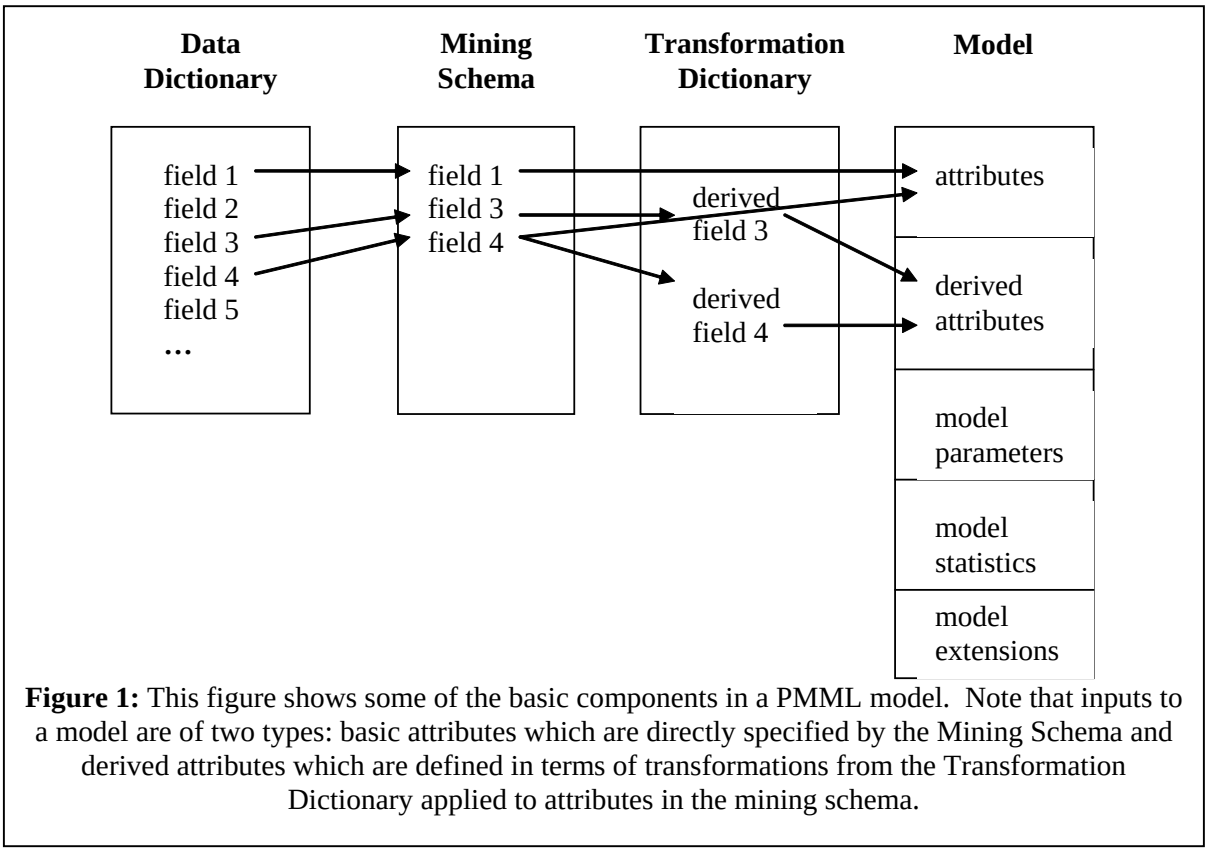


Figure 1: This figure shows some of the basic components in a PMML model. Note that inputs to a model are of two types: basic attributes which are directly specified by the Mining Schema and derived attributes which are defined in terms of transformations from the Transformation Dictionary applied to attributes in the mining schema.

5. Overview of UIMA

Key Concepts

In this section, we describe some of the key concepts associated with the Unstructured Information Management Architecture (UIMA) and framework.[1]

For simplicity and concreteness, we describe these concepts in the context of documents, although UIMA is broadly applicable to other types of semi-structured or unstructured data. This section is adapted in part from [Ferrucci:2004].

- **Analysis Engines**. The fundamental processing component for document analysis in UIMA is the *Analysis Engine* (AE). AEs have a standardized interface and may be declaratively composed to build aggregate analysis capabilities. AEs built by composition have a recursive structure - primitive AEs may be core analysis components implemented in C++ or Java, whereas aggregate AEs are composed of such primitive AEs or other aggregate AEs. Because primitive and aggregate AEs have exactly the same interfaces, it is possible to recursively assemble advanced analysis components from more basic elements while the implementation details are transparent to the composition task. AE's analyze artifacts (documents, images, conversations, videos etc.) represented in a common data structure called the Common Analysis Structure or CAS. The logical interface to an Analysis Engine is essential CAS in/CAS out. AE's behavior is assumed to depend only on its input CAS. AEs are implemented to perform a wide variety of artifact analysis tasks. These include, for example, document tokenization, parsing, name-entity detection, relationship detection, speaker identification, translation etc. Generally, AEs produce metadata describing the artifact in the form of annotations. Annotations are used to label regions of the original artifact. Different types of Analysis Engines include for example, *Annotators* and *CAS Consumers*. Annotators typically produce new annotations, adding metadata to a CAS. CAS Consumers extract metadata from a CAS and populate structured resources. These are described in more detail below.

- **Common Analysis Structure (CAS).** Analysis Engines operate on and share data through a standardized representation called the CAS. The CAS represents the artifact being analyzed (document, image, conversation, video etc) and the set of assertions produced by some sequence of Analysis Engines. The set of assertions in the CAS is intended to describe the artifact or features there-of and may be collectively referred to as *artifact metadata* or just metadata. The

[1] IBM freely provides a framework implementation supporting the creation and composition and deployment of UIMA-compliant analytic applications. This implementation is included in the UIMA software development kit (SDK) and is available on the IBM alphaWorks site. The UIMA Java framework is also available as open-source http://uima-framework.sourceforge.net/.

metadata may take the form of annotations. UIMA annotations are similar to the notion of annotations in other systems like Catalyst [Mardis] and TIPSTER [Grishman]. These are implemented as separate objects that point into the original artifact rather than as in-line markup. This allows multiple sets of interpretations or labelings of an artifact to exist simultaneously and does not corrupt the original representation. Analysis Engines may use different interfaces to read and update the CAS. The UIMA framework may also contain different internal implementations of the CAS. The UIMA framework provides different interfaces to the CAS including the JCAS which maintains fast indices to CAS contents and projects a native java object programming model to the data elements in the CAS.

- **Type Systems.** A CAS conforms to a user-defined Type System. This is essentially a schema or data model that defines the types of objects and properties that may be asserted into a CAS by a component analytic.

- **Analysis Engine Flows.** Aggregate AE's specify a workflow of internal component engines. UIMA includes a control component interface called the *Flow Controller*. The UIMA framework provides an implementation of this interface that allows the specification of simple static linear flows among component engines. The developer, however, is free to create complex and dynamically-determined flows by providing a different implementation for the Flow Contoller interface. During the execution of an Aggregate Engine the Flow Controller considers incoming CASs (and possible other information about the aggregates' component engines or other external information) and determines the next engine in the flow that should get access to each CAS. The framework then performs the necessary routing. Each AE in the flow considers the CAS content and updates the metadata with new assertions. The updated CAS is the results of the analysis.

- **Collection Readers.** Another UIMA component interface is the *Collection Reader*. Collection Reader implementations connect to information sources (databases, file systems, audio or video streams etc.) and segment these sources into artifacts. Each artifact is used to initialize a CAS for down-stream analysis by workflows of Analysis Engines. The logical interface for a Collection Reader is "get next CAS". Collection Readers may be implemented to support a wide variety or source formats. They are required to report when the end of collection is detected.

- **CAS Consumers.** After an artifact has been analyzed by some flow of Analysis Engines the analysis results are contained in a CAS. The CAS is typically used to populate some structured resources that provide the application targeted and efficient access to the results of interest. *CAS Consumers* ingest CASs and interface with structured resources including for example, file systems, search engine indexers, databases or knowledgebase. They populate these resources with application-relevant metadata extracted from the CAS. The simplest case may be a CAS Consumer that converts the CAS content into XML format and writes it to a directory on the file system. A more complex example may be a CAS Consumer that extracts a subset of metadata, organizes it as a collection of facts over entities and populates a formal knowledge base. In any case, the structured resources populated by CAS Consumers are in turn used to support high-level decision support applications. CAS Consumers are so named because they are not intended to update input CASs.

- **Collection Processing Engines.** In UIMA, a special type of Aggregate Engine called a *Collection Processing Engine* may be assembled to perform collection-level analysis, where entire collections of documents are analyzed as a whole to produce collection-level results. Collection Processing Engines are made up of a Collection Reader, a set of Analysis Engines and a set of CAS Consumers. Examples of collection-level analysis performed by Collection Processing Engines include the creation of glossaries, dictionaries, databases, search indexes, and ontologies from the results of analyzing entire collections of artifacts. UIMA does not restrict the content, form, or technical realization of collection-level results; they are treated generally as structured information resources.

- **Component Descriptors.** All UIMA component implementations are associated with XML documents called *Component Descriptors*. These contain meta-data describing the component, including its identification, version, constituents (if an aggregate), configuration parameters and capabilities. This information is used to aid in tooling, component management, discovery, packaging and installation.

A UIMA Framework

A UIMA framework [Ferrucci:2004] supports the component or application developer in creating, composing and deploying UIMA-Compliant component and/or services interfaces. The UIMA SDK is a UIMA framework implemented in Java. In addition to Java-based analysis components, it supports the integration of C++ analytics and allows for the interoperation of component analytics written in other languages viz. services oriented interfaces.

UIMA framework functions are designed to simplify the development of analytic applications. They provide a variety of facilitates to the component and application developer. The following is partial list of UIMA framework facilities:

Type System Validation. UIMA defines an XML format for specifying type systems. The framework includes utilities for assisting in the definition and validation of type system definitions.

Component Descriptor Validation. The UIMA framework includes methods for validating component descriptor XML files.

Engine Creation. Analysis Engines are the pluggable components managed, composed and deployed by the framework. Developers, however, implement a simple interface called the *Annotator* interface. This interface is logically CAS In/CAS Out. Annotator implementations encapsulate the analysis algorithm code. Annotator implementations joined with their component descriptor are passed to factory methods provided by the framework. The framework wraps the annotator with infrastructure code and returns a pluggable analysis engine component that projects the standard analysis engine interface.

Engine Composition. Analysis Engines may be composed to produce aggregate analysis capabilities. The UIMA framework supports declarative specifications for describing aggregate engines in terms of component engines and a workflow specification. This declarative specification, called an aggregate descriptor, is passed to a framework factory method that encapsulates the aggregates internal

components and returns a pluggable Analysis Engine that projects the standard analysis engine interface. The UIMA framework will automatically create a type-system for an aggregate based on merging the type-systems of its named components.

Engine Deployment. Analysis Engines may be deployed as *Integrated* or *Remote Engines*. Integrated Engines are *co-located* with the framework run-time and other cooperating components making up an aggregate. CASs are shared as part of the same memory space with other integrated components. Remote Engines are deployed as network services. To communicate with Remote Engines the framework handles CAS serialization and transport using a variety of pluggable transport protocols, including SOAP. The engine developer need not write any special code to deploy an engine as either integrated or remote but rather indicate the desired option in the component's descriptor.

CAS Management and Transport. The UIMA framework manages CAS pools and handles CAS routing and transport among components cooperating as part of an Aggregate Analysis Engine or Collection Processing Engine. The framework is responsible for ensuring that, independent of engine deployment options, the CAS is delivered to component engines in the order determined by the appropriate Analysis Sequencer.

Collection Processing Management. The UIMA framework supports deployment options for Collection Processing Engines to exploit multi-threading and multi-processing environments to improve robustness and throughput for large collection processing application environments.

7. Building Applications Using Both PMML and UIMA

We can now summarize the discussion thus far:

By **models**, we mean statistical or data mining models. Models are defined by PMML files. As an example, an Analysis Engine for a UIMA-based analytic application can employ a PMML model to specify how the Analysis Engine processes documents or other data. As a second example, a Collection Processing Engine for a UIMA-based analytic application can produce a PMML model that is the result of analyzing a collection of documents

We assume that **components** for analytic processing depend only upon inputs and are described by standardized XML interfaces. Both PMML and UIMA support this view of components.

A **framework** provides an implementation of one or more analytic components in a particular a language; say C++, Java or Python. The Data Mining Group is a consortium of vendors who develop and market one or more PMML components. The UIMA Java framework of the UIMA SDK, is open source.

By an **analytics platform** we mean a standards-based collection of interoperable components for the analytic processing of data. Both PMML and UIMA support the key requirements of an analytics platform by employing models and components.

An analytics platform requires a standardized **container** for data and metadata. This , for example, is provided in UIMA by the CAS. PMML does not currently have an analogous concept, except as implicitly defined by data and mining schemas.

Finally, an analytics platform requires a **workflow** language so that different components can be combined. UIMA explicitly supports the integration of different components using workflow concepts; currently PMML does not but future versions of PMML are expected to do so.

Integration Approaches

The simplest way to develop UIMA/PMML based applications is simply to develop an application using UIMA and PMML based services.

Another approach to integrating PMML is to wrap PMML components as UIMA Analysis Engines and have these PMML components access and use PMML models as UIMA *Shared Resources*. This approach provides a relatively easy path to leveraging UIMA's facilities, including those for component aggregation, deployment, reuse, data sharing and workflow management, while enabling PMML capabilities to plug-in by changing how they access and use PMML models.

A more sophisticated approach aimed at achieving a tighter integration between UIMA and PMML may represent PMML models directly in the UIMA CAS and enhance a CAS interface to provide specialized functions for manipulating PMML models. This approach would have to continue to support direct XML-based access to PMML models to maintain backward compatibility with PMML components.

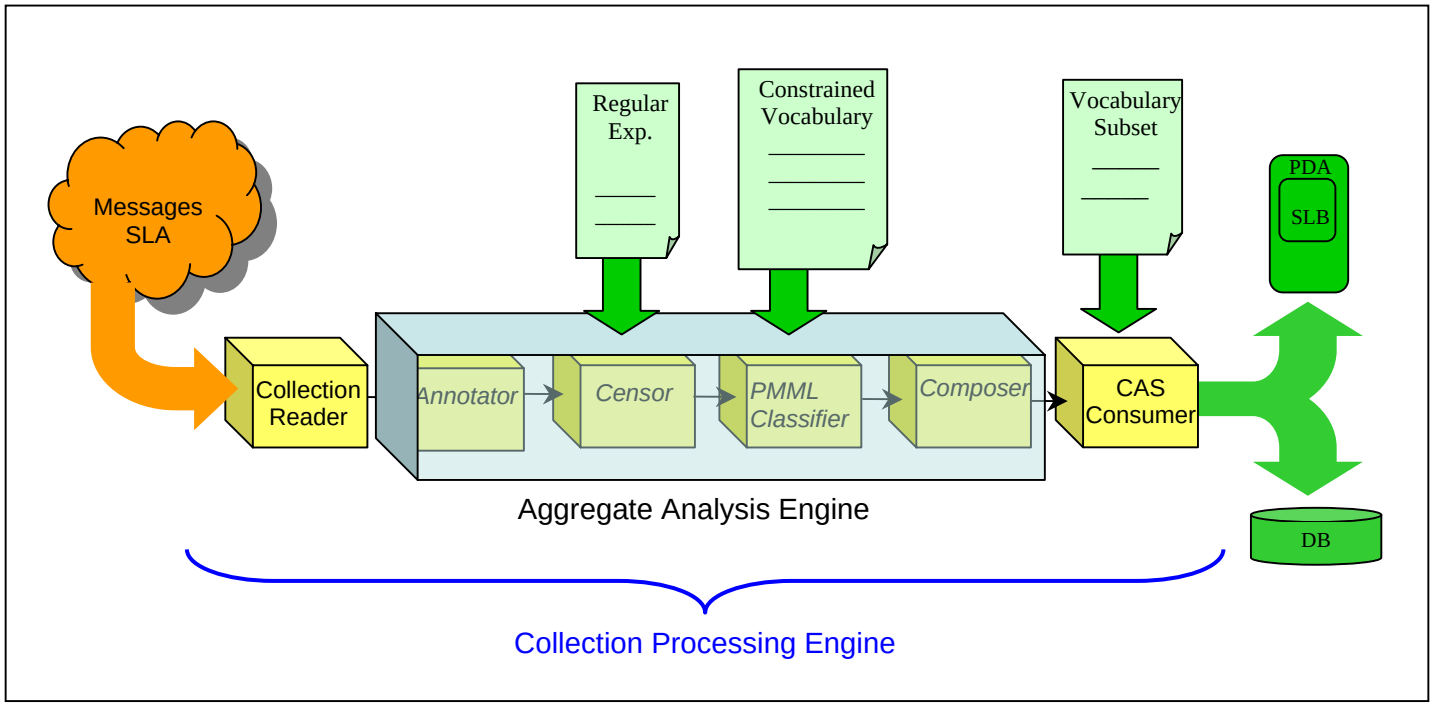
In this paper we start by describing a case study that takes the straight forward approach outlined above.

8. Case Study

This section is adapted from the technical report [OSC:2005].

In this section, we illustrate a *Message Transformation Application* using a UIMA-based and PMML-based architecture. In this application, incoming messages at a security level denoted as SLA, stream through a UIMA Collection Processing Engine (CPE) that transforms the text to produce outgoing messages at a lower security level, denoted as SLB. The following high-level sequence of transformation operations is applied to each message in series:

1. Metadata within the message are converted to annotations.
2. Certain words are removed due to their sensitivity.
3. A PMML model is used to categorize the message.
4. On the basis of this categorization, mappings are applied in turn to determine the *who, what, where* and *when* of the new message.
5. Additional filtering is done if required on the words and terms in the transformed message.



The behaviors and capabilities of the UIMA components comprising the CPE are as follows:

Collection Reader. The *Collection Reader* connects to the message source and iterates through the messages initializing a UIMA CAS for each. The CAS stores the message text to be processed as well as message-level metadata.

Annotator. The first UIMA Analysis Engine is an *Annotator* that parses the message, converting its structure and metadata to annotation objects that are indexed and stored in the CAS for processing by down-stream AEs. In particular, paragraphs with explicit security designations are annotated as either "blocked" or "pass-through" depending on a security-level threshold parameter.

Censor. The role of the *Censor* AE is to eliminate specific terms from the message text. These terms are specified by regular expressions listed in an external resource file. The cleaned text is stored in a bag-of-words object stored in the CAS.

PMML Classifier. The *Classifier* AE scores the terms in the constrained vocabulary according to their relevance to the bag-of-words associated with the message. The resulting scores are indexed in the CAS.

Composer. The Composer AE produces a transformed message by composing a string from the highest scored terms in the constrained vocabulary. In addition, paragraphs annotated as "pass-through" are appended to the string without modification. This string is stored in the CAS as a composed-message object.

CAS Consumer(s). The role of the *CAS Consumer* is to output the transformed message. In addition, an outgoing message filter may be applied to output only those transformed messages that include a chosen subset of the constrained vocabulary terms. A variety of CAS Consumers can be used simultaneously to affect a range of message storage or transmission options.

Regular expressions, representing censor words, terms constituting the constrained vocabulary and scoring engine model parameters, are stored in external resource files. In addition, all other UIMA component parameters are specified in associated *Resource Descriptor Files*. Thus, all the terms, patterns, and parameters may be modified or extended without requiring re-compilation of the code.

9. Summary and Conclusions

In this paper, we have introduced the key concepts and ideas behind the Predictive Model Markup Language (PMML) and the Unstructured Information Management Architecture (UIMA) and have shown how they may be combined to create an open, standards based platform for the analytic processing of structured, semi-structured, and unstructured data.

References

[DMG] The PMML Working Group is part of the Data Mining Group. See www.dmg.org.

[Ferrucci] D. Ferrucci and A. Lally, Building an example application with the Unstructured Information Management Architecture, IBM Systems Journal, Volume 43, 2004, pages

[Grishman] R. Grishman, Tipster Architecture Design Document Version 2.2, Technical Report, Defense Advanced Research Projects Agency (DARPA), U.S. Department of Defense (1996).

[Grossman:2002] Robert Grossman, Mark Hornick, and Gregor Meyer, Data Mining Standards Initiatives, Communications of the ACM, Volume 45-8, 2002, pages 59-61

[OSC:2005] Object Sciences Technical Report, A UIMA Example for the High to Low Processing of Messages, 2005.

[PMML] PMML documentation can be found on the web site.: sourceforge.net/projects/pmml/

[Mardis] S. Mardis and J. Burger, "Qanda and the Catalyst Architecture," AAAI Spring Symposium on Mining Answers from Text and Knowledge Bases, American Association for Artificial Intelligence (2002).