TAP: A Platform for Enabling Enterprises to Develop Business Specific Text Analytic Applications

Neeraj Agrawal
IBM India Research Lab
nagrawal@in.ibm.com

Scott Holmes
IBM Almaden Research Lab
holmes@us.ibm.com

Sachindra Joshi
IBM India Research Lab
jsachind@in.ibm.com

Sumit Negi
IBM India Resarch Lab
sumitneg@in.ibm.com

Abstract

Many enterprises are beginning to exploit the vast amount of data available on the Web, to streamline their business processes and gain advantage over their competitors. However, building text analytic applications that provide such vital business information, is very hard. Further, there are several functionalities that are common across many text analytic applications. In this paper, we provide a platform called TAP (Text Analytic Platform), that provides several tools and services that are used commonly across many text analytic applications. TAP could be used by business enterprises to build text analytic applications rapidly. It uses WebFountain to gather the application-specific data and provide other tools that help in developing and deploying application-specific miners.

Keywords

1. INTRODUCTION

The immense popularity of the Web has transformed the Internet into the largest global information infrastructure. Its existence has enabled ordinary citizens to become not only consumers of information but also its disseminators. This wealth of information has opened up new avenues for business. Today, many enterprises are beginning to exploit this vast amount of information to streamline their business processes and to gain advantage over their competitors. This has led to the development of business applications that mine the unstructured text of Web pages and extract relevant information that could lead to a better understanding of the factors affecting their business. However, building text analytic applications that provide such vital information, is hard.

There are several components that are common to many text analytic applications. These components can be abstracted out to build a platform that enables rapid development of text analytic business applications. Figure 1, depicts the key building blocks that make up a general text analytic application. All text analytic applications contain a data gathering module, a repository to store data, an indexer, a set of base-level miners also called annotators, and an environment for the execution of miners. For scalability reasons, these building blocks can exist on a distributed environment and communicate with each other using service-level protocols such as SOAP [5] and Vinci [1]. In this paper, we provide a platform called TAP (Text Analytic Platform), that could be used by businesses to build large-scale text analytic applications rapidly. TAP provides tools for data gathering, data storing and developing text analytic applications.

Data gathering is an important task for text-analytic applications. Search engines have been by far the most popular way of gathering topic-specific information. Search engines provide a query interface, that given a set of keywords, returns a ranked list of documents containing the key words. Though this method is very simplistic, it lacks the desired flexibility, power and semantic richness for a meaningful consumption of the information present on the Web. As an example, it is difficult for a business to automatically extract information from the Web that may act as a trigger event for the sale of their products. An Example of such trigger event could be indications of revenue growth of a company that could be a potential customer. To detect such trigger events automatically, one needs to gather all relevant documents. Most text-analytic applications need an application-specific document set. However, there is no simple and easy way of obtaining such a document set using the search engine interface. Thus there is a need for a more powerful way of gathering the set of relevant documents. TAP uses an alternative approach for building an application-specific (on topic) store. We argue that for building a relevant on-topic store for an application, certain types of entity identification and annotations needs to be done apriori. Such annotations could then be used to build on-topic store. Further, several pre-processing tasks such as language identification, removal of pornographic pages and duplicate pages need to be done. We call the modules that perform such general text analytics and annotations as *base level miners*. Performing such

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

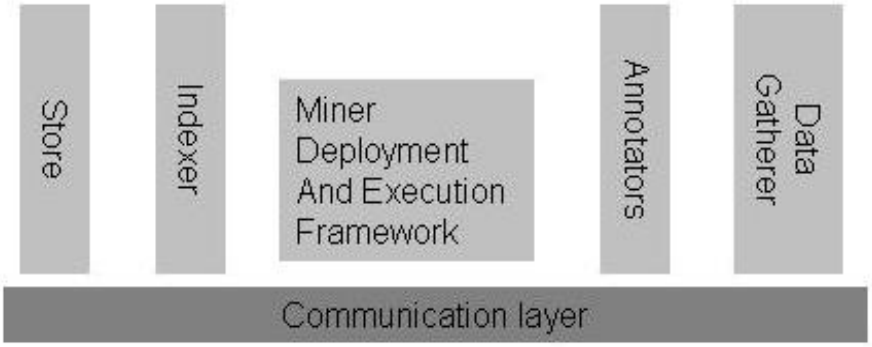

Figure 1: Basic Building Blocks of Text Analytic Applications

base level mining operations and annotations at the Web scale requires tremendous amount of resources.

TAP is built on top of WebFountain [4] that gathers documents from the Web, and then runs them through the *base level miners*. WebFountain is a platform for very large-scale text analytics applications. WebFountain itself could also be seen as a text analytic business application that can be used to provide semantically rich application-specific data to different businesses. It consists of a distributed crawler, store, indexer and several *base-level miners*.

TAP comprises of the WebFountain(WF) cluster, WF Web services, WF gateway and WF Appliance. It provides tools to build an application-specific store, a set of base level miners and, an environment for developing text analytic applications. The proposed platform is flexible, powerful and extensible.

Gruhl et.al. [4] describe in detail the different components used in performing large scale mining and addresses scalability issues. Thus the focus of the paper is on "an architecture for large scale text analytics". On the other hand this paper describes an approach that builds on top of the mined data provided from the WebFountain cluster. Our proposed technique is useful for companies who want to leverage this mined data for building their own high-level text analytic applications.

The rest of the paper is organized as follows. In Section 2, we briefly describe the building blocks of the WebFountain cluster. In Section 3, we first present an overview of TAP and then describe WF Web services and WF gateway in detail. In Section 4, we provide details of WF Appliance. We describe an example text analytic business application in Section 5, and discuss how our framework could be used to build it rapidly. Finally, in Section 6, we give our conclusion and some future work.

2. BACKGROUND

Our proposed framework uses WebFountain to gather application specific data. In this section, we present brief description of the different components of WebFountain.

WebFountain is designed as a loosely coupled, share-nothing parallel cluster of Intel-based Linux servers. As discussed in Section 1, WebFountain itself can be viewed as a text-analytic business application. Figure 2 depicts the key building blocks and overall architecture of WebFountain. Below, we briefly describe different components of WebFountain. Readers are referred to [4] for further details.

Data Gatherer: The data gathering component is responsible for collecting unstructured, semi-structured, and structured data from various sources, such as the World Wide Web, corporate intra-nets, news feeds, and, bulletin boards. The WebFountain crawler is distributed, meaning that performance can be scaled linearly by adding machines and continuous, meaning that once started, the crawler implements a strategy to maintain a fresh copy of every document by re-crawling the Web. In addition to crawling the Web and corporate intra-nets, the data gathering component is also responsible for acquiring other sources, such as bulletin boards, news feeds, and third-party-purchased data sources.

Store: In WebFountain storage model, all data in the system is represented as one or more entities, which are typed set of keys with associated typed values. The task of the WebFountain Store component is to manage entities represented as frames in XML files. The store provides functions to store, modify and access these entities. Every entity in the system has a 16-byte universal entity identifier called a UEID, which is hashed to determine storage locations, providing uniform distribution across all the devices.

Indexer: The WebFountain indexer is used to index not only text tokens from processed entities, but also conceptual tokens generated by annotators. The WebFountain indexer supports a number of indices, each supporting one or more different query types. Boolean, range, regular expression, and spherical queries are typical. The indexing approach is scalable and not limited by main memory as they adopt a sort-merge approach in which sorted runs are written to disk in one phase and final files are generated in a merge phase. The indexer supports the WebFountain Query Language (WFQL), the language that allows processes within the system to specify declarative combinations of result sets

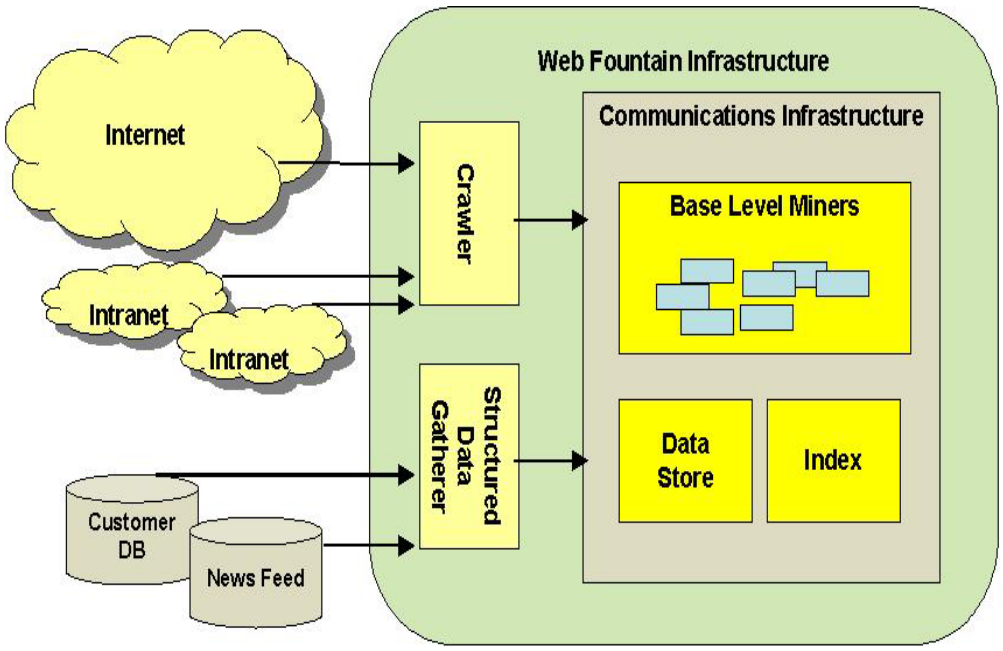

Figure 2: Architecture of WebFountain Cluster

from different parts of the system.

Base Level Miners: Miners are special-purpose programs that extract information from entities in storage and add new key-value pairs to the entities. These are also known as annotators. The basic purpose of miners is to remove noise such as duplicate and pornographic pages and to identify entities such as company and people names, locations and dates that could be used to build application-specific data store. Miners read certain key-value pairs of an entity, process them and store back new key-value pairs. As an example, the *porn miner* reads the *dotages content* of a page and writes back a key named *isPorn* that captures the information about whether the given page is pornographic or not. Output of a *miner* can be used by some other higher level *miner*. *Language detector*, *porn detector*, *company spotter* and *location spotter* are some of the examples for *base level miners*.

Communication Layer: WebFountain uses service oriented architecture for each of its component. It uses a lightweight, high-speed Simple Object Access Protocol (SOAP) derivative called Vinci [1]. Vinci is an architecture for building distributed systems using a light weight, point to point XML-like RPC protocol. Vinci is based on non-validated XML document exchange in order to allow for loose connections between distributed components. It dispenses with much of the overhead associated with security, verification, and correctness checking associated with communication between untrusted parties. It is therefore suited for fast and scalable intra-net application. The components of WebFountain cluster use Vinci and provide their capabilities to other components in form of Vinci services.

3. OVERVIEW OF TAP

3.1 Overview

TAP provides a platform for rapid development of text analytic applications. It consists of the WebFountain cluster, WF gateway, WF Web services and, WF appliance. TAP uses the pre-annotated data of the WebFountain cluster to build application-specific data store. It also provides tools and services needed for rapid development of text analytic applications. Figure 3 describes the different components and their interactions. WF gateway is the entry point for the outside world to the WebFountain cluster. WF Web services run on the WF gateway and provide access to the resources of the WebFountain cluster. WF Appliance uses the WF gateway to build an application-specific store and provides several tools and services that help in development of text analytic solutions . WF Web services and WF gateway enables TAP to access the pre-annotated data stored in the WebFountain cluster. In the following subsections, we present details about WF Web services and WF gateway. We provide details about WF Appliance in Section 4.

3.2 WebFountain Web Services

As described in Section 2, WebFountain uses a proprietary XML based communication protocol called Vinci. It is optimized for high performance and lacks data validity and security. Vinci could be used for inter-component communication if all the communicating components exist in a trusting environment such as in an intra net. The level of trust that is assumed in an intranet setting can not be assumed when the communicating components belong to different environments. In order to provide a secure access to the mined

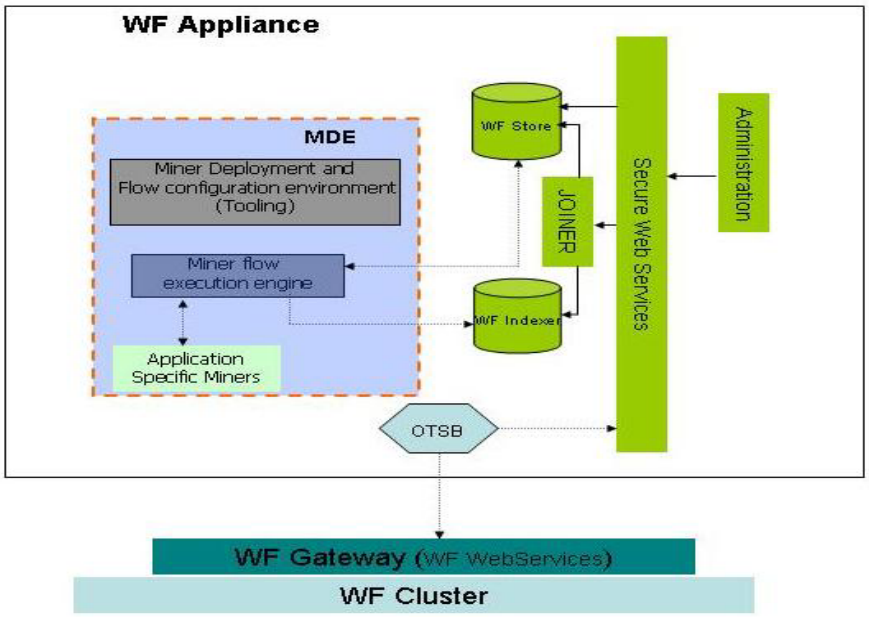

Figure 3: Overview of TAP

data on WF cluster, Vinci services provided by the different components of WebFountain are exposed to the out side world using the Web service [2] approach. Web services are programmatic interfaces used for application-to-application communication through the Web. They establish a messaging channel that shuttles XML messages back and forth. WebFountain Web services are wrapper classes that take SOAP messages as an input from the external world and invoke corresponding Vinci services. WebFountain Web services can be categorized in two different classes depending on the access rights required to use them.

- **Read Web Services:** These Web services can only read data from the WebFountain cluster and do not modify them. We have the following set of *read Web services*.
  - *Resource discovery Web service:* Application developer can view the details of all authorized resources on the WebFountain system, such as stores, keys and indices, using *Resource discovery Web service*. The users can specify a service name, or can invoke this Web service without any parameters in which case details about all the authorized services are given.
  - *GetEnumeration Web service:* With the *GetEnumeration Web Service*, a client can query the WebFountain platform. The user specifies a valid WFQL query for creating a result enumeration. Enumerations are defined as unordered streams of UEIDs traversable by an iterator.
  - *GetKeys Web services:* With the *GetKeys Web service*, a client can retrieve the key-value pairs for a given set of documents from a store. The user specifies a valid store name, a list of UEIDs, and the key names for which to obtain the values.
- **Modify Web Services:** These Web services can create, drop and modify instances of components. We have the following set of *modify Web services*.
  - *Create store Web service:* With the *Create store Web service*, a client can create an empty store. At the store creation time a user specifies a store name of the store to create and type of documents that will be stored into the store.
  - *Build index Web service* Using the *Build index Web service*, a client can index a store with a given set of keys and index configuration options. The user specifies a valid store name, the keys in the store that are to be indexed, and an index service name.
  - *Drop store Web service:* Using the *Drop store Web service*, a client can drop and clean (deallocate) a store and its associated index service or services. The user is required to specify a valid store name for dropping.
  - *Drop index Web service:* This Web service can be used to drop an index.
  - *Write to store Web service: Write to store Web service* could be used to populate an already created store. A user needs to specify a valid store name, the entities that are to be added to the store along with any keys that need to be dropped prior to adding the new keys.

3.3 WebFountain Gateway

The WebFountain gateway provides access to the mined data stored in the WebFountain cluster to application developers. *Read Web Services* given in Section 3.2, are deployed on the WebFountain gateway. This set of Web services provides only querying and reading mechanism on the WebFountain cluster, to the application developers. WebFountain gateway also ensures quality of service and provides security mechanisms. Following subsections present details on these modules.

3.3.1 Security

The Security Administration component is intended to be a facility for both, the WF Gateway as well as WF-Appliance. It manages users and their access rights to various resources. The resources on the WebFountain cluster can be categorized into stores, indices, keys (or annotations), and web services. There could be multiple stores, each containing its own set of keys and indices on the WebFountain cluster. WF Web services are also treated as resources. This enables selective access to the capabilities of WebFountain components by the different users.

With different users querying the WebFountain cluster for building an application-specific data store, it is essential to have a secured and authorized access framework in place. The security in our system is at two levels viz., (1) authentication: wherein a user supplies a user-id and password, and once authentication succeeds, (2) authorization: to check whether this user is authorized to access the data and annotations, or the invoked web services. Since users and the WebFountain cluster could be remotely located, the communication between them is over an encrypted channel which is SSL based. An Appliance or WF GateWay administrator can administer control to the resources at different levels:

- **Store level access policy:** Store level access policy could be used to allow or disallow the read and write operations on a store. A user may also be granted permission on whether he is allowed to index a store or not.
- **Index access policy:** At the index level a user may be granted permission on whether he is allowed to query the index or not.
- **Key level access policy:** Apart from the store level access policy, the access can also be controlled at a much finer level i.e., at the key level. Each key, for a particular store, has a set of access policies associated with it. These include permissions to read the value of that key, write or overwrite (if the key already exists), or drop the key using the WebService layer. The authorization at key level is granted only if the user has the required permissions for the corresponding store.
- **Web service level access policy:** As mentioned earlier, a WF Appliance and the WF Gateway expose a set of Web Services. Using the WebService access policy level an Administrator can grant authorization to users for invoking any subset of the services.

The Security Administration component allows an administrator to manage user accounts. Other components like logging and billing also call upon the services of the Security Administration component for their tasks.

3.3.2 QOS Controller

An application using WebFountain infrastructure can put significant load on the WebFountain cluster. Unlike search engine clients, a WebFountain application may consume *all* the documents that satisfy a particular query. Number of documents satisfying a typical application query may be as large as tens of million. Transferring such a large amount of data will utilize a lot of resources of the cluster and may affect quality of service to other applications. WF Gateway has a mechanism to control the cluster resource utilization by an application. This mechanism takes four parameters into consideration

- **Quality of Service Agreement:** Each application is assigned a number between 0 and 100 depending on the contract. Higher numbers signify commitment towards superior quality of service.
- **Query Complexity:** The complexity of a query is approximated by the number of *OR* and *AND* operators present in the query. Typically, higher number of query terms consume more resources of the cluster. An upper limit to the number of *AND* and *OR* terms present in a query is set for each user, depending on his agreement.
- **Current Load:** It is desirable that all applications slow down their access to the WebFountain cluster in the event of high load on the system. QOS Controller monitors the response time of various services and quantifies the load on the system as a number between 0 and 100. The number 100 suggests that the cluster is idle and 0 signifies a very high load state.
- **Application Load:** We assume that response time to a query is proportional to the load caused by the query. Therefore, longer the time it takes for the cluster to respond, more the application is penalized. QOS Controller keeps an average of the time taken by the requests from each application. Average for the next request is calculated as half of the sum of the time taken in current request and previous average.

The system tries to keep gaps of certain time interval between two subsequent requests by the application, in order to control the speed of access. The duration of this interval is calculated as a function of the average time taken by the application requests, cluster load and, the QOS agreement with the client. The function is given by the following equation:

$$f(load) = 2^{(50-load)/X} \tag{1}$$

$$interval = avg * f(QoS) * f(currLoad) \tag{2}$$

Load is assumed to be between zero and hundred. Function f takes the value 1 when *load* is 50. It increases with the decrease in the load value. X is a constant which defines the sensitivity of the load function. For every deviation in the value of load by X causes load factor to double. QoS

controller ensures that time difference between end of the last request and new request is at least equal to the interval. Otherwise it blocks the application request for some time to ensure minimum interval.

4. WF APPLIANCE

4.1 Overview

The WF Appliance contains a scaled down version of the WebFountain cluster. It hosts a local WF store and indexer. The Appliance provides a rich set of user interface to use and administer the different WF components such as store and indexer. WF Appliance also hosts a wide variety of a generic set of tools that enables customers to rapidly develop and deploy business specific text-analytics solutions. The tools are designed considering the different stages involved during the development of most of the text mining solutions. In general, text mining solutions comprise of the following stages:

- Domain specific data gathering
- Domain specific mining
- Querying, retrieving and visualizing

WF Appliance provides tools to support all of the above mentioned stages. In the following subsections, we describe these tools in detail.

4.2 Domain Specific Data Gathering

Gathering of domain specific data is one of the most difficult and time consuming step in building any text analytic solutions. WF appliance provides two different tools that could be used to build domain specific stores rapidly. Below, we provide details of these tools.

4.2.1 On-topic store builder

Goal of a text analytic applications is to extract and analyze the information available on the Web. The first task of any text analytic application is to build a data set from the Web. It is important to not contain irrelevant documents in the data set as they can introduce noise in the results. The set of irrelevant documents also take extra space and waste computing resources. Data set being analyzed by the application should also ave all the relevant pages present on the Web. Otherwise the results obtained would be incomplete. This suggests that it is important to have a high precision and recall of the relevant documents in a data store. Gathering the data set is one of the biggest hurdle in building a large scale web data mining application. Focused crawling [3] has been proposed as a solution to the problem of getting relevant documents from the Web. However, it is difficult to have a high coverage and relevancy through focused crawling because of the following reasons.

- A good set of seed URLs is required for the crawler to reach most of the relevant pages on the Web. It is very difficult to identify such seed URLs for each topic. Further, there is no way of finding out if the seeds URLs indeed cover the set of relevant documents on the Web.
- Focused crawlers have to rank the URLs without looking at its content. Hence, it is likely to make mistakes more often. Focused crawler may miss important documents, if they are conservative in crawling links. On the other hand if they follow every possible link they may get lots of irrelevant documents.

As described in Section 1, WebFountain cluster could be used for building the on-topic store. The WebFountain cluster uses a query language *Web Fountain Query Language* (WFQL). In next subsection we give a brief description of WFQL.

4.2.2 Web Fountain Query Language

Web Fountain Query Language (WFQL) is a powerful query language. Due to space constraints, we do not discuss the syntax of WFQL, but only describe some features and functionality that are relevant here. For more details, please refer to the Web Fountain Architecture[4]. Following are the three main features of WFQL:

- **Key word based search:** returns documents with the given keys words. It has support for proximity as well as phrase search.
- **Regular expressions search:** can be done on the keys that have been indexed using WILD_CARD mode. It provides capability of searching on regular expressions. As an example, to obtain all the pages that contain the word "computers" in their URL, a regular expression query, "*/computers/*" could be used.
- **Integer and boolean search:** A store may contain keys that have integer or boolean values. Integer and boolean search could be used to query such types of keys. As an example a query "(OutLinks > 100) and (isPorn = false)" would return all the pages that contain more than 100 out-going links and are not pornographic pages.

These constrains can be nested inside each other to form more complex queries. For example, a query that returns all documents that have "on demand" and "ibm" within a window of 10 words and "URL like *ibm.com" could be constructed by using key word based proximity search and regular expression search. We call a WFQL query as an *on-topic query* if it covers a topic well. For example, an *on-topic query* on automobile sector contains terms like speed, acceleration, fuel efficiency, price etc(better example). A typical on-topic query may contain hundreds of terms. We found that it is easier to come up with search terms than to identify the good seed URLs.

WF Appliance provides a tool called *on-topic store builder* (OTSB) through which one can build "on-topic store" with high coverage and precision using the WebFountain platform. Following are the steps for building an on-topic store.

1. **On-topic query building:** An "on-topic query" can be constructed using a simple user interface. The user interface offers the full richness of WFQL without requiring users to be familiar with it.
2. **On-topic store building:** Once the on-topic query has been built, user can see the sample documents.

We plan to integrate a summarizer which would get the first few thousands documents and help the user to understand the kind of documents query returns. This would help user to refine their on-topic query for better coverage and relevancy to the topic.

3. **Refreshing on-topic store:** Once an on-topic store build is complete, OTSB periodically refreshes the store at the WF Appliance end by getting newly crawled pages at the WebFountain cluster. This is achieved by adding a condition "crawldate > 'lastrun date'" in the WFQL.

4.2.3 Support for Heterogeneous Data Sources

Enterprises contain tremendous amount of structured data that is produced and consumed during different business processes. This data is stored in different databases and can be used in many text analytic applications. As an example, an application that gathers weather information for different locations of a company, would need information on addresses of all the offices of the company. This could be provided from a database table. Keeping in mind such applications, that may require inputs from structured data, WF Appliance provides the functionality of ingesting the data contained in database tables into a store.

Several database implementations such as DB2, Oracle and Microsoft SQL server are popular in industry. Instead of providing an ingesting mechanism for individual database implementations separately, WF Appliance leverages upon the XML export capability of databases and provides a service called XML Ingester. XML Ingester service takes a set of XML documents and the associated XML schema to convert the XML data into the WF document format and then pushes it into a WF store. The XML documents could be generated by XML export mechanism built in databases. XML schema file defines the different data fields that are to be ingested along with their types in WF store. As an example, an XML document may contain three data fields corresponding to *day*, *month*, and,*year*. For the text analytic application, application developer may want to view these three fields aggregated in a single field and store them in WF store with *date* type format. Such conversions are achieved by an XML schema file provided by the application developer during the ingestion process.

4.3 Domain Specific Mining

4.3.1 Overview

WF Appliance provides a component called MDE (Miner Deployment and Execution Environment) that provides support for the development and the execution of domain specific miners on the data. Figure 4 depicts the different subcomponents in MDE, and their interactions. MDE provides the following functionalities:

- Guidelines for developing the Miners
- Deploying the miners
- Defining the flow of miners.
- Executing the predefined flow of miners.

In the following subsections, we provide details on these functionalities.

4.3.2 Guidelines for Developing Miners

Miners are special-purpose programs that extract information from already stored key-value pairs of an entity and add new key-value pairs to the same or other entities. Note that the content of a document is also stored as a key-value pair. The WF infrastructure is fairly simple from a miner's perspective. Web pages are stored in an application-specific store along with other meta data in the form of key-value pairs. Enumeration (list of UEIDs) iterators provide several ways to selectively step through the data in a store. Miners read key-value pairs from a store using standard read APIs and extract pertinent information. It then writes back the mined data to the store using the standard write APIs.

4.3.3 Deploying the Miners

Deployment of a miner involves uploading the miner binary and an XML descriptor file. The XML descriptor file contains the specification of input and output of the miner in terms of the key-value pairs read and written from and to the store. It also contains the different options supported by the miner. Both, the binary version of the miner and its descriptor file can be zipped together and uploaded to the appliance for its deployment by a Web based user interface.

4.3.4 Defining the Flow of Miners

Most text analytic applications can be viewed as a set of miners run in sequence where output of a lower level miner acts as an input to another higher level miner. As an example, an application that finds CEOs of companies automatically from a set of pages, would execute a miner that identifies company names and another miner that identifies person names, followed by a miner that given a set of pages with identified company and person names, would find the relationship. Therfore, from the application development point of view, chaining of miners is an important task. MDE provides support for chaining of a set of deployed miners into a single execution unit called a *mining flow*. During the creation of a *mining flow*, the user can configure each individual miner in the flow. This makes the task of application development simple. Once a *mining flow* is defined the user can save this flow under a given name for future reuse.

4.3.5 Executing the flow of miners

A user can choose any of the predefined *mining flows* and execute it on a chosen store. MDE provides a flow execution environment called *foreman* that executes each miner in the *mining flow* as a child process. It monitors and restarts the individual miners if they crash and thus provides failure recovery. It also keep statistics on performance and error rates. Each page is passed through the chain of miners, before being written back to the store, so that the overhead of store communication is reduced.

4.4 Querying, Retrieving and Visualizing

In general, text analytic applications require ability of indexing and querying the rich annotations identified by it. One of the most preferred way of consuming this information is by retrieving a set of documents based on user queries. As described in Section 4.1, WF Appliance provides a powerful indexer that can be used to index the annotations. For retrieving a set of documents, queries can be formed using

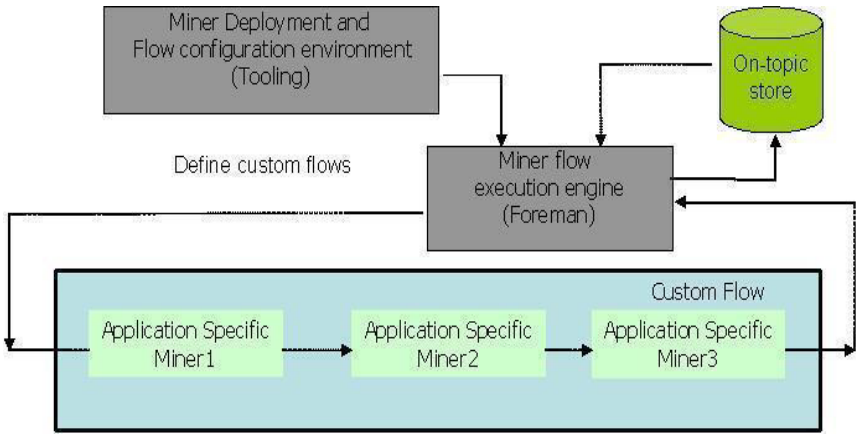

Figure 4: Miner Deployment and Execution Environment

WFQL. The set of relevant documents can be obtained by executing the given WFQL on the WF indexer. WFQL is a very rich and non-trivial query language. Users may find it difficult and cumbersome to create queries in WFQL. Therefore, WF Appliance provides a simple user interface for the creation of queries. It also provide a *query translator service* that converts a user query into the corresponding WFQL.

Typically, the set of returned documents in response to a query could be very large. Therfore, several ranking algorithms have been developed that order the returned set of documents such that the most relevant document appears on the top. WF Appliance provides a service called *search service* that given a WFQL query, uses the WF indexer to obtain the set of documents. It further uses the *tf-idf* algorithm to rank the returned set of documents. Other ranking algorithms could be plugged in the *search service*.

In many situations, putting the annotated data in databases or in OLAP servers and then performing the slicing and dicing operations on the mined data could provide valuable insight. Instead of providing a functionality of exporting the data contained in WF store to individual database implementations, WF Appliance provides a functionality called *XML export* to export the data contained in WF store to XML format. The XML can then be ingested in databases using some other third party tools. WF Appliance provides a user interface that guides an appliance user step by step to obtain an XML dump of selected portion of a store into XML files.

5. AN EXAMPLE APPLICATION

In this section, we describe the development and deployment of a sample text analytics application on TAP. The sample text analytic application that we consider here answers following types of questions:

- What are the competitors of a given company?
- Who all are the customers of a given company?
- What is the relationship of a given person with a given company?

We present the development of this application in three steps.

Step 1

In this step, the application gathers the relevant data set from the Web. It uses OTSB provided in WF Appliance for this purpose. Lets assume, that WebFounatin cluster has a *base level miner* called *company spotter* that identifies company names in documents. The application developer can formulate a query using the OTSB user interface, that returns all the documents containing at least one company name in it. The built on-topic store contains all the pages that have at least one company name in it. Note that it would have been very difficult to build such store without the WebFountain platform.

Step 2

In this step we perform domain specific mining. It is evident that, to build this application one needs to build at least 2 miners. One, that spots person names in a document and another called *relationship miner* that finds relationships between entities present in a document. Here entities could be either person names or company names.

The *relationship miner* could further be comprised of several lower level miners such as *sentence boundary detector miner*, and *coreference miner*. After developing all the required miners, MDE could be used to build a *mining flow* that contains a chaining of the miners. The *mining flow* can then be executed using MDE.

Step 3

In this step, we provide different ways of consuming the extracted information. The tagged annotations could be indexed and then *search service* could be used to answer specific queries. One does not even need to build a user interface for this functionality. The search UI provided in WF Appliance along with the *WFQL translation service* could be reused for this purpose. Alternatively, the annotations could be exported to a database or OLAP server using XML export functionality.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a platform called TAP, that

could be used by enterprises, to build text analytic applications. TAP uses WebFountain cluster to gather the application-specific data. It uses WF Web services and WF gateway for building on-topic store securely and efficiently. TAP also contains WF Appliance, which provides several tools and services that could be used for the rapid development of text analytic applications. We also provided an example text analytic application and describe how TAP could be used to build such an application rapidly. In our future work, we will look at further functionalities that could be helpful for extracting semantically rich information and visualization of data. We also plan to include the functionality of more complex conditional chaining of miners for building higher level miners.

7. ADDITIONAL AUTHORS

Ajay Dhawale (email:adhawale@in.ibm.com), Ana Lelescu (email:lelescu@us.ibm.com), Hongcheng Mi (email:hcmi@us.ibm.com), Manish Sethi (email:manishsethi@in.ibm.com), and Amit Tuli (email:tamit@in.ibm.com),

8. REFERENCES

[1] R. Agrawal, R. J. B. Jr., D. Gruhl, and S. Papadimitriou. Vinci: A service-oriented architecture for rapid development of web applications. In *10th World-Wide Web Conference (WWW10)*, pages 355–365, May 2001.

[2] B. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture*. http://www.w3.org/TR/ws-arch/, February 2004.

[3] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery, May 1999.

[4] D. Gruhl, L. Chavet, D. Gibson, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: an architecture for very large-scale text analytics. In *IBM Systems Journal*. IBM, March 2004.

[5] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. *Simple Object Access Protocol (Version 1.2)*. http://www.w3.org/TR/soap12/, June 2003.