Exploiting Evidence from Unstructured Data to Enhance Master Data Management

Karin Murthy[1] Prasad M Deshpande[1] Atreyee Dey[1] Ramanujam Halasipuram[1]
Mukesh Mohania[1] Deepak P[1] Jennifer Reed[2] Scott Schumacher[2]
[1]IBM Research - India [2]IBM Software Group - US
[1]{karinmur|prasdesh|atreyee.dey|ramanujam.s|mkmukesh|deepak.s.p@in.ibm.com}
[2]{reedj|schumacs@us.ibm.com}

ABSTRACT

Master data management (MDM) integrates data from multiple structured data sources and builds a consolidated 360-degree view of business entities such as customers and products. Today's MDM systems are not prepared to integrate information from unstructured data sources, such as news reports, emails, call-center transcripts, and chat logs. However, those unstructured data sources may contain valuable information about the same entities known to MDM from the structured data sources. Integrating information from unstructured data into MDM is challenging as textual references to existing MDM entities are often incomplete and imprecise and the additional entity information extracted from text should not impact the trustworthiness of MDM data.

In this paper, we present an architecture for making MDM text-aware and showcase its implementation as IBM InfoSphere MDM Extension for Unstructured Text Correlation, an add-on to IBM InfoSphere Master Data Management Standard Edition. We highlight how MDM benefits from additional evidence found in documents when doing entity resolution and relationship discovery. We experimentally demonstrate the feasibility of integrating information from unstructured data sources into MDM.

1. INTRODUCTION

Master data management (MDM) systems provide a consolidated view of business entities such as customers or products by integrating data from various data sources. A primary function of MDM is to identify multiple records that refer to the same "real-world entity", a process called *entity resolution* [1]. Entity resolution resolves that two records refer to the same entity despite the fact that the two records may not match perfectly. For example, two records that refer to the same person entity may contain a slightly different spelling for the person's name. Other terms used to describe the concept of entity resolution are *record linkage* [7], *record matching* [6], *identity resolution* [9], and *duplicate detection* [5].

Today's state-of-the-art MDM systems are limited to integrating and resolving data from structured data sources (see Figure 1). However, a large amount of entity information is also contained in unstructured data sources such as emails, ASR transcripts, comments, and chat logs. In fact, it is estimated that 80% of enterprise data is in unstructured form and is growing more rapidly than the structured data [21]. A global study on MDM published by PwC in November 2011 lists "converting unstructured data into MDM-compatible information" as a key challenge for the MDM of the future [16]. In this paper, we address this problem and show how MDM systems can be enhanced to leverage unstructured data from various sources (see Figure 1).

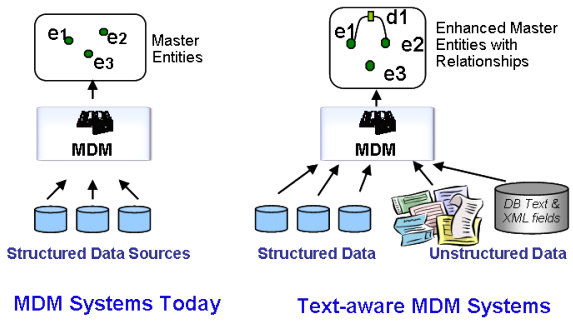


Figure 1: Evolution of MDM systems

Taking into account information from unstructured data sources has many benefits for MDM systems. In particular, we highlight *entity resolution* and *relationship discovery* as two important applications that benefit from text-aware MDM systems. We start by demonstrating how information from unstructured sources can be exploited for entity resolution.

For illustration, throughout the paper, we have picked *person* as a representative entity type. A person entity is defined by a set of atomic attributes (for example, nationality) and composed attributes (for example, a person's name which may consist of first name, middle name, and last name). To determine whether two person records refer to the same person entity, MDM compares the corresponding attribute values and computes an overall matching score for the two records. If the matching score is above a certain

Proceedings of the VLDB Endowment, Vol. 5, No. 12

threshold, MDM automatically merges the two records into a single entity.

For various reasons two records that belong to the same entity (that is, they actually refer to the same person) may not match sufficiently for MDM to automatically merge them. For example, one or both records may be incomplete or some attribute values may be incorrect. If two such records score sufficiently close to the threshold for automatic merging, MDM marks the two records for manual inspection. During manual inspection, a data analyst needs to decide whether the two records belong to the same person or not. For this task, information extracted from unstructured data may provide the missing evidence that enables the data analyst to make a decision.

Id	Name		Employer	Address		Email
	FirstName	LastName		Country	City	
1	Manoj	Patil	IBM	India	Delhi	mpatil3@in.ibm.com
2	Manish	Patel	IBM		Bangalore	m.patil@us.ibm.com
3	Tom	Smith	ABC	US	New York	
4	Tom	Smith				tom.s@abc.com
5	Sara	Lee	IBM	US		

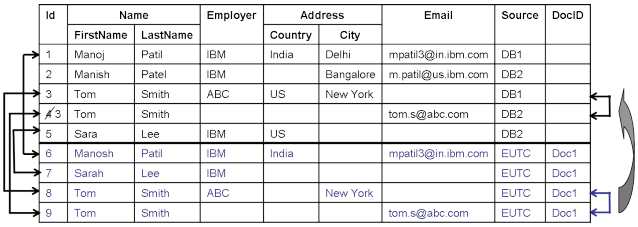
Figure 2: Entities in MDM

Manosh Patil and **Sarah Lee** from **IBM** met in **New York** with **Tom Smith** from **ABC** to discuss XYZ. The meeting took place on 21. Aug 2011.
Manosh from **IBM** in **India** is currently on a six month assignment to the office in **New York** to help **Sarah** and **Tom** with planning XYZ, a joint growth-market initiative of **IBM** and **ABC**. **Tom** is scheduled to spend considerable time in **India** later this year to oversee the execution of XYZ in **India**.
Please contact **Manosh** (**mpatil3@in.ibm.com**) or **Tom** (**tom.s@abc.com**) for further information.

Figure 3: Document linking the entities

As an illustration, consider the MDM dataset in Figure 2. It is possible that records 3 and 4 belong to the same entity, but there is not enough evidence to automatically merge them. These records remain unlinked in the MDM system. Now consider the text document shown in Figure 3 mentioning some of the MDM entities of Figure 2 (highlighted in bold). Based on existing master data information, four new person records can be extracted from the document and linked to existing entities as shown at the bottom of Figure 4. (The details of this process are described in Section 4). In the example, the extracted record 8 is linked to the existing entity 3 whereas the extracted record 9 is linked to the existing entity 4. The information that the two records 8 and 9 were extracted from the same document may be enough additional evidence for a user to decide that entities 3 and 4 pertain to the same person and should be merged into a single entity.

The second application that benefits from text-aware MDM is relationship discovery, which is the task of identifying relationships between distinct entities. Traditionally, MDM systems have focused on entity resolution and gathering all information about an entity. However, in some applications, it is also useful to identify relationships between different

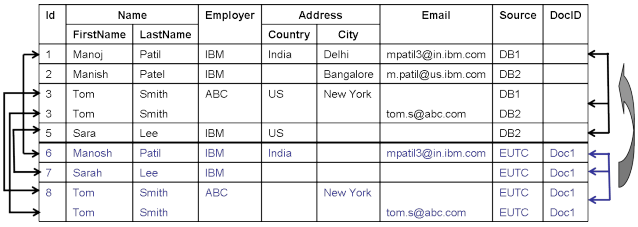


Id	Name		Employer	Address		Email	Source	DocID
	FirstName	LastName		Country	City			
1	Manoj	Patil	IBM	India	Delhi	mpatil3@in.ibm.com	DB1	
2	Manish	Patel	IBM		Bangalore	m.patil@us.ibm.com	DB2	
3	Tom	Smith	ABC	US	New York		DB1	
3	Tom	Smith				tom.s@abc.com	DB2	
5	Sara	Lee	IBM	US			DB2	
6	Manosh	Patil	IBM	India		mpatil3@in.ibm.com	EUTC	Doc1
7	Sarah	Lee	IBM				EUTC	Doc1
8	Tom	Smith	ABC		New York		EUTC	Doc1
9	Tom	Smith				tom.s@abc.com	EUTC	Doc1

Figure 4: Improved entity resolution

entities. For example, an enterprise might want to detect various kinds of relationships between its customers, for example, whether two customers belong to the same family or household. Simple relationships can often be detected based on a match on attribute values; examples include matching the last names or matching the address attribute.

Other relationships may not be as obvious and depend on the context for which the entities intersect. For example, in a public-safety scenario, the government might like to track certain suspicious entities and detect any relationships between them. Documents such as news reports, emails, or other confidential reports often contain information about multiple entities and capture that two entities interacted with each other or are related to on another and the relationship context. Text-aware MDM systems can extract these types of relationships leading to richer master data. In our example, the document shown in Figure 3 provides evidence that a relationship exists between the MDM entities 1, 3, and 5. See Figure 5 for an illustration.



Id	Name		Employer	Address		Email	Source	DocID
	FirstName	LastName		Country	City			
1	Manoj	Patil	IBM	India	Delhi	mpatil3@in.ibm.com	DB1	
2	Manish	Patel	IBM		Bangalore	m.patil@us.ibm.com	DB2	
3	Tom	Smith	ABC	US	New York		DB1	
3	Tom	Smith				tom.s@abc.com	DB2	
5	Sara	Lee	IBM	US			DB2	
6	Manosh	Patil	IBM	India		mpatil3@in.ibm.com	EUTC	Doc1
7	Sarah	Lee	IBM				EUTC	Doc1
8	Tom	Smith	ABC		New York		EUTC	Doc1
	Tom	Smith				tom.s@abc.com	EUTC	Doc1

Figure 5: Improved relationship discovery

In this paper, we describe a system that can use the above described evidence from unstructured information sources to enhance master data management. EUTC (Extension for Unstructured Text Correlation) bridges the gap between structured and unstructured data and enables MDM systems to provide a real 360-degree view of each entity. To link structured and unstructured data, EUTC automatically extracts references to existing entities from arbitrary text. The extracted entity references allow MDM systems to improve entity resolution and relationship discovery for existing master data.

EUTC addresses three main challenges:

- Text is noisy by nature and entity references are often incomplete and uncertain. Thus, the system needs to be tolerant to spelling variations, allow for fuzzy matching of values, and be able to deal with incomplete references.

- Multiple entities may be mentioned in the same document and may be referenced even within the same sentence. Thus, the system can not rely on techniques that require each entity to be mentioned within its own unit of text such as a sentence or paragraph.

- Different types of entities (for example, products or persons) are described by different attributes. And even the same type of entity may be described differently in different domains. For example, in a public-safety scenario, a person's description may include attributes such as nationality, passport number, and place of birth; whereas a human-resource scenario may include attributes such as email address, employee ID, and salary. Thus, the system should not rely on techniques that exploit domain-dependent data semantics.

EUTC addresses these three challenges and provides a generic approach to extract entity-related information from any type of document with respect to any type of MDM system domain. It leverages the probabilistic matching functionality provided by MDM systems to identify the matching entities. A specific instance of EUTC has been implemented as IBM InfoSphere MDM Extension for Unstructured Text Correlation, an add-on to IBM Initiate Master Data Service version 9.7 and IBM InfoSphere MDM Standard Edition version 10. Wherever needed, we use this implementation for explanation and experiments. However, EUTC as a concept is not limited to a specific MDM system.

The remainder of the paper is organized as follows. In Section 2, we describe the architecture of the system and run through an example execution. In Section 3, we explain how existing structured data in MDM systems is leveraged for information extraction. Section 4 describes how EUTC exploits the matching capability provided by MDM for its entity construction. We evaluate the system experimentally and present the quality and performance results in Section 5. Finally, we present some related work in Section 6 and conclude the paper in Section 7.

2. SYSTEM OVERVIEW

In this section, we introduce some MDM terminology, describe the architecture of EUTC and its individual components, and walk through an example of the execution of the EUTC process.

2.1 MDM Terminology and Concepts

We use IBM Initiate Master Data Service (MDS) for illustrations and for the experimental evaluation. Thus, the MDM terminology introduced here is influenced by the terminology used in the context of MDS.

A *member* is defined as a set of attributes that represents a type of individual (for example, a person or an organization) or a type of thing (for example, a car or a machine part). For illustration, we use the member type *Person*, which is defined by a set of demographic attributes. Figure 6 shows the snapshot of a sample MDS data model for the person member type.

A *member record* is the set of all attribute values that a single source system asserts to be true about a person. For example, each row in Figure 2 is a member record.

An *entity* is defined as "something that exists as a particular and discrete unit". In terms of data management,

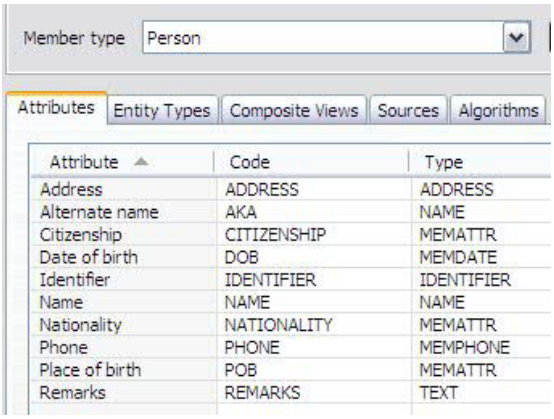


Figure 6: Attributes of person member

an entity is the logical link between two or more member records. An entity is sometimes also called a *linkage set*. For example, in Figure 5, two member records are grouped into entities 3 and 8, respectively.

Attribute Matching or Scoring is the process of comparing individual attributes using one or more appropriate comparison functions. For example, to match two person names, a phonetic comparison based on Soundex and a syntactic comparison based on edit distance may be used. The combined output of all comparison functions for matching two attribute values is called *matching score*.

Record Matching or Scoring is the process of combining the individual attribute-level scores to arrive at the likelihood that two records belong to the same entity. MDS applies a likelihood function to determine the probability that different values of an attribute match and how much weight a given attribute should contribute to the overall score between two records. This process of comparing two member records is also referred to as *probabilistic matching*. For details on MDS matching we refer the interested reader to the IBM white paper on data matching [8].

Entity resolution is the process of merging two (or more) member records into a single entity. This happens automatically if the records' matching score exceeds the *auto-link* threshold or manually if the score exceeds the *review* threshold and a user determines that the records belong to the same entity.

A *relationship* is a link between two distinct entities. For example, in Figure 5 entities 6, 7, and 8 are *directly* linked by the fact that they all appear in the same document; entities 1, 3, and 5 are *indirectly* linked by the fact that they are all linked to entities extracted from the same document.

2.2 EUTC Architecture and Components

2.2.1 Architecture

Figure 7 shows the basic architecture of EUTC. EUTC interacts both with structured and unstructured data sources. Structured data is provided by an MDM system. While EUTC works in principle with any MDM system (or for that matter any source of structured entity data), an MDM system that encompasses sophisticated methods for matching, can significantly improve EUTC's performance. (We discuss this aspect in Section 2.2.5.)

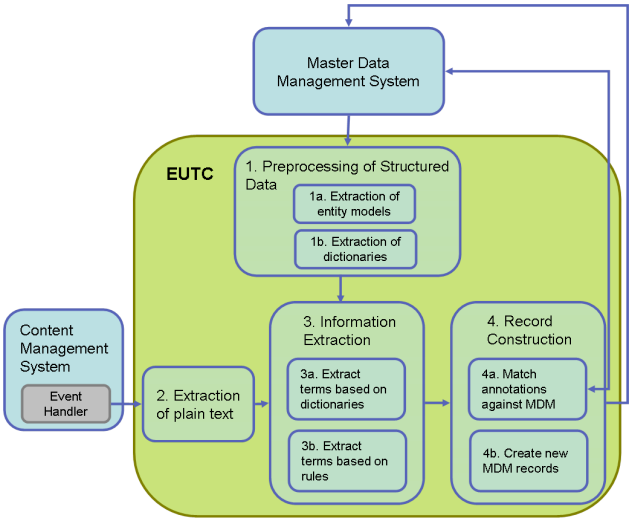


Figure 7: Architecture of EUTC

Unstructured data can come from many different sources. Content management systems such as EMC's Documentum[1] or IBM's FileNet P8[2] may invoke EUTC whenever a new document is uploaded to the document management system. However, unstructured text may also reside in the file system or be stored along with structured data as a CLOB in a database. In such cases, a separate event handler is needed to monitor the unstructured data and invoke EUTC whenever new text is available. When EUTC is first installed, it can perform bulk-processing of all existing documents.

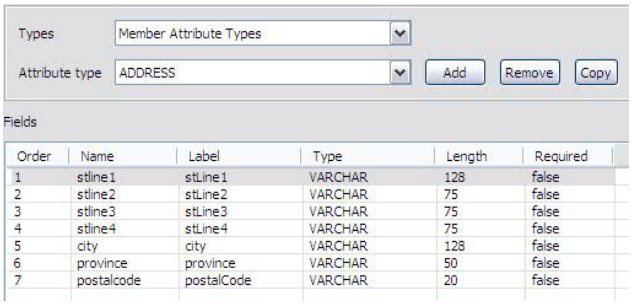
2.2.2 *Preprocessing of Structured Data*

In order for EUTC to identify references to existing entities in unstructured text, it needs to be aware of all the structured data. Thus, during configuration, EUTC extracts the data model for all members of interest from MDM (Step 1a in Figure 7). In addition, it extracts, for each atomic attribute, a dictionary with all distinct values for the attribute (Step 1b in Figure 7). For example, for the member type Person shown in Figure 6 the attribute *ADDRESS* may have seven atomic attributes as shown in Figure 8, in which case EUTC will create seven dictionaries. After configuration and setup of EUTC, dictionaries are automatically updated whenever new content in MDM creates a new dictionary entry.

2.2.3 *Extraction of Plain Text*

EUTC accepts plain text documents as well as documents in a majority of well known data formats such as PDF, MS Word and HTML (Step 2 in Figure 7). It uses functionality provided by Apache's Tika project[3] to extract the plain text. The plain text is then passed to the annotation component of EUTC. In addition to the plain text, meta data may be passed on and eventually be stored in MDM. For example, a URI may be associated with each document, allowing users of MDM to retrieve the respective document. Alternatively, the plain text of the document may be stored in MDM. Storing the document text in MDM makes it easy to re-process relevant documents when MDM data changes. It would allow the users of MDM to view the document text using traditional MDM applications that may not support activation of a URI to fetch the original document. It also supports cases where the original text cannot be made accessible by a URI.

[1]http://www.emc.com/enterprise-content-management/index.htm
[2]http://www-01.ibm.com/software/data/content-management/filenet-p8-platform
[3]http://tika.apache.org



Order	Name	Label	Type	Length	Required
1	stline1	stLine1	VARCHAR	128	false
2	stline2	stLine2	VARCHAR	75	false
3	stline3	stLine3	VARCHAR	75	false
4	stline4	stLine4	VARCHAR	75	false
5	city	city	VARCHAR	128	false
6	province	province	VARCHAR	50	false
7	postalcode	postalCode	VARCHAR	20	false

Figure 8: Definition of MDS attribute type ADDRESS

```
<element>
        <type>atomic</type>
        <name>CITIZENSHIP</name>
        <MDSAttrRecNo>8</MDSAttrRecNo>
        <queryable></queryable>
        <MDSTable>mpi_memattr</MDSTable>
        <MDSColumn>ATTRVAL</MDSColumn>
        <annotators>
                <annotator>
                        <type>EUTC</type>
                <internalDict>CITIZENSHIP-mpi_memattr-ATTRVAL</internalDict>
                </annotator>
        </annotators>
</element>
<element>
        <type>atomic</type>
        <name>DOB</name>
        <MDSAttrRecNo>9</MDSAttrRecNo>
        <queryable></queryable>
        <MDSTable>mpi_memdate</MDSTable>
        <MDSColumn>DATEVAL</MDSColumn>
        <annotators>
                <annotator>
                        <type>EUTC</type>
                        <rule>date</rule>
                </annotator>
        </annotators>
```

Figure 9: Part of EUTC configuration file

2.2.4 *Information Extraction*

By default each attribute is associated with a dictionary and EUTC uses fuzzy matching to extract terms in the text that match a dictionary entry (Step 3a in Figure 7). Figure 9 shows part of the EUTC configuration file where a dictionary has been automatically associated with the attribute CITIZENSHIP. Section 3 discusses the details of how those dictionaries are used to find all matching terms within the text.

Obviously, dictionary-based annotation may not be appropriate for all attribute types. For such cases, EUTC uses rule-based information extraction (Step 3b in Figure

7). For example, there are so many variations of writing a date that using a dictionary to annotate all instances of dates is not appropriate. Thus, EUTC automatically detects whether an attribute is of type *date* and associates the appropriate rule-based annotator with the date attribute. Figure 9 shows part of the EUTC configuration file where a rule-based annotator is associated with the attribute Date of Birth (DOB).

Note that, so far all information extraction is completely domain-independent and does not require any customization. This is in stark contrast to existing solutions where months of effort may be spent to develop appropriate annotators for each domain and setting. However, if specialized annotators have already been developed, they can be easily plugged into the EUTC configuration. EUTC's information extraction component is built on top of Apache's Unstructured Information Management Architecture (UIMA) framework[4] and allows easy integration of UIMA-compliant custom annotators.

2.2.5 *Record Construction*

EUTC needs to determine which entities in the MDM system might be referenced within the document using the information it extracted from the document in the form of attribute-value pair annotations. A naive approach is to enumerate all possible combinations of annotations and query MDM for exact matches. If a combination yields a single match with MDM, the corresponding annotations are likely to be a reference to the matched entity. However, this approach neither scales nor does it account for the uncertainty associated with information extracted from text.

A key observation is that, given a set of attribute-value pairs, finding an entity that matches it is a primary functionality provided by MDM. Thus, rather than implementing the matching ourselves, we exploit the sophisticated matching capabilities provide by MDM systems. In this paper, we specifically describe how probabilistic matching (for example, as provided by MDS) is used to infer which entities are likely matches to the set of all annotations extracted from the document. Section 4.1 discusses the details.

Based on the results retrieved from MDM, EUTC creates member records by computing the overlap between the values of a returned MDS record and the values in the annotation set. See Section 4.2 for details. For each member record EUTC creates, it keeps track of which existing MDM entity it matched and with which score. The records are then provided to MDM. Section 4.3 discusses how MDM consumes the extracted records.

Attributes

View configured attributes and attribute history.

Attribute	Value
Name	Miran Mada
Address	Via Riasc 21 Compione Italy
Citizenship	United States
Place of birth	Portland, Oregon
Date of birth	1938-09-15

Figure 10: Attribute view of MDS entity 1574

2.3 Example EUTC Execution

We show an example from a public-safety scenario where the MDM system contains a large amount of potential suspects collected from multiple data sources. Each person in MDM is described by the attributes listed in Figure 6. A common task for an analyst is to gather all available information about a suspect and examine any connections to other suspects.

Assume that the analyst is interested in a person called Miran Mada. She may use the IBM Initiate Inspector[5] application to find out everything MDS knows about her suspect. Figure 10 shows the attribute view for the MDS entity associated with Miran Mada (to which MDS assigned the identifier 1574). When exploring the relationship view, the analyst finds out that there are no known relationships with other entities.

Now consider the document shown in Figure 11, whose made-up content is representative for documents we observed in the public-safety scenario. This document establishes a relationship between the suspect and another entity called Maranda Group of Companies.

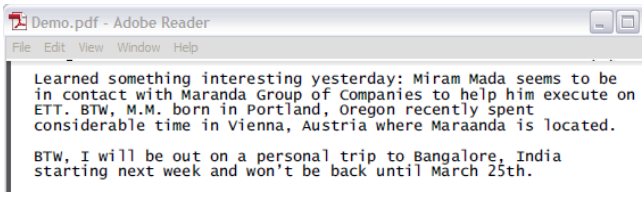


Figure 11: Sample document for public-safety scenario

IBM InfoSphere Master Data Management Extension for Unstructured Text Correlation

File Processed:Demo.pdf

Annotation Type	Annotated Text
CITIZENSHIP	India
NATIONALITY	India
POB	Portland, Oregon
POB	India
ADDRESS.CITY	Maranda
ADDRESS.CITY	Vienna
ADDRESS.COUNTRY	Austria
ADDRESS.CITY	Maraanda
ADDRESS.CITY	Bangalore
ADDRESS.COUNTRY	India
ADDRESS.PROVINCE	Portland
AKA.NAME1	Miram
AKA.NAME2	Miram
AKA.NAME2	Maranda
IDENTIFIER.ISSUER	Austria
IDENTIFIER.ISSUER	India
NAME.NAME1	Miram
NAME.NAME2	Mada
NAME.NAME1	Maranda Group of Companies
NAME.NAME1	Maranda
NAME.NAME1	Maraanda

Annotated File Contents

Miram Mada seems to be in contact with **Maranda Group of Companies** to help him execute on ETT. BTW, M.M. born in **Portland, Oregon** recently spent considerable time in **Vienna, Austria** where **Maraanda** is located. BTW, I will be out on a personal trip to **Bangalore, India** starting next week and won't be back until March 25th.

 # Records Constructed: 2

Record #	Type	Text
1	MEID	1574
	CONFIDENCE	67
	DOCID	Demo.pdf
	NAME.NAME1	Miram
	NAME.NAME2	Mada
	POB	Portland, Oregon
Record #	**Type**	**Text**
2	MEID	3652
	CONFIDENCE	50
	DOCID	Demo.pdf
	ADDRESS.CITY	Vienna
	ADDRESS.COUNTRY	Austria
	NAME.NAME1	Maranda Group of Companies

Figure 12: Illustration of EUTC execution

[4]http://uima.apache.org

[5]www.ibm.com/software/data/infosphere/inspector

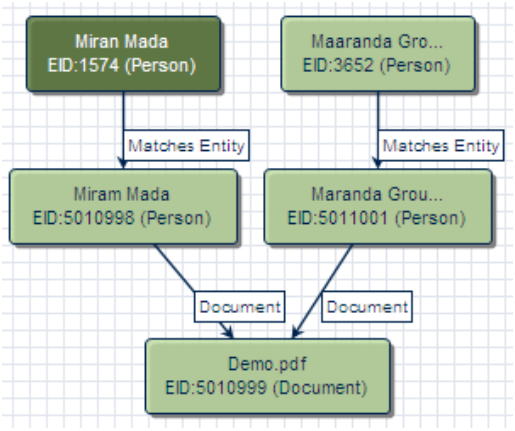


Figure 13: Relationships established by EUTC

Figure 12 illustrates the execution of EUTC over the sample document. The annotations received when executing information extraction are shown in the left column of Figure 12. The annotations are also highlighted in the text shown to the right. Note that, not all annotations pertain to the two entities mentioned in the document (Miram Mada and Maranda Group of Companies) and the first name of the suspect entity is spelled differently in the document as in MDS (Miram versus Miran). Nevertheless, EUTC correctly identifies a reference to the suspect entity 1574 as well as to another existing entity with the identifier 3652, as shown in Figure 12 (right side, bottom). Based on this, EUTC inserts two new member records into MDS. The analyst can now explore the newly established relationship in the Inspector application (see Figure 13).

3. EXPLOITING STRUCTURED DATA FOR INFORMATION EXTRACTION

In this section, we describe our domain-independent approach to extract entity-related information from unstructured text documents.

3.1 Dictionary-based Matching

The basic idea behind EUTC is that any reference in a document to an existing MDM entity contains information known to MDM. For example, the occurrence of *Bangalore* in a document is only relevant, if there is at least one entity in MDM that is related to Bangalore (for example, a person entity may live in Bangalore or may have been born in Bangalore). We exploit this fact and create a dictionary for each atomic MDM attribute. Each dictionary contains all the distinct values for the attribute across all known MDM entities. For example, for the MDM instance in Figure 2, the *FirstName* dictionary is: {*Manoj, Manish, Tom, Sara*}. Every mention of a dictionary entry in a text, is a potential clue that an existing entity may be referenced. For example, the occurrence of the first name *Sara* in a document, can be evidence that the document talks about entity 5 in Figure 2.

By default, EUTC treats all attributes as strings and applies dictionary-based matching. As discussed in Section 2.2.4 some attributes are not amenable to dictionary-based matching in which case EUTC applies rule-based information extraction instead. The remainder of this section focuses on dictionary-based matching.

3.2 Approximate Matching

When identifying potential clues, EUTC needs to be robust to noise commonly associated with text data. For example, we do not want to miss out on a reference to *Sara Lee* (entity 5 in Figure 2) just because the person may be spelled *Sarah Lee* in a document. Typical kinds of noise in text data such as emails and web documents include spelling errors, alternative transliterations of names of non-English origin, abbreviations, vowel dropping, and non-standard words[6].

In order to accommodate such noise during dictionary-based matching we use edit distance [10] (aka Levenshtein distance) to determine whether a dictionary entry is mentioned the a text. Despite many advances in approximate string matching, Levenshtein distance remains a popular metric for identifying approximate matches [13]. Note that, for EUTC purposes, we do not care too much about false matches as those are filtered out in later stages of EUTC. However, we do care that potential matches with a dictionary are detected in the text.

3.3 Efficient Matching

Computing the edit distance between every substring in the text and every value in each dictionary is prohibitively expensive. Thus, we include a common character-3-gram constraint in our approximate matching semantics which can be checked efficiently. We use $3grams(str)$ to denote the set of all contiguous 3 character substrings of a string str; for example, $3grams(\text{“}sample\text{”})$ evaluates to $\{sam, amp, mpl, ple\}$.

We consider a string str_1 an approximate match of the string str_2, if **both** of the following conditions are satisfied:

- $3grams(str_1) \cap 3grams(str_2) \neq \phi$, that is, there is at least one common character-3-gram between the strings
- $ed(str_1, str_2) \leq min\{d_{max}, \frac{length(str_1)}{df}, \frac{length(str_2)}{df}\}$, where d_{max} (by default set to 4) provides an upper bound on the allowed edit distance and df (by default set to 4) controls the allowed edit distance as a fraction of the length of the shorter string.

The edit distance threshold is set to at most d_{max} to avoid spurious matches between long strings. Our experiments show that using the default values, a combination of the above two conditions leads to fairly accurate extraction of entity-related information.

To aid fast verification of the first condition, we create an in-memory inverted index on character 3-grams for each attribute dictionary. A subset of such an index for the *FirstName* attribute values listed in Figure 2 is shown in Figure 14. We implement the index as a HashMap[7] that maps each character 3-gram to a list of all entries containing the character 3-gram. We do not create any additional index structure to aid edit-distance computation.

[6]http://en.wikipedia.org/wiki/Noisy_text_analytics
[7]http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html

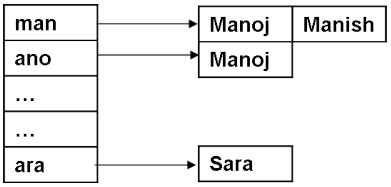


Figure 14: Inverted 3-gram index for *FirstName*

Alg. 1 ***Approximate Matching Overview***

Input. Document $\mathcal{D}$
Input. Set of Inverted Indexes $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m\}$
Output. Set of matches as 3-tuples, $[matched, \mathcal{I}, entry]$, each tuple indicating that the string *matched* in the document $\mathcal{D}$ was matched with the entry *entry* in the inverted index $\mathcal{I}$

1. $\mathcal{R} = \phi$
2. Split $\mathcal{D}$ into component tokens $\{w_1, \ldots, w_n\}$
3. $\forall w \in \{w_1, \ldots, w_n\}$
4. $\quad \forall \mathcal{I} \in \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m\}$
5. $\quad\quad m_{\mathcal{I}}^w = \phi$
6. $\quad\quad \forall t \in 3grams(w)$
7. $\quad\quad\quad m_{\mathcal{I}}^w = m_{\mathcal{I}}^w \cup entries(\mathcal{I}, t)$
8. $\quad\quad \forall m \in m_{\mathcal{I}}^w$
9. $\quad\quad\quad \forall k \ from \ (\#tokens(m) + 2) \ to \ 0$
10. $\quad\quad\quad\quad c = [w, \ldots, w_k]$
11. $\quad\quad\quad\quad if(ed(m, c) \leq min\{d_{max}, \frac{|m|}{df}, \frac{|c|)}{df}\})$
12. $\quad\quad\quad\quad\quad \mathcal{R} = \mathcal{R} \cup \{[c, \mathcal{I}, m]\}$
13. $\quad\quad\quad\quad\quad$ break
14. Return $\mathcal{R}$

3.4 Information Extraction using Approximate Matching

We now describe the complete algorithm for annotating a text with all approximately matching entries in the MDM attribute dictionaries. An overview of the algorithm appears in Algorithm 1. The algorithm takes an inverted index $\mathcal{I}_i$, for each attribute i and the document $\mathcal{D}$ as input. It identifies every occurrence of an approximate match with a dictionary entry and adds it to the result set in the form of a 3-tuple $[matched, \mathcal{I}, entry]$ where the string *matched* in the text was matched with the string *entry* in the attribute dictionary index by $\mathcal{I}$. Note that, dictionary entries (that is, values of an attribute) may not always be single tokens. For example, the value $New\ York$ in Figure 2 for the attribute *City* consists of two tokens; the company name IBM could have been written as *International Business Machines Corporation*.

The algorithm starts with an empty result set $\mathcal{R}$. It first tokenizes the input document into tokens $\{w_1, w_2, \ldots, w_n\}$ (Line 2). For every token and index pair $(w, \mathcal{I})$, we identify all entries in $\mathcal{I}$ that share at least one 3-gram with w. In lines 6-7 all such entries are collected in $m_{\mathcal{I}}^w$. For all entries that satisfy the 3-gram criterion, we proceed to identify those that also satisfy the edit-distance criterion. As mentioned above, entries in $m_{\mathcal{I}}^w$ may consist of multiple tokens. Since an entry is likely to match with a token sequence in the text of similar length, we start by comparing the entry to the string comprising of $\#tokens(m) + 2$ starting with the token w to check for the edit distance criterion (Line 9). We progressively shrink the sequence of tokens by dropping a token at the end of the sequence, until we find a match or we run out of tokens to drop; at any point, if the edit-distance criterion is met, we stop the search and add the appropriate 3-tuple to the result (Line 12).

Consider the text *"... NewYork press has reported ..."* where *New York* is incorrectly spelled without the whitespace in between. When considering $w =$ *"NewYork"*, the algorithm identifies the *city* dictionary entry *"New York"* as a candidate match due to the common 3-gram *"New"*. Since $\#tokens(\text{"}New\ York\text{"}) = 2$, it considers up to 4 tokens starting from *NewYork* to compare with. The edit-distance criterion is obviously not satisfied for the pairs [*"New York"*, *"NewYork press has reported"*], [*"New York"*, *"NewYork press has"*] and [*"New York"*, *"NewYork press"*]. Only after dropping one more token, the edit-distance criterion is satisfied for the pair [*"New York"*, *"NewYork"*] (with an edit-distance of 1) and the match is added to $\mathcal{R}$. This showcases how successively dropping tokens makes the matching technique tolerant to missing whitespace. Analogously, starting the matching at a length of $(\#tokens(m) + 2)$ instead of $\#tokens(m)$ makes the matching tolerant to spurious whitespace characters.

3.5 Matching Composed Attributes

Until now, we have only considered atomic attributes. However, attributes are often hierarchical in nature (see for example, the attributes *Name* and *Address* in Figure 2). For example, *Name* is composed of two attributes *FirstName* and *LastName*. For the text *"... Sarah Lee ..."*, the matching algorithm returns a set with [*"Sarah"*, *FirstName*, *"Sara"*] and [*"Lee"*, *LastName*, *"Lee"*].[8] We now want to combine the atomic attribute matches into a single composed-attribute match for *Name* denoted by [*"Sarah Lee"*, *Name*, *"Sara Lee"*].

In general, consider a composed attribute $\mathcal{A}$ with p ordered atomic attributes $[\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p]$. We consider a sequence of atomic attribute matches $[m_1, m_2, \ldots, m_q]$ $(q \leq p)$ as a match for the composed attribute $\mathcal{A}$ if **all** the following conditions hold:

- $\forall i, 1 \leq i < q, \quad cont(m_i, m_{i+1})$ where $cont(m_i, m_{i+1})$ indicates whether the matches appear contiguous in the text. We consider two strings to be contiguous if no more than three arbitrary characters appear in between.
- $\forall i, 1 \leq i < q, \quad attribute(m_i) < attribute(m_{i+1})$. That is the atomic attributes associated with the matches appear in the same order as the attributes associated with the composed attribute.
- $\nexists m'$ *s.t.* $\exists i$ *where* $[m_1, \ldots, m_i, m', m_{i+1}, \ldots, m_q]$ is a match for $\mathcal{A}$. This condition ensures that we only consider the maximal match.
- $q \geq 1$. This ensures that at least one component attribute of $\mathcal{A}$ matches.

In short, a sequence of atomic attribute matches is regarded a match for a composed attribute if it contains *some* of its component attributes in the same sequence, the matches appear contiguously in the text, and no more matches may

[8]For convenience, we use the attribute name instead of the index name in the second element of the tuple.

be included. If non-maximal matches were also considered, *"Sarah", "Lee", "Sarah Lee"* would all be considered as matches for *Name*, leading to redundancy. Since we do not enforce that all component attributes be covered, a composed attribute *Address* that consists of atomic attributes *Line1, Line2, Line3* may have a match that contains a 2-length contiguous sequence comprising of a match from $Line1$ followed by one from $Line3$.

Given all the matches for each $\mathcal{A}_i$, exhaustively searching for all possible matches of $\mathcal{A}$ leads to an exploration of an exponential search space. Thus, we adopt a greedy strategy and allow a match for a component attribute to figure only once in a match for $\mathcal{A}$. We start from each match of $\mathcal{A}_1$ as a single-element sequence and iterate over the other attributes in order, adding matches from them to the sequence as long as they are contiguous in the text. Each sequence is reported as and when found and the component matches are excluded from further consideration. When the matches of $\mathcal{A}_1$ are exhausted, we proceed to $\mathcal{A}_2$ and so on until all component attributes of $\mathcal{A}$ are covered. This greedy strategy may potentially miss some valid matches for composed attributes. However, our experiments show that such cases rarely appear.

4. RECORD CONSTRUCTION

This section describes how EUTC exploits existing MDM capabilities to find entity references in the text based on the extracted annotations.

4.1 Querying MDS

Once all annotations have been collected as described in Section 3, EUTC exploits MDS's probabilistic matching to retrieve relevant records from MDS. As briefly described in Section 2.1 MDS uses probabilistic matching to determine how likely two member records refer to the same entity. The same probabilistic matching is used to answer queries to MDS. A query specifies a set of values for some MDS attributes. For example, it may ask for member records were NAME1="Tom", NAME2="Smith", and NATIONALITY = ("US" or "Canada"). Note that MDS matches a query against member records (and not entities). However, the user may choose to receive the results grouped by entities.

Results are returned ranked by matching score. For example, MDS may return the records listed in Table 1, where the first result record has a high matching score because all three attribute values contribute to the overall score. If multiple values are specified for the same attribute, MDS picks the highest matching score it achieved for any of the values (for example, for result record one and attribute NATIONALITY, it would have picked the matching score for "US"). The second record is assigned a lower score as only two attributes contribute to the overall score because the nationality of the MDS entity is unknown.

To decide how much a match for a certain attribute contributes to the overall matching score, MDS employs weights. For example, if a rare name matches, the match is scored higher than if a common name such as "Tom Smith" matches. MDS also uses negative weights to penalize non-matching attribute values. For example, the third result record is assigned a very low score because this person's nationality does not match any of the nationalities specified in the query.

EUTC queries MDS with all the annotations extracted from a document to retrieve all member records that match the annotation set with a high enough score. However, as EUTC may collect information about multiple entities in a single query, it does not employ negative weights. In the above query, EUTC would not penalize the last record because its nationality does not match any of the specified values. It is possible that a document never mentions the nationality of Tom Smith but does mention the nationality of completely unrelated entities. Thus, EUTC only collects positive evidence but does not penalize for non-matching attribute values.

Table 1: Ranked MDS Query Results

NAME1	NAME2	NATIONALITY	...	Score
Tom	Smith	US	...	high
Tom	Smith	unknown	...	medium
Tom	Smith	Great Britain	...	very low

Note that it is possible that the EUTC query could contain information from multiple entities whose combination of attributes will match an entity in MDS that is not referenced within the document. However, we show in our experimental results that at higher matching scores such a result is less likely to occur. Our experiments also show that the probabilistic matching offsets the imprecision in EUTC's annotations. For example, MDS may contain a person with the first name "Said" and a document may contain text such as "XYZ said ABC". In this case, EUTC will incorrectly add an annotation for NAME.NAME1 = "said". However, MDS will assigns a low score to any record where nothing beyond the first name annotation matches.

When processing a document, EUTC needs to strike a balance between recall (that is, all references to an existing entity are found) and precision (that is, all references found are indeed correct references to an existing entity). In order to do so, EUTC enforces a user-specified minimum threshold when querying MDS. Only results whose matching score is above the threshold are returned by MDS. This threshold is closely related to the auto-link and review thresholds (outlined in Section 2.1) specified for entity resolution inside MDS. In Section 5.3 we show how to choose a good threshold based on a small set of labeled data.

In certain cases, the user may also need to make adjustments to the weights to avoid spurious matches due to the fact that in EUTC negative matches are not penalized. In order to compensate, distinguishing attributes such as a name or email should be given more weight than non-distinguishing attributes such as citizenship or place of birth.

4.2 Processing MDS Query Results

After EUTC receives all the records that matched the annotation set, it iterates over each MDS record and computes the overlap between the MDS record and the annotation set. It employs the same fuzzy matching (as during annotation) to compare values of the same attribute type against each other. It constructs an EUTC record which contains the best matching annotation for each attribute type. If an annotation for a composed attribute contains only one atomic attribute value, but the MDS record contains multiple atomic attribute values, EUTC discards the match. For example, the annotation (NAME.NAME1 = "said") is not added to the EUTC record, if the corresponding MDS record contains two values (NAME.NAME1= "Said" and

NAME.NAME2="Lastname"). However, the annotation (NAME.NAME1 = "XYZ Company") is added to the EUTC record, if the corresponding MDS record contains only one value (NAME.NAME1="XYZ Company"). The same logic is applied for all composed attribute types.

4.3 Providing EUTC Results

The records constructed by EUTC are inserted into MDS as new member records. Additionally, each EUTC record is linked explicitly to the referenced MDS entity (see Figure 13 for an illustration). Each EUTC record also has an additional attribute DOCID that contains a URI to the document from which the record was extracted. MDS automatically creates a document entity for the attribute DOCID and links all records with the same document URI to the same document entity (see Figure 13 for an illustration).

MDS maintains a source code attribute to distinguish between records from different data sources. Thus, EUTC records can easily be distinguished from records received from various structured data sources. No merging of EUTC records with other MDS records is enabled. Instead, as described above, an EUTC record is explicitly linked (by a relationship) to the corresponding MDS record. This allows applications that depend on highly trusted data to use only member records from traditional data sources, whereas more exploratory applications can use all member records independent of their source.

Alternatively, EUTC records could be handled in the same manner as those records coming from structured data sources. In that case, explicit linking is unnecessary as MDS automatically merges records that score above a certain threshold into the same entity.

As briefly outlined in Section 2.1, MDS employs two different thresholds in order to decide when to merge automatically and when to involve user interaction. Similarly, EUTC accepts two different thresholds in order to determine when to insert an EUTC record into MDS and when to store the record outside MDS for review. If both thresholds are set to the same value, all records with a score above the threshold are inserted into MDS. If the two threshold values differ, EUTC records which score between the two thresholds are inserted into a separate data source outside of MDS. This allows MDS to maintain its status as a trusted data source, while it allows analysts to reach out and gather more information about an entity from the external data source.

5. EXPERIMENTAL EVALUATION

In this section, we report experimental results and an analysis of EUTC's performance.

5.1 Experimental Setup

We have implemented EUTC using Java version 1.5. On the MDM side, we have setup an Initiate MDS Version 9.7 instance running DB2 Version 9.7 for data storage. On the content side, we have implemented an event handler that monitors a specific folder in the file system and invokes EUTC whenever a new document is added to the folder.

For the experiments, all three components (MDS, EUTC, and the event handler) were running on the same machine. All experiments were run on a machine provisioned from the cloud with two Intel Xeon E5640 processors running at 2.67 GHz. The virtual machine was allocated 4GB RAM and was running Windows server 2003, service pack 2.

Table 2: Document Data Characteristics

	Minimum	Maximum	Average
Characters	417	4,013	2,554
Entity References	2	13	5.6

5.2 Data

Our experiments are based on a confidential real-world MDS dataset from a public-safety scenario. The dataset consolidates person data from two different sources with 4,865 and 488 member records, respectively. The combined 5,173 member records were grouped into 4,919 entities by MDS's automatic entity resolution. A subset of the member records have been marked as potential duplicates and tasked for manual review. The thresholds for auto-linking and manual review were determined based on a user-specified sample of records that must link or should not link.

The attributes for a person in this dataset are listed in Figure 6. All records contain a name; 75% contain an address; approximately 50% contain identifier information such as passport or license numbers, a date of birth, and an alias name; all other attributes appear in less than 30% of the records. Note that many records contain multiple values for the same attribute. For example, it is common for a record to contain a large number of aliases for the AKA attribute.

We also had access to a representative set of real-world documents of reports related to the entities in the MDS data set. We manually examined those documents to identify references to existing MDS entities. Each document was manually annotated with the IDs of all entities in MDS referenced by the document. This process was done by two people and only if both people agreed on the annotations, the document was added to the test set. The test set consists of 50 documents in which on average each document contains about 2,600 characters and references 5.6 MDS entities (see Table 2 for more details).

In other real-world scenarios, the number of MDM records may be much larger. In order to demonstrate the scalability of EUTC, we generated additional member records with a distribution similar to the 5,173 real-world member records. We used the dictionaries extracted from the existing MDS data to generate values for the new members. To avoid a disproportional increase in false positives, exact name matches were excluded from the name dictionaries. Based on the generated data, we prepared two additional datasets by adding 25,000 and 50,000 records, respectively, to the original 5,137 records.

5.3 Qualitative Analysis

The most important task of EUTC is to correctly identify references to existing entities. We measure how well EUTC does for this task with respect to the test set described in Section 5.2. We ran EUTC over the test set using various threshold values and then compared the automatically extracted references with those assigned during the manual process. We measure the quality of EUTC in terms of precision, recall, and F-measure for various thresholds. As MDS is implemented to provide trusted data, customers are reluctant to introduce unreliable data. This causes them to favor precision over recall. Thus we use $\alpha = 0.5$ for the weighted F-measure

$$F_\alpha = (1+\alpha)\frac{precision*recall}{\alpha*precision+recall}.$$

As shown in Table 3, the maximum recall EUTC achieves is 90%. This is due to EUTC's generic, domain-independent approach to annotating entity-information. For example, EUTC does not consider "The Cooperative Import Export Enterprise" (contained in a text) and "Co-operative Export-Import Enterprise" (name value of an MDS entity) as similar. Thus, EUTC is unable to detect a match in the respective text with the value of an existing MDS entity and misses to annotate the respective name. Notice that MDS itself would match the two name variants with a high score. If a customer needs to improve recall for a particular attribute, such as name, a custom annotator can be implemented and used by EUTC.

EUTC achieves perfect precision for a threshold of 130. But perfect precision comes at the cost of a significant drop in recall to 46%. However, even if EUTC detects less then half the entities referenced by text, it is still adding new information into MDS, thus improving the value of MDS for the customer. Better recall with a still decent precision is achieved for a threshold of 100 where EUTC achieves over 90% precision with a recall of almost 75%. In general, we suggest using the threshold with the highest F-measure (100 for this dataset) as the review threshold as it achieves the best balance between recall and precision. We suggest using a threshold that achieves a precision close to 100% (130 for this dataset) as the insert threshold to preserve the quality of MDS data.

Table 3: Precision and Recall of EUTC

Threshold	Recall	Precision	$F_{0.5}$-Measure
50	0.90	0.13	0.18
60	0.81	0.16	0.22
70	0.79	0.35	0.43
80	0.77	0.62	0.66
90	0.76	0.86	0.82
100	0.73	0.93	0.85
110	0.65	0.96	0.83
120	0.58	0.97	0.79
130	0.46	1.00	0.72

5.4 Comparison with Naive Approach

A naive approach to find entity references in text is to just rely on a person's name, ignoring all other attributes. We tested to what extent finding all exact matches of a person's name in the text, helped in correctly identifying entity references. We used all name-related dictionaries extracted from MDS and annotated all exact matches in a document. Wherever possible we used the algorithm described in Section 3.5 to combine the name annotations to create a full name. We performed an exact text search for the full name in MDS for the NAME and AKA attribute, respectively.

Using an exact name match yields a recall of 48% and a precision of 91%. At a similar precision EUTC achieves a considerably higher recall of 73%. To improve recall for the naive approach, we also ran some experiments allowing fuzzy matching for the name. However, even at very low levels of fuzziness, the precision dropped to unacceptably low levels. The reason the naive approach works at all, is that our specific dataset is well curated and different variants of a name have already been captured in the alias attribute. Also, due to the size of the dataset, the number of person entities that shared the same name was not statistically significant. Many real-world MDM instances, such as the customer base of a large bank, contain multiple persons with the same name and instances where the same person's name is spelled differently. In such a case, the precision of the naive approach will drop rapidly.

Also, in the naive approach the user has no way to reduce the number of false matches and must accept the achieved precision as final. EUTC instead allows the user to specify a matching threshold such that the achieved precision is compatible with the intended use of the extracted entity information.

Table 4: Breakdown of Runtime (in sec)

	original dataset	original + 25K	original + 50K
Annotation	0.17	0.21	0.20
Member Search	0.82	4.36	7.65
Member Score	0.02	0.03	0.03
Member Details	0.03	0.05	0.05
Record Construction	0.07	0.19	0.27
Member Insert	0.71	0.81	0.92
Overall	1.82	5.65	9.13

5.5 Performance

We report the performance of EUTC as the time taken per document for the original dataset, the dataset with 25,000 additional records, and the dataset with 50,000 additional records. Table 4 shows the time taken for running EUTC with an insert threshold of 130 and a review threshold of 100. The parameters were chosen based on the analysis in Section 5.3. Note that the reported numbers only reflect an upper bound on what can be expected in a customer setting. The testing environment in which the performance tests were run is not optimized for the performance of MDS in terms of hardware or system settings.

EUTC performs five different API calls to MDS. *Member Search* retrieves a list of all member record that match the annotation query. This is the most time consuming step as MDS needs to figure out which records in MDS score above the threshold against the set of annotations. Once the member records are returned by MDS, *Member Score* retrieves a breakdown of the matching score for all returned records. This breakdown is inserted into MDS later on as an attribute of the relationship link between the extracted entity and the MDS entity. For each returned record, *Member Details* retrieves the corresponding entity ID. Finally, *Member Insert* includes two API calls, one to insert the extracted member records and one to establish the relationship between each extracted entity and its matching MDS entity.

In terms of components external to MDS, only *Annotation* takes significant amount of time. The remaining tasks (summarized in *Record Construction*) such as extracting the text from a document or processing results returned by MDS contribute very little to the run time.

Figure 15 shows the distribution of runtime over the different components of EUTC. For the original dataset, the combined time for matching the annotations with MDS entities and inserting the extracted records into MDS dominate

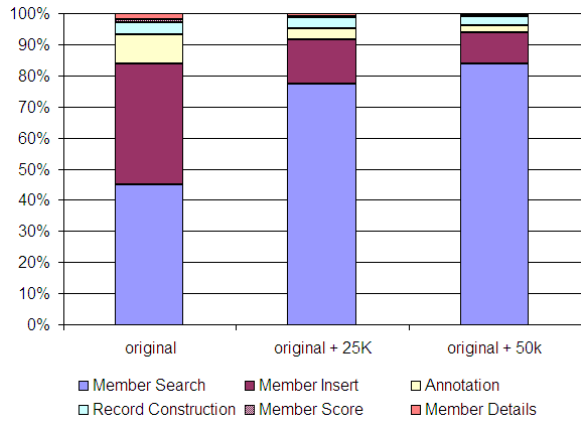


Figure 15: Distribution of EUTC runtime

the overall runtime. For larger MDS datasets, the annotation time contribute little to the overall processing time. Also, for larger MDS datasets, the time for querying MDS to retrieve the matching entities dominates the overall runtime. Overall, it takes between 1.8 and 9 seconds to process a document. It is important to stress that the processing time will be lower in a customer setting where MDS has been optimized for performance.

6. RELATED WORK

We now provide a brief overview of literature related to the specific tasks accomplished by EUTC.

Named-entity Recognition: There is ample literature on named-entity recognition, that is, the task of finding named-entity mentions in text. Note that, in this context, the term *named entity* is used to denote a single attribute of an entity in the MDM context. The approaches for named-entity extraction can broadly be classified into three categories: dictionary-based, learning-based, and rule-based.

Dictionary-based approaches find approximate matches in a text with respect to a large dictionary of known named-entity values (see [22, 12, 11, 18] for some recent papers on the topic). Learning-based approaches (for example, conditional random fields [20]) use text data labeled with named entities to learn how to extract named entities. Learning-based approaches may also be combined with dictionary-based approaches [4]. In rule-based approaches such as SystemT [3], a human being specifies how to extract named entities. Rule-based approaches complement dictionary-based extraction by finding named-entity mentions that might not be covered by the dictionary.

Note that most of these approaches may improve upon EUTC's named-entity extraction process described in Section 3. However, the goal of EUTC is to provide a domain-independent, out-of-the-box approach to integrating any unstructured data with any type of master data repository. Thus, EUTC intentionally employs a very generic approach to information extraction. However, EUTC includes the capability for a customer to extend upon the named-entity extraction to improve the extraction for a specific domain or named-entity (see 2.2.4).

Relationship Extraction: Entity records can be viewed as a multi-way relationship between named entities. Extracting binary relationships between two named entities of different type, is a common task in information extraction [19]. For example, a binary relationship extractor may extract a *phone-belongs-to* relationship between a named entity *phone* and a named entity *person name*. More recently, information extraction also deals with more complex multi-way relationships. For example, the authors of [15] propose a technique to identify maximal cliques in a graph where attributes are interconnected by pairwise relations, and generalize it to probabilistic cliques, where each binary relation may have a confidence associated with it. The drawbacks of combining binary relations using agglomerative algorithms or the technique used in [15] for record extraction are analyzed in [23]. The authors of [23] propose a modified approach that evaluates the *compatibility* of a set of attributes. Such a compatibility function is seen to achieve better accuracy in record extraction. Contrary to EUTC, all of the approaches mentioned for relationship extraction require considerable customization for each new domain.

Combining Structured and Unstructured Data: The authors of [14] present a method for integrating information from semi-structured sources with existing large database with multi-relational entities. The presented approach exploits a variety of recognition clues available in the database of entities to aid traditional information extraction with conditional random fields or semi-markov models. It also augments the database itself with the information about multiple noisy variants of entities to aid future extraction. The approach works well if the task is limited to extracting information from semi-structured text (for example, to integrate publication data from personal home pages with existing structured databases such as ACM digital library or DBLP).

EROCS (Entity RecOgnition in Context of Structured data) [2] efficiently links text documents with structured data. It employs entity templates and noun-phrase identification to identify the entities that best match a document. EROCS also embeds the identified entities in the document, thereby creating links between the structured data and segments within the document. With EROCS, the user explicitly defines entity templates that specify the entities to be matched in the document and for each entity, the context information that can be exploited to perform the match. EUTC improves upon EROCS by automatically extracting the entity data model from MDM and exploiting the probabilistic matching provided by the MDM system to perform the entity matching.

There is also existing work to integrate documents from content management systems with MDM. IBM InfoSphere Master Content for InfoSphere MDM Advanced Edition[9] links documents stored in IBM FileNet Content Manager with master records in IBM InfoSphere MDM Advanced Edition repository. The linking is done based on a document's metadata and does not exploit the document text. It works well in scenarios where each document contains information about a single entity. For example, it facilitates a single view of a citizen by providing access to documents such as birth certificate, driver's license, and passport.

The authors of [17] present an extension to IBM InfoSphere Master Content that additionally extracts information from the text of each document. The information extracted can be used to recognize errors in content metadata,

[9]http://www-01.ibm.com/software/data/infosphere/mdm_server/master-content.html

enables the concept of master content, and helps to identify duplicates. However, the extension still assumes that each document contains information about a single entity. Also, it employs domain-specific information extraction which requires considerable customization for each new domain.

7. CONCLUSION

In this paper, we have established that it is both possible and beneficial to integrate information from unstructured data sources into a MDM system. To the best of our knowledge, IBM InfoSphere MDM Extension for Unstructured Text Correlation (EUTC) is the first commercial solution that integrates information extracted from unstructured data into a MDM system. EUTC provides a generic architecture that works with any MDM system and any type of unstructured text. It exploits existing structured data to discover references to MDM entities in any free text. EUTC is domain-independent and works out-of-the-box; despite the fact that information extraction usually requires considerable customization to work for new domains or datasets. We have demonstrated experimentally that EUTC works well in a public-safety scenario. We are in the process of evaluating EUTC for other domains such as financial services. In terms of technical capabilities, we are extending EUTC to also learn previously unknown facts about an entity from text and automatically enrich existing master records with new attribute values.

8. ACKNOWLEDGMENTS

The authors would like to thank Dave Wilkinson, Alex Eastman, and Jim Cushman for their support and the constructive discussions. In addition, the authors are grateful for the help received from Puneet Sharma and Bruhathi Sundaramurthy in improving the paper.

9. REFERENCES

[1] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, 2009.

[2] V. T. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. Efficiently linking text documents with relevant structured information. In *VLDB*, pages 667–678, 2006.

[3] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan. SystemT: an algebraic approach to declarative information extraction. In *ACL*, pages 128–137, 2010.

[4] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *KDD*, pages 89–98, 2004.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[6] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *The VLDB Journal*, 20(4):495–520, 2011.

[7] I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[8] IBM Corporation. Master data management and accurate data matching. http://www-01.ibm.com/software/data/master-data-management/library.html, 2010.

[9] J. Jonas. Identity resolution: 23 years of practical experience and observations at scale. In *SIGMOD*, pages 718–718, 2006.

[10] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[11] G. Li, D. Deng, and J. Feng. Extending dictionary-based entity extraction to tolerate errors. In *CIKM*, pages 1341–1344, 2010.

[12] G. Li, D. Deng, and J. Feng. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In *SIGMOD*, pages 529–540, 2011.

[13] S. Lim. Cleansing noisy city names in spatial data mining. In *ICISA*, pages 1–8, 2010.

[14] I. R. Mansuri and S. Sarawagi. Integrating unstructured data into relational databases. In *ICDE*, pages 29–29, 2006.

[15] R. McDonald, F. Pereira, S. Kulick, S. Winters, Y. Jin, and P. White. Simple algorithms for complex relation extraction with applications to biomedical IE. In *ACL*, pages 491–498, 2005.

[16] M. Messerschmidt and J. Stüben. *Hidden Treasure*. PricewaterhouseCooper AG Wirtschaftsprüfungsgesellschaft, 2011.

[17] K. Murthy, D. P, P. M. Deshpande, S. Kakaraparthy, V. S. Sandeep, V. Shyamsundar, and S. Singh. Content-aware master data management. In *COMAD*, pages 206–209, 2010.

[18] N. Okazaki and J. Tsujii. Simple and efficient algorithm for approximate dictionary matching. In *COLING*, pages 851–859, 2010.

[19] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[20] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPS*, pages 1185–1192, 2004.

[21] R. L. Villars. *Scale-Out Storage in the Content-Driven Enterprise: Unleashing the Value of Information Assets*. IDC, 2011.

[22] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD*, pages 759–770, 2009.

[23] M. Wick, A. Culotta, and A. McCallum. Learning field compatibilities to extract database records from unstructured text. In *EMNLP*, pages 603–611, 2006.