ORACLE
Health Sciences

Identity Resolution and Data Quality Algorithms for Person Indexing

Resolving cross-referencing problems and establishing single views of patient and provider IDs – the science behind the Oracle Healthcare Master Person Index (OHMPI)

WHITE PAPER / OCTOBER 2, 2018

ORACLE®

EXECUTIVE OVERVIEW

Master Data Management (MDM), and more specifically, (Enterprise) Master Patient or Person Index (MPI or EMPI) represents the technology and framework that helps resolve cross-referencing problems and establish single views of person IDs in healthcare or in any complex enterprise data that needs to be „cleansed‟ from possible duplicates of the same entities. The underlying core technologies that MPI relies on are highly complex mathematics and algorithms from a wide range of disciplines including computer sciences, statistics, operational research, and probability.

This white paper highlights the technologies that process and resolve the inconsistencies within the data using data quality tools such as data profiling and cleansing, data normalization and standardization, phonetization, and finally data matching, also known as identity resolution. It will describe how these components work and how they are logically related to each other, and cover best practices around these processes which have been productized in the industry-leading Oracle Healthcare Master Person Index (OHMPI).

DISCLAIMER

The following is intended to outline our general product direction. It is intended for information purposes only and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

INTRODUCTION

As we head into the digital information age, more and more companies and institutions are required to deal with large and constantly increasing amounts of very heterogeneous and diverse type of „raw‟ data that needs to be intelligibly processed through distinct types of filters and sophisticated algorithms to reach a stage where the company can greatly benefit from its outcome as a „cleaner‟ and more meaningful data, without jeopardizing the integrity of original information. This is very much the case in the healthcare sector, where there is a genuine need for fast access to meaningful, accurate and structured patient data at various levels of healthcare services. Emergency care needs a tool that matches the patients incomplete and approximate information to legacy databases with the highest possible accuracy to avoid any possible medical error. A doctor in his office needs to access previous visits, medical treatments, and prescriptions, possibly in multiple systems or even in different hospitals and medical offices that his patient had visited prior to the present visit. The technology and framework supporting patient identity cross-referencing, sometimes also known as single patient view, is usually called Master Patient Index (MPI) – although we call it Master Person Index, since the same technologies are extended to the realm of Provider matching –, which is one representation of the more generic Master Data Management framework which involves a set of data quality tools, workflows and processes that maintains and presents a consistent and unified view of Master Data consisting originally of data fragments held in various applications and systems[i] [ii].

In this paper, we will look at data quality tools and their related algorithms that form the core engines of an MPI. The term „data quality‟ is used in this context to include the multitude of tools and algorithmic engines used to clean up, resolve conflicts and correlate the different entities within the information sources. Such tools embrace functionality known as data profiling and cleansing, geocoding, data standardization, data normalization and phonetization, and most importantly data matching (also known as identity resolution), which represents the ultimate step in correlating the different entities and resolving possible duplication issues.

Data quality components, which represent the building blocks of MPI, can be grouped under four major categories:

- Identification components, which analyze the data and establish its statistical signature (Data Profiling);
- Cleansing components, which filter some of the obvious errors and abnormalities (Data Cleansing);
- Standardization components, which inject some order, structure and normalization into the data;
- And Data Matching components, which identify and resolve replication of unique entities.

Some data quality specialists consider identity resolution (data matching) as a separate functionality from the other data quality elements mentioned above. However, we do not intend to discuss that or take a stance for or against that in this white paper. For the purposes at hand, it suffices to understand that there are four critical components to provide the single view of the master data.

The steps for cleansing and resolving conflicting data start by analyzing the incoming information using statistical analysis tools, to evaluate the degree of cleanliness and to uncover the peculiarities of the information (this is called the profiling step). After this, the user needs to act on the obvious inconsistencies and issues by modifying the data (this is the cleansing step which is related to profiling). Then comes the important phase where we uncover underlying details of the data by identifying the types and the order of the different „microscopic‟ elements (this is the standardization step, which includes the more specific normalization process). This step performs a sophisticated parsing and „typing‟ to prepare the field for the matching step. When the data is well-defined and “typed”, corresponding fields‟ values are compared together to compute an overall weight that will measure the degree of closeness of comparable entities, which is the final matching (or identity resolution) step.

DATA LOADING, AGGREGATION AND FORMATTING

This step is not traditionally part of the data quality procedure per se but is a very useful and necessary step when it comes to loading various complex data from different physical systems, and with diverse source formats and categories. Data Integration or so-called ETL (Extraction, Transformation, and Loading) tools can help perform these complex aggregations and formatting functions by hiding most of the complexity related to connectivity details to heterogeneous and diversified data sources. They ensure, for example, that when extracting two different source tables with different formats, these are merged into a target table with one unified format. Visually-rich modeling environments to perform required mappings and transformations between the different data makes such tools even more appealing. Care must be taken though, when dealing with transformation using ETL in the context of an MDM / MPI project. Users should be very cautious about not overlapping transformation tasks in ETL with similar ones in the cleansing step. By default, ETL does not change the content unless it is an obvious filtering requirement.

DATA PROFILING / CLEANSING

After extracting and loading data from multiple data sources to a consolidated staging tables or files, users can start inspecting and understanding the raw information contained in those table(s) using a data profiling engine. The incoming data is in batch-mode (as opposed to real-time flow) to complete the profiling process. Such components are expected to delineate the statistical signature of the examined data and detect various types of anomalies. Among the most key features a user should look for in the profiling phase are:

- Frequency counts of the different values within each strategic field in the data. For example, in a first name field column, we could have five thousand “John” out of a list of a hundred thousand first name values, which represents five percent of the total count. Such information could be further processed to account for locally-based statistics. We might find it suspicious to have a relatively high frequency count for “John” in a city where the existing local statistics points to an average number close to 0.5 percent.
- The frequency counts of empty values or „illegal‟ set of characters within any probed fields.
- Formatting issues within different fields (for example, a date of birth with a wrong format or with out-of-range dates).
- Generic values (for example, “baby of” value is very frequent for new babies' first names).
- Degree of cleanliness of the entire record within the data (assuming we have multiple property fields). For example, a record of ten fields having two „empty/illegal‟ values is 'cleaner' than one with six „empty/illegal‟ values.

The notion of a frequency count for a specific value within a field can be further extended to a more general concept of patterns-based frequency where instead of searching for, let say, „999999999‟ values, a user can rely on regular expressions like all values that start with three nines „999*‟. All the features highlighted above can be formalized by using some flexible rules formulated through configurable files (rule-based profiling engine).

Finally, the profiling engine outputs detailed reports about the statistical properties, and ideally an easy-to-read aggregated report about the major singularities found within the data. The profiling engine defines a set of rules that help separate the records into two distinct groups. The „good‟ file holds the records flagged as being above a certain cleanliness threshold and the „bad‟ file which encompasses all the records that were rejected by the set of rules and need to undergo cleansing processing, which represents the second logical phase after profiling the data.

The cleansing step, associated with enforcing the rules formulated in the first profiling phase, corrects as much inconsistencies as possible from the data records, before updating the „good‟ and „bad‟ files with the corrections. The aim here is to minimize the issues related to format, illegal characters, empty fields, etc. This two-phase process can be iterated as many times as needed until we reach an acceptable level of clean data where the „bad‟ file size becomes relatively minor compared to the „good‟ one.

It is noteworthy to pinpoint that the effectiveness of the profiling phase will noticeably increase in the iterative process if the raw data is normalized / standardized (in the cleansing phase) before going throughout the next profiling procedure. Here we are referring to the normalization / standardization processes that come later in the data quality sequence. This will correct the frequency counts of the different values. Names like “Beth”, “Bessie”, “Betsy”, “Bette” and “Bettie”, in the US locale for example, will normalize to “Elizabeth” increasing the frequency count.

STANDARDIZATION

Standardization can be defined as the process of creating structure in unstructured or semi-structured data, while normalization, which is a special case of the more general standardization process, is an enhancement of an already structured data. Both functionalities help optimize the matching results, and can be enlisted as pre-match procedures. The key operations here are parsing the incoming record into basic fields, identifying the types of each atomic element, normalizing their values and finally defining the best order in which the elements should be reorganized. This comes down to finding the right patterns from the locale-specific associated dictionary file for each type.

For example, the following free-form address: “716 N RICHARD ARRINGTON JUNIOR BOULEVARD BIRMINGHAM”, within a „US“ locale, can be standardized into:

- Street number: 716
- Directional prefix: North
- Street name: RICHARD ARRINGTON JR
- Street type: Blvd
- City: BIRMINGHAM

The names on the left represent generic and basic address types that would apply for different locales. For example, in the specific case of address-type standardization, the different steps consist of:

- Parsing. Breaking down the string into different components and defining fundamental types like numeric, alpha-numeric, special characters.
- Identifying address-types. Looking up the different type and locale-specific data dictionaries to identify street types, street directions, business buildings, etc.
- Normalizing the fields. Replacing the different fields' values with their standard forms.
- Finding the right Pattern: In general, there is more than one pattern for the same set of inputs of data types. For example, in the street address example above, we have the following input-output configuration in the pattern dictionary table:
  – Input: NU AU AU A2 TY DR AU
  – Output: HN NA NA NA ST SD EI T* 85

Here, the two-character tokens define diverse input and output types („NU“ stands for numeric and „AU“ for alpha string as inputs, while „HN“ accounts for house number and „NA“ for street name as outputs), and the ordered set of tokens define the input representation of the address and the possible output solution. A locale weight (in our example: 85) that defines the relative importance of the pattern in case it is included in a larger pattern. The higher weight will overcome the lower ones. This process is non-linear in nature and will select the best possible pattern for a given street address. It needs some expert knowledge to set the list of patterns.

NORMALIZATION

Normalization is an enhancement process of an already structured and typed data object, meaning that the „structure‟ already exists and the fields‟ types are known parameters, but they need to be set to some pre-configured standard values. Let say, for example, we have a person name, in a US locale, like: (First name, Last name, Generational suffix, Title) = {Rick, Phinque, Junior, Pres.}, then, the normalization of this person attributes will consist of transforming the previous values to {RICHARD, FINK, JR, PRESIDENT}, assuming that we use configurable locale-specific dictionary files that classify "Richard" as the standard first name for "Rick" and "Fink" as the standard last name for "Phinque", and so on and so forth. We will mention later how such functionality is at the heart of the OHMPI's framework[ii] [iii].

PHONETIZATION

The technique of phonetization is meant to capture words that have different spelling but have the same pronunciation in a given language and assemble them together. The most important application of phonetic encoders is fuzzy data retrieval. It can be regarded as the first attempt to retrieve data in a way that is more flexible than traditional techniques. Such a technique is a good candidate for identifying blocks of relevant data as we will see later in the matching process. The most commonly used phonetic algorithms are Soundex and NYSIIS.

Soundex is a simple yet efficient encoder that outputs a four-character length alphanumeric. It is composed of a short list of static rules that work best for English names, but there are some other language-specific equivalents to the English version (for example, the French Soundex in OHMPI[iv]).

NYSIIS, which stands for New York State Identification and Intelligence System is a more advanced encoder composed of a longer list of static rules. It works best for English names. For example, names like "Martha", "Marta", "Mirta", and "Mrta" return a „M630‟ code with Soundex and a „MRT‟ code with NYSIIS, in their original versions. Other phonetic encoders were developed like the RefinedSoundex, a more sophisticated version of the Soundex algorithm meant to be used as a spell-checking device. It has more discriminatory power than Soundex. Also, in the same group of phonetic encoders, we have Metaphone and DoubleMetaphone available in OHMPI too. Table 1 gives the differences between these algorithms.

Table 1: Comparison of Phonetic Encoders

NAME	SOUNDEX	SOUNDEXFR	REFINEDSOUNDEX	NYSIIS	METAPHONE
Martha Mrta	M630 M630	MRT MRT	M80960 M8960	MART MRT	MRO MRT
David Dave	D130 D100	DV DV	D60206 D6020	DAVAD DAV	TFT TF
Suhanto Santo	S530 S530	SNT SNT	S30860 S30860	SANT SANT	SHNT SNT

DATA-TYPE VALIDATION: THE POSTAL ADDRESS EXAMPLE

A complementary and sometimes surrogate technique to standardization is data-type validation. We can illustrate it best with a postal address type, where the validation algorithm compares the incoming address with a set of accurate, and regularly updated, legacy addresses from a postal service like USPS (United States Postal Service).

Such technique needs to narrow the selection by city/county to make the web-based services reasonably fast and functional, and to retrieve a smaller list of addresses, preferably only one. In general, the following logic is carried out to validate the address.

Check for reverse directional type (meaning from “main st n” to “n main st”), missing directional type (from “main st” to “n main st”), incorrect directional type (“s main st” to “main st”), incorrect street type (“main ave” to “main st”), and incorrect spelling (“from maine st” to “main st”).

The advantage of standardization over validation is that the former structures the data into typed and independent atomic-level elements that can be used independently and effortlessly in matching. On the other hand, data validation has the benefit of correcting the data with official, up-to-date, information. Both techniques can work in tandem, though, which gives the best value.

MATCHING AND DEDUPLICATION

Data matching, also called deduplication or record linkage, addresses the problem of identifying and resolving issues with those records that belong to distinct data sources, or to the same source, which are multiple representations of the same entity but for complex reasons, are difficult to correlate and link together. A match engine measures a degree of similarity between any two comparable records, and outputs a matching weight that is computed by comparing all the underlying characteristics of each record. In the case of a person object for example, those characteristics might be first name, last name, date of birth, social security number, and so on.

One of the most important components of the matching calculation is the comparison functions[v] [vi] [vii] which evaluate the closeness of the related elements of the records. When the compared records hold only one field, matching can look easy, since it comes down to comparing two field’s values without accounting for anything else. Let’s say we have first names: “Anderson” vs. “Andresun”. Finding the right comparison function will resolve the problem. But, in real-life things are more complicated, and we might have multiple fields in each record, those fields might be correlated, and we need to understand the statistical properties of the data. In these terms, matching is a multidisciplinary field involving computer science (which provides the comparison algorithms), operational research (through the optimization algorithms that help choose the best solution[viii] [ix]), statistics (which analyzes the large set of data using statistical techniques) and usually probabilities (which are at the heart of the most recognized method).

MATCHING METHODOLOGIES

One of the most accepted methodology for matching was developed by Fellegi & Sunter[x] [xi] who established a formal mathematical framework for record matching that is known today as the standard model because of its overwhelming adoption. It calculates two types of conditional probabilities for each of the fields involved in matching, relying on an optimization approach of the different parameters, and then measuring a locale match weight as a function of the logarithm of the ratio of those two probabilities.

Finally, it calculates a composite weight by summing up all the individual fields' weights, using the approximation that the different records' fields are mutually statistically independent. In recent times, we saw the introduction of new promising approaches that rely on artificial intelligence methodologies like machine learning techniques that might resolve some of the issues with the old methods, but the foundation of the Fellegi & Sunter methodology still holds strong ground and can be used with the newer methodologies.

One important step in the matching process consists of estimating the match and potential duplicate thresholds. In simple terms, the distribution of weights generated by the cross-comparison of two data files can be looked at as two separate groups of N-dimensional weights that we can designate as the true matches and the true non-matches. But the solution is more complex since the lack of certainty knowledge of the true matches and non-matches generates a third group of hard-to-resolve weights that fit into a fuzzy area between the two groups, and that we call potential duplicates. These third-group weights need manual intervention to be resolved or maybe an additional re-run with different configuration parameters. The goal of the methodology is to minimize this fuzzy area by relying on an optimal decision rule, using optimization techniques, to determine the best thresholds.

In short, the standard model consists of cross-comparing two independent files modeled as sets of element records **A**(a) and **B**(b) (be aware that we assume the files to be clean. If they hold duplicates, we first need to cleanse the files, then start this merging procedure). Any pair of records (a, b) belong to the product space **A x B** of all pairs, and must be classified exclusively as a true match **M** or a true non-match **U.** The size of **M** is at most equal to N, the number of records per file, while **U** is of order N^2, with:

$$M = \{(a, b): a=b,\ a \in A,\ b \in B\}$$

$$U = \{(a, b): a \neq b,\ a \in A,\ b \in B\}$$

We define record properties associated with elements **a** and **b** as **α(a)** and **β(b)** respectively, and we define a comparison vector **γ = (α(a), β(b))** from the comparison space **Γ.**

Each comparison vector **γ (α(a), β(b)) = {γ^1 (α(a), β(b)), …, γ^K (α(a), β(b))}** is of dimension K, K being the number of matching fields per record. Our goal is to decide for every **γ** if it belongs to **M** (true match), to **U** (true non-match), or is an undecided case. To this purpose, we calculate, for every single field, the conditional probabilities of true matches $m_k(\gamma^k)$ and true non-matches $u_k(\gamma^k)$, where k is the field's index. The composite weight is formulated as:

$$m(\gamma) = m_1(\gamma^1).\, m_2(\gamma^2) \ldots m_k(\gamma^k)$$

$$u(\gamma) = u_1(\gamma^1).\, u_2(\gamma^2) \ldots u_k(\gamma^k),$$

assuming that the different fields are mutually statistically independent. We can reformulate these equations by introducing the ratio $m\,(\gamma)/u\,(\gamma)$ and use their logarithm (order n) since it is a monotonically increasing function, which leads to:

$$w(\gamma) = w^1 + w^2 + \ldots + w^k \text{ where } w^j = \log(m(\gamma^j)) - \log(u(\gamma^j))$$

We finally obtain the composite weight $W_\gamma = \sum_{j-1}^{K} w_\gamma^j$ for each pair of records. To this mean, we define a random decision function D = {d(**γ**)} where:

$$d(\boldsymbol{\gamma}) = \{P(A_1 \mid \gamma), P(A_2 \mid \gamma), P(A_3 \mid \gamma)\};\ \gamma \in \Gamma \text{ and}$$

$$\sum_{j=0}^{j=3} P(A_i \mid \gamma) = 1,$$

with A_1, A_2 and A_3 respectively the sets of true match, potential duplicates and true non-match, which will help decide for every given **γ** if it belongs to **M** (true match), **U** (true non-match) or is an undecided case.

We define also a decision rule L: **Γ(γ) → D,** which is a mapping from the comparison space to the decision function, as the optimization parameter, along with the types of errors associated with linkage rules. The first one occurs when a true non-match is set as a match. It has the probability:

$$P\ (A_1 \mid U) = \sum_{\gamma \in \Gamma} u(\gamma) P(A_1 \mid \gamma)$$

The second one occurs when a true match is set as a non-match. It has probability:

$$P\ (A_3 \mid M) = \sum_{\gamma \in \Gamma} m(\gamma) P(A_3 \mid \gamma)$$

Let's define a linkage rule as the one on the space Γ, at levels μ and λ (0< μ<1, 0< λ<1) denoted by L (μ, λ, Γ), where μ = P (A_1 | U) and λ = P (A_3 | M). Then, among all the possible linkage rule functions L' (μ, λ, Γ), the optimal one L is defined by:

$$P\ (A_2 \mid L) \leq P\ (A_2 \mid L')$$

That means that the optimal linkage rule is the one that maximize the probabilities of positive disposition (A1 ,A3) and minimize the potential duplicate region while respecting the errors constraints levels μ and λ. For a given admissible (μ, λ) pair of errors, we can define the integers n and n` such that:

$$\sum_{i=1}^{n-1} \quad u_i < \mu \le \sum_{i=1}^{n} \quad u_i \quad \text{and} \quad \sum_{i=\tilde{n}}^{N_\Gamma} \quad m_i < \lambda \le \sum_{i=\tilde{n}+1}^{N_\Gamma} \quad m_i$$

This will lead us to the optimal solution L0 (μ, λ, Γ) represented through:

$$d(\gamma_i) = \begin{Bmatrix} (1,0,0) & i \le n-1 \\ (P_\mu, 1-P_\mu, 0) & i = n \\ (0,1,0) & n < i < \tilde{n}-1 \\ (0,0,1) & i \ge \tilde{n}+1 \end{Bmatrix},$$

where $P_\mu$and $P_\lambda$are the solutions of the equations:

$$u_n . P_\mu = \mu - \sum_{i=1}^{n-1} \quad u_i \ \text{and} m_{\tilde{n}} . P_\lambda = \lambda - \sum_{i=\tilde{n}+1}^{N_\Gamma} \quad m_i \ .$$

If we define two positive numbers $T_\mu = \frac{m(\gamma_n)}{u(\gamma_n)}$and $T_\lambda = \frac{m(\gamma_{\tilde{n}})}{u(\gamma_{\tilde{n}})}$, then, the optimal solution becomes:

$$d(\gamma_i) = \begin{Bmatrix} (1,0,0) & T_\lambda \le m\,(\gamma)/u\,(\gamma) \\ (0,1,0) & T_\lambda \le m\,(\gamma)/u\,(\gamma) < T_\mu \\ (0,0,1) & m\,(\gamma)/u\,(\gamma) < T_\lambda \end{Bmatrix}$$

$T_\mu > T_\lambda$are respectively the assumed match threshold and the potential duplicate threshold. So, from this point we need to calculate the m(γ) and the u(γ) and eventually the T_λ and the T_μ to fully resolve the matching problem.

Once we collect all the weights associated with the matching process, we need to make a decision on the associated pair of records. To this end, we need to estimate the thresholds previously defined, as shown in figure 1.

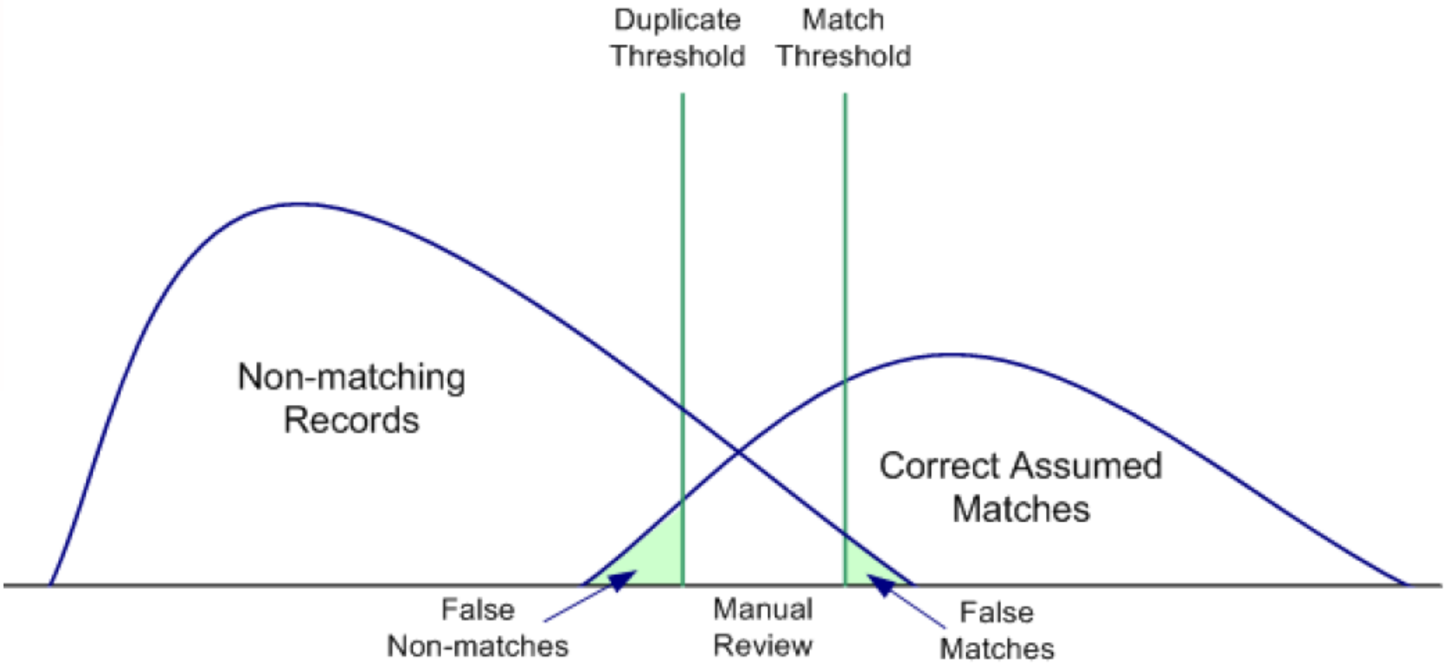

Figure 1

COMPARISON FUNCTIONS

One of the most critical step in the matching process is to choose the right comparison function to associate with a given match field. For example, if the field is a numeric, then the comparator should handle all the peculiarities of numbers.

Approximate String Comparators

There exists a large library of string comparators in computer science with algorithms ranging from simple to very complex. The algorithms strive to account for the many human-related possible errors when typing, writing or exchanging the information. It ranges from accounting for different levels of transpositions between characters or set of characters[v] [vi] [vii] [xii] [xiii], to insertions and deletions, etc. For example, the Bigram algorithm accounts for two-character length transpositions[v]. They are widely used in information retrieval, the Jaro algorithm accounts for more sophisticated transpositions within a specified length and it also includes insertions and deletions, while the Winkler-Jaro algorithm takes it a step higher and improves the Jaro algorithm by adding three additional enhancements (scanning/keypunch errors[v]; non-linear weighting of the first characters relative to the last ones[v]; special handling of strings longer than six-characters justified by statistical data findings[vi]).

An extensive study of approximate string comparators in computer science found that the Jaro and Winkler-Jaro algorithms are the most powerful and efficient among twenty comparators[v]. In a large study, Budzinsky[xiii] concluded that the comparators due to Jaro and Winkler were the best among twenty comparators in the computer science literature. The basic Jaro algorithm does:

- Compute the string lengths.
- Find the number of common characters in the two strings.
- Find the number of transpositions.

The definition of common is that the agreeing character must be within half the length of the shorter string. The definition of transposition is that the character from one string is out of order with the corresponding common character from the other string. The string comparator value (rescaled for consistency with the practice in computer science) is:

$$Jaro(S_{1,}S_2) = 1/3\,\{\frac{N_{common}}{L(S_1)} + \frac{N_{common}}{L(S_2)} + 1/2\,\frac{N_{transpotition}}{N_{common}}\}$$

Approximate Data-Type Comparators

We can extend the concept of approximate string comparison to embrace larger sets of data type comparators. It could be a date comparator that handles different type of date format and calendars, including handling dates by their distances in time or it could be some airplane-specific parts comparator that contains the needed algorithm for that specific functionality. Following this concept, we can build large sets of business-specific and vertical-specific comparators that will be used as needed.

The comparators presented above represent the most important components of the match engine algorithm since they control the outcome weight to a very high degree[iii].

ORACLE HEALTHCARE MASTER PERSON INDEX

Implementing MPI (and in general MDM) solutions begins with defining an appropriate object model that fits the data-set at hand and illustrates the intended solutions. It proceeds with extracting the relevant data from one or multiple source applications, possibly on a distributed environment, and mapping them into the master data repository. Such extraction can be performed through a web-based interface or using an ETL-type extractor, when dealing with large data-sets. During the loading process into the MPI repository, some or all the functionality introduced in this article (profiling, cleansing, standardization, normalization, phonetization and matching) are executed in the appropriate order.

After the incoming records are classified as new records (i.e. there are no matches) or as already present in the repository (i.e. true matches, assuming that we have already resolved all possible potential duplicate conflicts), we can offer data consumers and the data sources, with a single view of patient data. Consumers, which may represent a network of doctors, can access a single patient's view through customized interfaces, while the data source can use and manage the deduplicated information for being synchronous with the master data repository.

Oracle Healthcare Master Person Index (figure 2) relies on the data quality and identity resolution capabilities described above, including a very flexible data object model that lets the users define and fit it to their needs. It leverages NetBeans platform to design master person indexes. OHMPI exposes many of the MPI operations as APIs and Web Services to provide data services for multiple healthcare consumer applications including SOA based applications. Thus, OHMPI offers a standards-based, services-enabled infrastructure to create and publish single person views.

The match engine (identity resolution component) and the standardization engine, described in this paper, are seamlessly integrated within the product. Master index Configuration Editor (figure 2 – design time) offers all the flexibility to visualize, choose and configure the different parameters from each of the engines. Figure 2, run-time section, shows an integrated set of components that work in harmony to ensure availability of unified, trusted single-view to all systems in the enterprise. A visually rich, browser-based application (Master Index Data Manager) is also available for data stewardship activities such as reviewing automatic merges, view of potential duplicates and executing manual merges, running activity reports, and conducting audit based on extensive transaction logs.

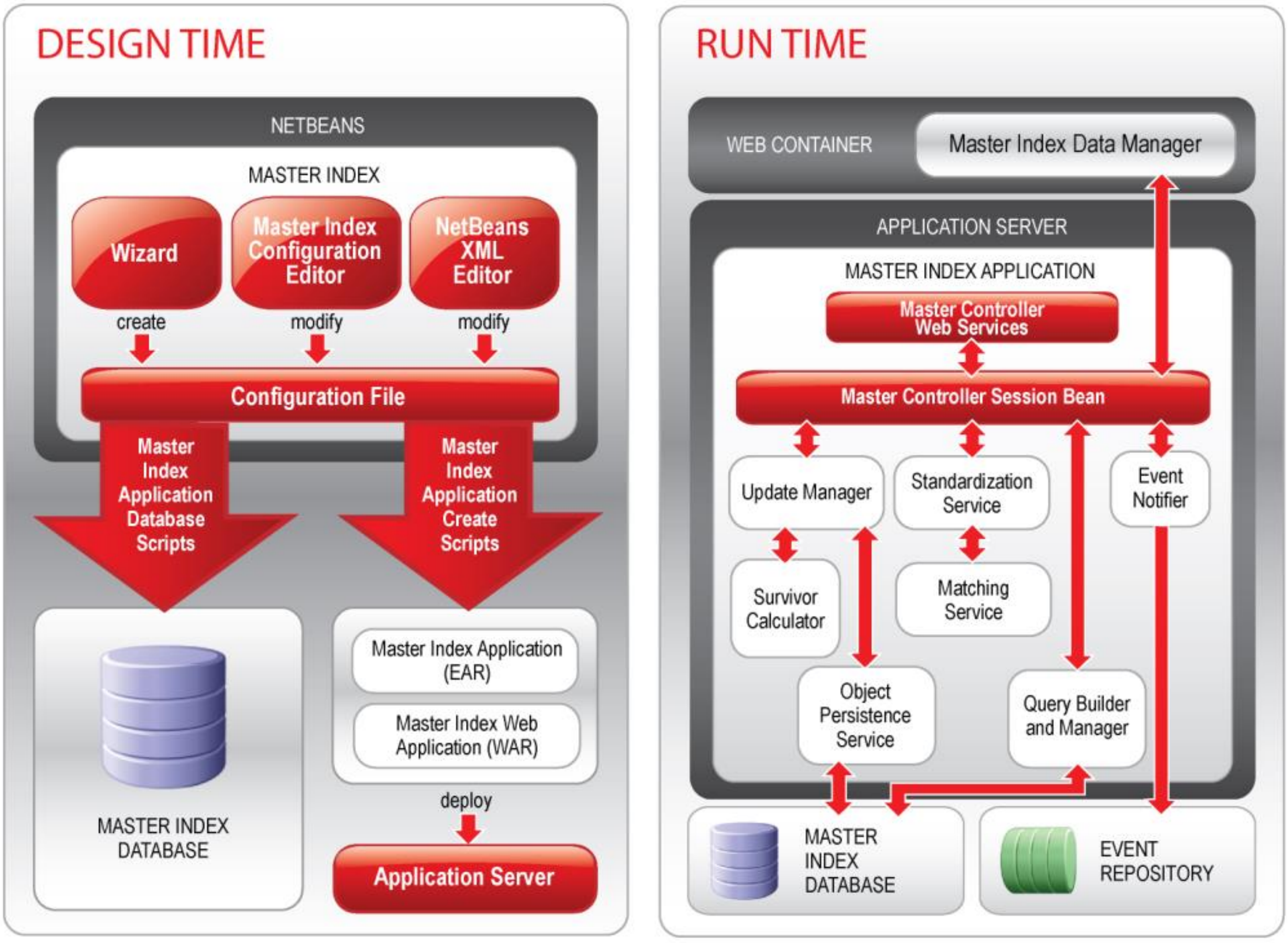

Figure 2 - OHMPI Design and Run-Time Architectures

Thanks to these abilities, OHMPI in its current or earlier releases has been adopted by many healthcare customers from multiple segments ranging from providers and payers to regional and national health exchanges.

- A very large national health system (200+ million patients) adopted OHMPI as the core engine for its Patient Cadaster to record, cross-index, and deduplicate patient information electronically. The resulting project and interaction with the implementation team, on a daily basis, helped improve and test to the extreme limit all of the product’s engines, and demonstrated the robustness, reliability, and scalability of the solution.
- OHMPI was implemented by a statewide public/private collaborative of universities and health systems who shared the vision of using health sciences research to improve the health and economic well-being of the members in their systems. They established a data framework to support interoperability and research that was based on Enterprise Master Patient Index (EMPI), an implementation of OHMPI. The solution helped providers collaborate on care and quality improvement initiatives through Health Information Exchange (HIE) and attract new bio-pharmaceutical investments focused on improving patient care. Benefits included increased patient and employee satisfaction, ability to link patient records across systems into a single record and establishing integrated projects for translational research.

References

[i] Master Data Management: Integrated information is not complete information
Sofiane Ouaguenouni & David K. Codelli. SOA World Magazine, November 2008 / Volume: 9 Issue 11
[ii] Improving Data Management with Sun's MDM Suite
David K. Codelli & Sofiane Ouaguenouni, Gartner MDM Summit presentation, Chicago 17-19 November 2008
[iii] Master Index Match Engine. Part1: Match Comparator Plug-In Framework
Sofiane Ouaguenouni. MDM Learning Series, June 2008
[iv] OHMPI Master Index Configuration Guide: Master Index Encoders Elements and Types
[v] Approximate String Comparison and its Effect on an Advanced Record Linkage System
Edward H. Porter and William E. Winkler, U.S. Bureau of the Census, 1997
[vi] Improved String Comparator, Technical Report, Statistical Research Division
Lynch, M. P. and Winkler, W. E. – Washington, DC: U.S. Bureau of the Census, 1994
[vii] String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage
Winkler, W. E. Proceedings of the Section on Survey Research Methods, American Statistical Association, 354-359, 1990
[viii] Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage
William E. Winkler. Proceedings, American Statistical Association, 1988
[ix] Maximum Likelihood of factor analysis using the ECME Algorithm with complete and incomplete data
Chuanhai Liu and Donald B. Rubin. Bell Labs and Harvard University Statistica Sinica 8 (1998), 729-747
[x] A Theory for Record Linkage - Ivan P. Fellegi, Alan B. Sunter. Journal of the American Statistics Association, Volume 64, Issue 328 (Dec. 1969) 1183-1210
[xi] Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida
Matthew A. Jaro. J. of the American Statistics Association, Volume 84, Issue 406 (Jun. 1989) 414-420
[xii] Automatic Spelling Correction in Scientific and Scholarly Text
Pollock, J. and Zamora, A. Communications of the ACM, 27, 358-368, 1984
[xiii] Automated Spelling Correction, Statistics Canada. Budzinsky, C. D. 1991

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway,
Redwood Shores, CA 94065
USA

Worldwide Inquiries

TELE	+ 1.650.506.7000
FAX	+ 1.650.506.7200

oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com/healthcare. Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com/oracle   facebook.com/oraclehealthsciences   twitter.com/oraclehealthsci

Integrated Cloud Applications & Platform Services

Identity Resolution and Data Quality Algorithms for Person Indexing
1st version: August 2010; Current version: Otober 2018
Main Author: Sofiane Ouaguenouni, Ph.D.
Contributing Authors: Miguel Coelho, Kumar Sivaraman, Terry Braun

ORACLE