TESINA DE LICENCIATURA

TITULO: Integrando GIS con OLAP y Bases de Datos de Trayectorias

AUTORES: Gardella Juan Pablo

DIRECTOR: Vaisman Alejandro

CODIRECTOR: Gordillo Silvia

CARRERA: Licenciatura en Sistemas

UNLP

Resumen

Los GIS disponibles en la actualidad no permiten integrar eficientemente información geográfica y de objetos móviles. Si agregamos a esto el hecho de que las organizaciones suelen disponer de grandes volúmenes de datos almacenados en repositorios denominados Datawarehouse para ayudar a la toma de decisiones mediante el uso de herramientas para el procesamiento analítico en línea, concluimos que es necesario proveer herramientas que faciliten la integración antes mencionada.
En este contexto, un problema frecuente consiste en asociar elementos espaciales en un GIS con elementos no espaciales en un Datawarehouse. En la primera parte de esta tesis pesentamos una herramienta que permita realizar esta asociacion en forma semiautomatica. En la segunda parte de la tesis, extendemos RE-SPaM, un lenguaje basado en expresiones regulares para expresar patrones de trayectorias, permitiendo incorporar a aquel condiciones geométricas sobre elementos espaciales en un GIS.

Líneas de Investigación

Datawarehouse, Sistemas de Información Geográfica, Base de datos de Trayectorias, Trayectorias semánticas, Record Linkage, OLAP, Matching, Data mining.

Conclusiones

Del análisis de la solución implementada, se desprende que cumple con el objetivo inicial de simplicar y automatizar en lo posible la integración de GIS y OLAP, al completar la estructura auxiliar en la plataforma Piet reduciendo considerablemnte el tiempo y el costo de realizar esta labor manualmente.
La extensión al lenguaje RE-SPaM propuesta, RE-SPaM++, cumple con el objetivo inicial de aprovechar la estructura auxiliar y de esta manera permitir dentro del lenguaje el uso mediante PietQL de la información GIS/OLAP en la definición de reestricciones a los patrones de trayectorias descubiertos por el algoritmo de RE-SPaM

Trabajos Realizados

Herramienta que facilita la integración de información geográfica con información no espacial.
Extensión al lenguaje RE-SPaM (RE-SPaM++) que permite definir reestricciones geográficas.

Trabajos Futuros

Implementación de una interfaz gráfica para la herramienta de integración independiente de JUMP.
Implementar el soporte a consultas parametrizadas en RE-SPaM++.

Fecha de la presentación: Diciembre 2009

Tesis de Licenciatura en Sistemas

Integrando GIS con OLAP y Bases de Datos de Trayectorias.

por

Gardella Juan Pablo

Universidad Nacional de La Plata
Facultad de informática

Director: Alejandro Vaisman
Codirector: Silvia Gordillo

Buenos Aires, 2009

Para mis padres y Eliana

Agradecimientos

Al finalizar un trabajo tan arduo y lleno de dificultades como el desarrollo de una tesis, es inevitable que te asalte un muy humano egocentrismo que te lleva a concentrar la mayor parte del mérito en el aporte que has hecho. Sin embargo, el análisis objetivo te muestra inmediatamente que la magnitud de ese aporte hubiese sido imposible sin la participación de personas e instituciones que han facilitado las cosas para que este trabajo llegue a un feliz término. Por ello, es para mí un verdadero placer utilizar este espacio para ser justo y consecuente con ellas, expresándoles mis agradecimientos.

Al Doctor Alejandro Vaisman y a la Doctora Silvia Gordillo por asesorarme a lo largo de la tesis y acompañarme en este camino que hoy culmina en el presente proyecto, por compartir su conocimiento conmigo e inspirar en mi mucha admiración.

A la Doctora Leticia Gómez por ayudarme a lo largo de la tesis desinteresadamente y brindarme su amistad.

A Eliana por estar siempre a mi lado, apoyándome y dándome las fuerzas necesarias para seguir adelante. Gracias por estar conmigo durante esta etapa de mi vida.

A mis padres, quienes durante todos estos años confiaron en mí, comprendiendo mis ideales y el tiempo que no estuve con ellos. Gracias por confiar en mí y por ser como son.

Resumen

Un Sistema de Información Geográfica (GIS) permite la creación, almacenamiento y manejo de información espacial. Por otro lado, la reducción del costo de los dispositivos electrónicos que permiten reportar información sobre su ubicación espacio-temporal, están abriendo una puerta a nuevas aplicaciones, por ejemplo herramientas que permiten consultar las trayectorias de los diferentes objetos móviles.

Los GIS disponibles en la actualidad no permiten integrar eficientemente información geográfica y de objetos móviles. Si agregamos a esto el hecho de que las organizaciones suelen disponer de grandes volúmenes de datos almacenados en repositorios denominados Datawarehouse para ayudar a la toma de decisiones mediante el uso de herramientas para el procesamiento analítico en línea, concluimos que es necesario proveer herramientas que faciliten la integración antes mencionada.

En este contexto, un problema frecuente consiste en asociar elementos espaciales en un GIS con elementos no espaciales en un Datawarehouse. En la primera parte de esta tesis pesentamos una herramienta que permita realizar esta asociación en forma semiautomática. En la segunda parte de la tesis, extendemos RE-SPaM, un lenguaje basado en expresiones regulares para expresar patrones de trayectorias, permitiendo incorporar a aquel condiciones geométricas sobre elementos espaciales en un GIS.

Índice general

Capítulo 1

Introducción

Un Sistema de Información Geográfica (GIS[1]) permite la creación, almacenamiento, análisis y manejo de información espacial, representada por geometrías almacenadas en capas (layers), junto con atributos relacionados. Un sistema de este tipo permite integrar, almacenar, editar, analizar y mostrar información geográfica.

El procesamiento análitico en línea (OLAP[2]) es un conjunto de herramientas y algoritmos brindan respuestas rápidas a consultas de tipo analítico. Una base de datos configurada para OLAP utiliza un modelo multidimensional que facilita la ejecución de estas consultas analíticas. La información OLAP está organizada, generalmente, en forma de dimensiones (en tablas de dimensión) y hechos (tablas de hechos). Las aplicaciones típicas de OLAP están relacionadas con informes de ventas, marketing, management, reportes financieros, etc., e involucran casi siempre operaciones de sumarización. Consultas típicas de un OLAP son de la forma "total de ventas semanales por región".

Las bases de datos de trayectorias ó de objetos móviles (Moving Objects Databases (MODs)) tienen como objetivo representar movimientos de objetos (trayectorias) en bases de datos que permitan realizar análisis, consultas en tiempo real, como así también minería de datos[3]. Las trayectorias se representan por una secuencia de $\langle o, t, p \rangle$ siendo o el objeto de interés, t el instante y p la posición. La posición es un punto geométrico (coordenadas x, y).

En este trabajo denominamos *Lugar de Interés de una Aplicación* (PoI[4]) a una geometría que contiene información semántica asociada en forma de atributos. Cuando un objeto móvil permanece el tiempo suficiente dentro de un PoI este es considerado una parada (stop) y sus coordenadas (x, y) se reemplazan por un objeto que contiene los datos del stop. Esto permite considerar cada objeto de trayectoria como una *secuencia de paradas* en lugar de *secuencia de puntos*. Esta visión de tayectorias, se denomina *trayectoria semántica* [SPD+07].

Algunas propuestas soportan la integración entre GIS y OLAP, comunmente

[1] Geographic Information System

[2] On Line Analytical Processing

[3] La minería de datos (DM, Data Mining) consiste en la extracción no trivial de información que reside de manera implícita en los datos

[4] Place of Interest of the Application

denominado SOLAP[5]. Una de ellas denominada Piet[pie], es una plataforma basada en un modelo formal para soportar esta integración. Asimismo, Piet provee un lenguaje de consulta denominado PietQL.

Regular Expression for Sequential Pattern Mining (RE-SPaM) [GV09] es un algoritmo de descubrimiento de patrones secuenciales que utiliza un lenguaje para restringir la secuencias obtenidas. Aplicado a trayectorias semánticas, los patrones secuenciales a descubrir son las secuencias de paradas (stops) por los cuales pasó dicho objeto. El lenguaje se basa en expresiones regulares sobre restricciones aplicadas a las paradas. Las expresiones pueden incluir atributos, funciones y variables.

1.1. Motivación y contribuciones

Piet implementa la integración GIS/OLAP mediante estructuras auxiliares (tablas) que relacionan información del datawarehouse con información del GIS. En la implementación actual de Piet, estas tablas son creadas y completadas manualmente. Este proceso implica asociar elementos en el datawarehouse con elementos geométricos en el GIS. Por lo tanto a medida que aumenta el volumen de información a relacionar, la tarea se torna costosa y sujeta a errores.

En esta tesis se propone una herramienta para automatizar el proceso de asociación entre elementos espaciales y no espaciales reduciendo, de esta forma, el costo de la creación de la estructura auxiliar.

Por otra parte, RE-SPaM opera sobre una base de trayectorias que contiene información geométrica que no puede ser aprovechada en la implementación actual, ya que el lenguaje no incluye funciones que permitan manipular datos espaciales (geométricos).

Se presentará una extensión del lenguaje RE-SPaM que permite, no sólo el soporte de funciones geométricas, sino también, la integración de objetos móviles en la plataforma de Piet aprovechando su modelo de datos mediante el lenguaje de consulta PietQL.

1.2. Organización del documento

La tesis está dividida en dos partes. La primera abarca la herramienta de integración de datos espaciales y no espaciales (GIS y OLAP), mientras que la segunda, la integración del lenguaje PietQL en el lenguaje de RE-SPaM.

La primera parte comienza en el Capítulo 2, donde se detalla el problema de la creación de las tablas auxiliares en la plataforma Piet, se describe el marco teórico en el que se basa la implementación y, se describen algunas herramientas relacionadas. En el Capítulo 3 se presenta la solución propuesta y arquitectura conceptual, mientras que en el Capítulo 4 se discute la implementación y la utilización de la misma. El Capítulo 5 presenta los experimentos realizados y se discute sus resultados.

La segunda parte comienza en el Capítulo 6, donde se describe en detalle el problema que se resuelve en la segunda parte de esta tesis para comprender la necesidad de la extensión de RE-SPaM, denominada RE-SPaM++. Luego se describe sintaxis y semántica de PietQL y RE-SPaM. En el Capítulo 7 se

[5]Spatial OLAP

presenta la extensión al lenguaje de expresiones, se brindan algunos ejemplos de uso. Se profundiza en los detalles de implementación en el Capítulo 8. Por último, en el Capítulo 9 se brinda la conclusión de algunas propuestas y mejoras a las soluciones implementadas.

M

Capítulo 2

Integración de datos espaciales y no espaciales

En este capítulo se describe el problema de integración de datos espaciales y no espaciales en la plataforma Piet. Se brindará el marco teórico en el cuál se basó la solución propuesta en esta tesis y por último, se presentan algunos trabajos relacionados.

2.1. Planteo del problema

Esta sección es una breve introducción de Piet[pie] y el modelo de datos que este utiliza. Luego se describe el problema a resolver en la primera parte de esta tesis.

2.1.1. Piet

Piet es una aplicación que permite realizar consultas que combinan datos espaciales almacenados en un GIS y datos multidimensionales almacenados en un data warehouse. Dichas consultas se especifican mediante un lenguaje de consultas denominado PietQl[GVZ08]. Específicamente permite cuatro tipos de consultas:

- Consultas integradas GIS-OLAP, de la forma "Obtener el total de ventas por producto en las sucursales que se encuentran en ciudades cruzadas por un río". Esta consulta involucra los distintos tipos de información (datos GIS y datos OLAP), y su resultado puede luego explotarse utilizando herramientas OLAP tradicionales.
- Consultas de Agregación Geométrica, como "total de sucursales en estados con más de tres aeropuertos", que sumariza valores asociados a geometrías.
- Consultas típicas de GIS.
- Consultas típicas de OLAP.

Las consultas típicas de OLAP son ejecutadas traduciendo las expresiones PietQL a los motores OLAP y GIS según corresponda.

Modelo de datos

El modelo de datos de Piet, en su estado inicial, se encuentra formado por la información de aplicación y la información geográfica sin relacionar (Figura 2.1). En la figura podemos ver la información geográfica, almacenada en forma de capas representadas cada una por una tabla: **usa_rivers** la capa de ríos, **usa_cities** la capa de ciudades, etc. y la información de aplicación fromada por una tabla de dimensión **stores** y una serie de tablas de hechos de ventas.

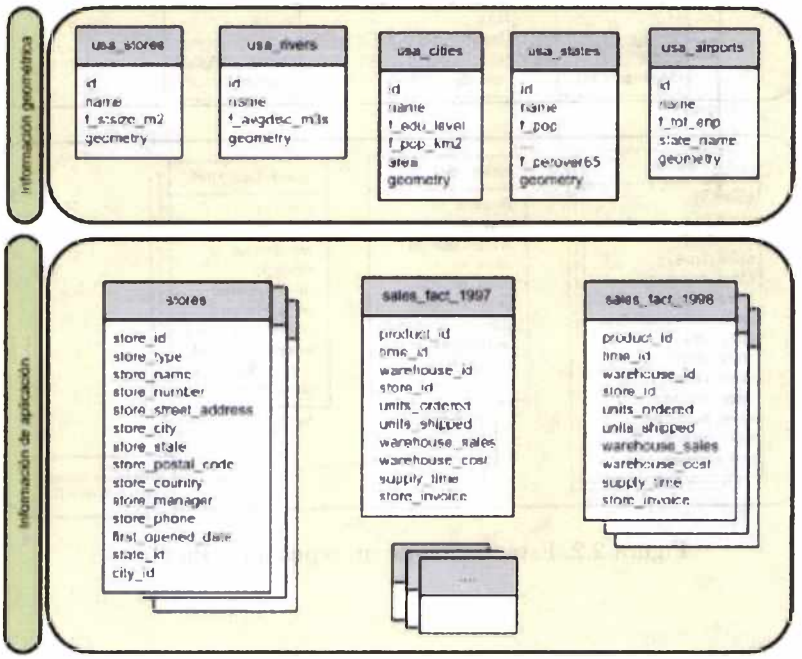


Figura 2.1: Estructura inicial del ambiente Piet

Basándonos en una estructura similar a la descripta en la Figura 2.1, el administrador Piet deberá crear asociaciones entre la parte geométrica y la parte de aplicación que quiera relacionar.
En la Figura 2.2 podemos ver el modelo de datos de Piet luego de la creación de las asociaciones. En la misma se puede ver que se agrega el área de *información de aplicación GIS-OLAP*. En esta área se almacenan las relaciones entre componentes geométricos y tablas OLAP por medio de los identificadores únicos de cada dato. Esta información es la que permite relacionar la parte de aplicación con la parte geométrica.

2.1.2. Descripción del problema

Como se describe en la sección anterior, Piet integra GIS con OLAP mediante las asociaciones que se muestran en la Figura 2.2. Estas tablas son creadas y completadas manualmente por el administrador Piet. Este proceso consta básicamente de:

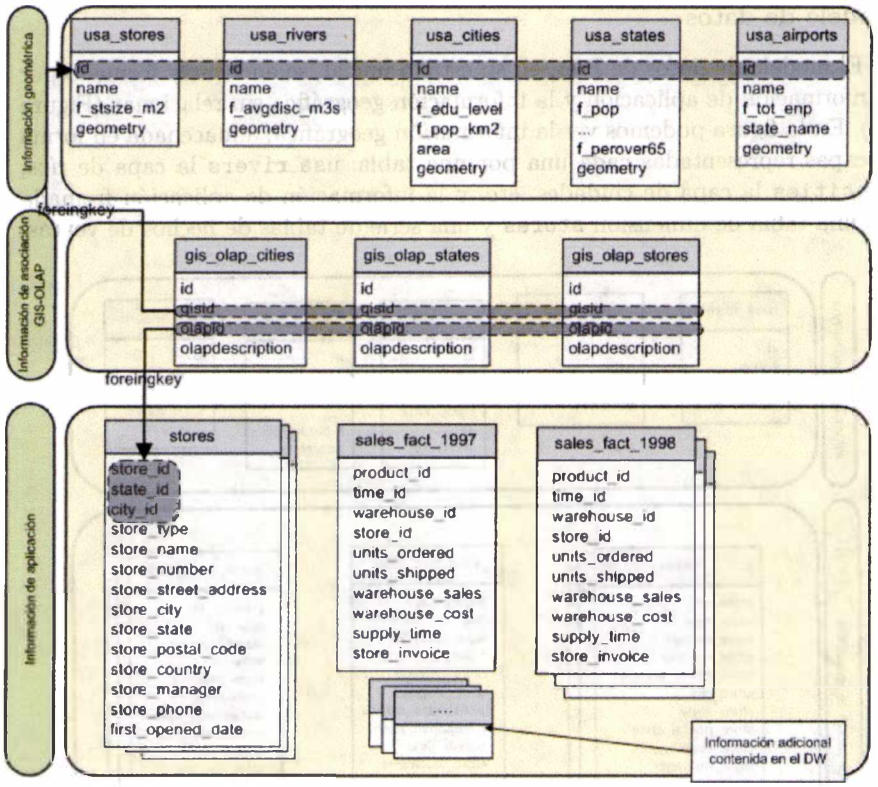


Figura 2.2: Estado final de un repositorio Piet

- Identificar qué tablas del GIS se relacionarán con qué tablas del data warehouse. Las tablas que se relacionarán y asociarán deben compartir un mismo dominio de información, es decir, los registros de las tablas representan las mismas entidades (ciudades, ríos, etc). En la Figura 2.2 la tabla **stores** del lado de OLAP se relaciona con otras tres del lado GIS. Esto se debe a que los registros se refieren almacenes que a su vez inlcuyen información de la ciudad y estado al cuál pertenece.

- Comparar todos los registros de una tabla con los de la otra identificando cuales corresponden a una misma entidad. Por ejemplo si se comparan ciudades, la ciudad **New York** debe asociarse con **NY**.

- Repetir el proceso hasta que no haya más tablas que comparar.

Este proceso tedioso, costoso y sujeto a errores es el que se propone automatizar integrando los datos del GIS con los del OLAP.

2.2. Marco teórico

Proponemos resolver al problema de integrar GIS y OLAP[1] mediante técnicas utilizadas en herramientas que permitan integrar datos (sin distinguir si son espaciales y no espaciales). A continuación describiremos conceptos relacionados y algunas técnicas (en especial las relacionadas a Record Linkage) que se utilizan en la solución implementada.

2.2.1. Conceptos y terminología

El término *Record Linkage*[2] se refiere al proceso de identificar registros que representan a una misma entidad (persona, ciudad, etc.) con el fin de eliminar duplicados (deduplication) en un archivo ó enlazar registros (linkage) entre dos archivos.

Durante un proceso de integración de datos[3] se comparan los registros de a pares. Se denomina *match* al par de registros que se refieren a la misma entidad y *link* al par de registros que **mediante algún proceso** se infiere que representan a la misma entidad. No todo math es un link, y no todo link es un match. En el Cuadro 2.1 se describen los posibles errores que se pueden cometer al clasificar un par de registros como link.

	match	no match
link	Correcto	Falso positivo
no link	Falso negativo	Correcto

Cuadro 2.1: Errores posibles en la clasificación de pares de registros.

La tarea del proceso de Record Linkage es la creación de links, por medio de la comparación, con el fin de determinar cuales son match.

2.2.2. Diagrama de procesos en un sistema de integración de datos

La Figura 2.3 muestra el flujo de trabajo típico en un proceso de integración de datos [EVE02]:

- *Normalización* se refiere al trabajo relacionado con la limpieza de datos.
- *Armado de bloques* descripto en detalle en la Sección 2.2.4, tiene como objetivo minimizar la cantidad de comparaciones a realizar.
- *Comparación* es el proceso de comparación de los registros. En la Sección 2.2.5 se estudia este proceso.

[1] La creación de las tablas de asociación, que se menciona en la sección 2.1.2

[2] Otros términos utilizados para referirse a Record Linkage son: entity heterogeneity, entity identification, object isomerism, instance identification, merge/purge, entity reconciliation, list washing and data cleaning.

[3] Que utiliza técnicas relacionadas a Record Linkage. Cuando nos referimos en esta tesis a un proceso de integración de datos, nos referimos a procesos que resuelven *linkage* ó *deduplication*.

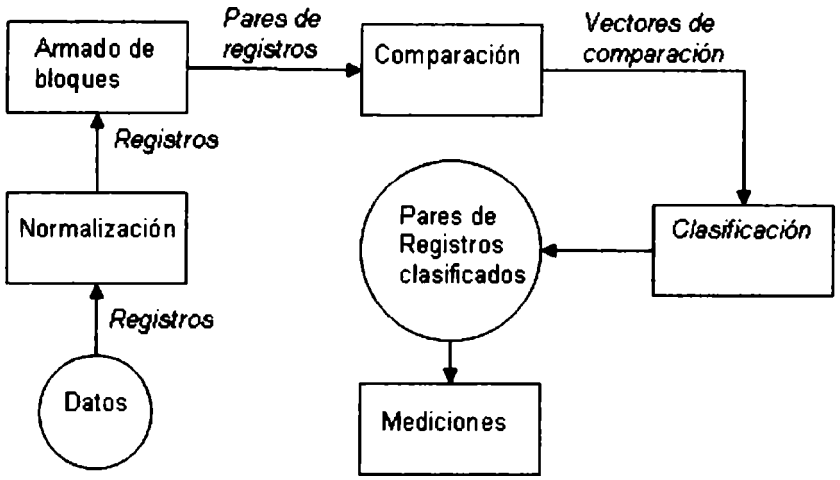


Figura 2.3: Diagrama de procesos de un sistema de integración de datos

- *Clasificación* es la etapa en que los pares de registros se clasifican y se decide si son o no un match.
- *Mediciones* representa todo lo relacionado a la calificación de los resultados.

El diagrama no es iterativo, pero hay implementaciones que permiten la iteración, como por ejemplo Automatch [Jar97].

2.2.3. Normalización

En muchos casos los datos de un campo, como por ejemplo el género, en un archivo puede ser M y F, y en el otro archivo Masculino y Femenino. Estos datos deben ser editados y depurados. Este etapa se conoce como *Normalización* y engloba los siguientes procesos.

Edición: Proceso relacionado a la detección y posible corrección de datos erroneos.

Parseo: Proceso relacionado a la unificación ó separación de campos para facilitar la comparación.

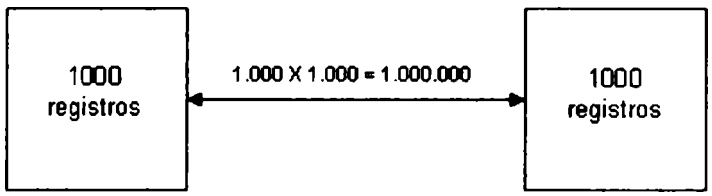
Formateo: Proceso a que permite hacer posible las comparaciones entre diferentes formatos. Por ejemplo 01/02/2009 con 20090201.

Concordancia: Proceso que se utiliza para unificar variables que representan algún código, como por ejemplo el sexo que puede estar codificado como 1 y 2 en un archivo y en otro como 'S' y 'M'.

Este proceso queda fuera de alcance del presente trabajo. Lectores interesados pueden consultar [BLN86], [KCGS93], [LNE89], [LC94] entre otros.

2.2.4. Armado de bloques

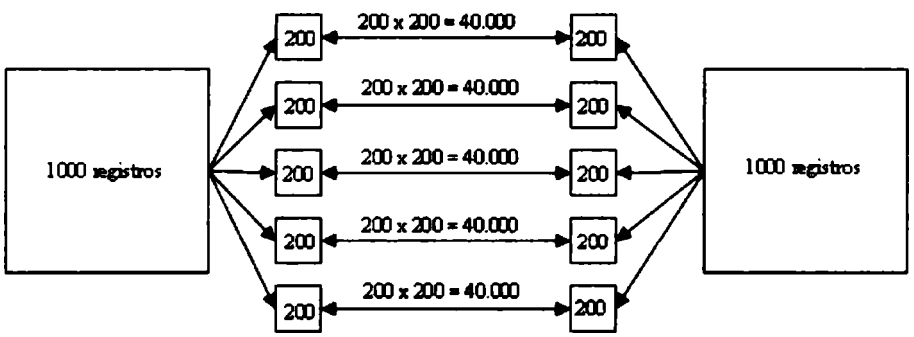
En un proceso de integración de datos es común que se realicen millones de comparaciones cuando sólo algunos miles de pares de registros pueden llegar a ser un *match*. Para reducir la cantidad de pares de registros a analizar se utiliza una técnica conocida como *armado de bloques*[GBVR03, p. 10]. Por ejemplo, si se comparan dos archivos de 1.000 registros se analizan 1.000.000 de pares como se muestra en la Figura 2.4:



Total de comparaciones = 1.000.000

Figura 2.4: Comparaciones entre dos archivos sin filtros.

La idea detrás del armado de bloques, es la aplicación de **filtros** que generen **bloques** para analizar pares de registros con mayor probabilidad de ser un match y así reducir la cantidad de comparaciones. Los filtros generan un bloque de acuerdo a los campos a los que se aplican. Por ejemplo, si se elije el campo sexo se generan dos bloques. Si se trabaja con el campo fecha filtrando por mes se generan doce bloques; con el campo fecha filtrando por mes y sexo 24 bloques. En la Figura 2.5 muestra la cantidad de pares que se analizarían al utilizar un filtro que genera cinco bloques de igual tamaño:



Total de comparaciones = 200.000

Figura 2.5: Comparaciones entre dos archivos utilizando un filtro que genera 5 bloques

2.2.5. Comparación

Para relacionar dos registos, se comparan[4] los atributos en forma individual y como resultado se asigna un valor que refleja la similitud entre el par de regitros. Este valor se denomina *peso del campo.* El cálculo del peso puede verse afectado por otros parámetros asociados al campo que dependen del *modelo de decisión*[5] que se este utilizando, como por ejemplo los parámetros *probabilidad m* y *probabilidad u* del modelo de Fellegi-Sunter [FS69]. Por lo tanto, el peso se calcula mediante alguna función que compara los campos e información asociada que depende del modelo de decisión. Una vez calculados los pesos de los campos, se calcula el peso asociado al par de registros. Por ejemplo, el modelo Fellegi-Sunter calcula el peso asociado al par sumando los pesos de los campos.

Si el valor de un atributo no es uniforme ó algún atributo tiene más relevancia que otro, se puede asignar un peso de importancia al campo [Win89]. Por ejemplo, el apellido Smith es más frecuente que el apellido Zabrinsky, por lo tanto una coincidencia en un apellido Zabrinsky debería tener más peso que la coincidencia de un apellido Smith. La idea es que coincidencias entre atributos de poca frecuencia sean más relevantes que aquellos con valores más frecuentes.

2.2.6. Clasificación y Modelo de Decisión

Generalmente, en un proyecto de integración de datos se procesan cientos de miles de registros y millones de pares son evaluados para determinar si son o no matchs. La gran mayoría de los pares no son match como se puede apreciar en la Figura 2.6[6]. Los pesos negativos indican que los pares no son match.

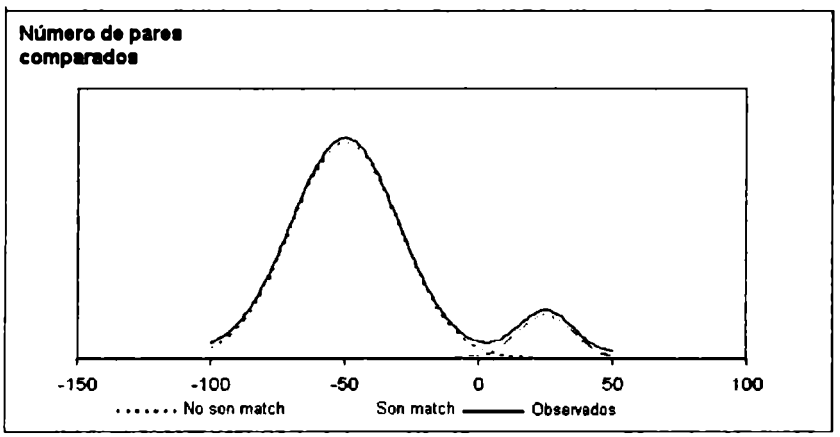


Figura 2.6: Distribución de los pesos compuestos a través de todos los pares posibles.

[4]Este no es el enfoque del presente trabajo. Para el lector interesado puede consultar [HSW07, cáp. 13], [CRF03]. En [sec] se encuentra una librería que implementa varias funciones de comparación implementadas en Java y más referencias.

[5]Ver sección 2.2.6

[6]Valor de los pesos en un análisis realizado por Statistics New Zealand en [Zea06, p. 43].

En la figura podemos apreciar la importancia de clasificar los pares de registros ya comparados (con peso asociado) en *match* o *no match* ya que la gran mayoría de los pares no son match. La *clasificación* se refiere a esta tarea y es realizada por el *modelo de decisión*[7].

2.2.7. Iteración

Una *iteración* es una nueva ejecución del proceso de integración donde se utilizan diferentes variables para el armado de bloques y posiblemente otras para el matching. Por ejemplo, una pasada podría utilizar el mes para blocking y el sexo y el nombre para matching. Otra pasada podría ser usar el sexo como blocking y el nombre y dirección para el matching.

El número de pasadas refleja la calidad del proceso de integración. Si sólo se obtiene pocos links en cada pasada, se necesitaría una gran cantidad de pasadas. Sin embargo podría ser un indicador que la calidad de los datos no es buena y no se necesitarían más pasadas. Estas decisiones quedan normalemente en responsabilidad del usuario.

2.3. Herramientas

En esta sección se verán herramientas relacionadas para observar similitudes y diferencias. De cada herramienta se comentan fortalezas y debilidades. La mayoría de los desarrollos se basan en el modelo propuesto por Fellegi y Sunter [FS69] que ya es un estándar [GBVR03, p. 2].

2.3.0.1. Febrl

Febrl (Freely extensible biomedical record linkage) v. 0.4 [Chr08] es una herramienta de código abierto gratuita construida en python[py]. Resuelve estandarización, deduplication y linkage.

Febrl provee varios métodos de armado de bloques (blocking), funciones de comparación y modelos de decisión que deben ser elegidos por el usuario.

Puntos fuertes :

1. Código abierto.
2. Provee una amplia cantidad funciones de comparación, varios métodos de blocking y modelos de decisión.
3. Permite exportar a un archivo la configuración del proceso en el cual se está trabajando.
4. Soporte de paralelismo.
5. Interfaz gráfica.
6. Multiplataforma al ser construido en el lenguaje python.

[7] El primer modelo de decisión formalizado fue propuesto por Fellegi y Sunter en el año 1969. Este modelo es implementado por la mayoría de los sistemas de integración de datos y es explicado en el Apéndice A.

7. Soporta iteración en su ejecución. Es decir una etapa se ejecuta y se detiene presentando sus resultados. Luego se puede continuar con la siguiente ó reejecutar la etapa modificando los posibles parámetros.

Puntos débiles :

1. No es simple incluir nuevas funciones de comparación, modelos de decisión, métodos de blocking y la forma de lectura de datos.
2. No soporta asignación única[8].
3. No soporta funciones de comparación a nivel registro.
4. No permite la ejecución desde línea de comandos.
5. Requiere entrenamiento por parte del usuario para la configuración de las diferentes funciones y modelos de desición.
6. No soporta internacionalización.

2.3.0.2. FRIL

FRIL(Fine-grained record integration and linkage tool [JLX+08] v. 1.0 (patch5) es una herramienta código abierto gratuita construida en Java. Resuelve deduplication y record linkage. Provee varias funciones de comparación y modelos de desición. Ofrece asignación manual y automática de pesos a los campos por medio del método EM [DLR77].

Puntos fuertes :

1. Código abierto.
2. Provee una amplia gama de funciones de comparación y modelos de decisión.
3. Estandarización embebida en el linkage.
4. Permite exportar e importar la configuración del proceso a un archivo.
5. Multiplataforma.
6. Interfaz gráfica sencilla.
7. Soporte de ejecución desde línea de comandos.
8. Combina el modelo de decisión con el método de blocking, lo cual simplifica al usuario el uso.

Puntos débiles :

1. No soporta asignación única.
2. No soporta funciones de comparación a nivel registro.
3. No es simple la inclusión de nuevas funciones y modelos de desición.
4. Combina el modelo de decisión con el método de blocking. No permite agregar un modelo de decisión sin implementar un método de blocking.
5. No soporta internacionalización.

[8]Que un registro no sea asignado más de una vez en la salida. Es decir, no haya dos o más pares clasificados como match que incluyan a un mismo registro

2.3.0.3. LinkPlus

LinkPlus [oCPC] es una herramienta que resuelve record linkage y deduplication orientada al campo de la salud, pero puede ser utilizada en otros ámbitos.

Puntos fuertes :

1. Posee interfaz gráfica.
2. Buena documentación de ayuda.
3. Varias funciones de comparación.
4. Provee de manera al usuario la asignación de los pesos a los campos y el cálculo de las probabilidades u y m tanto automática como manual, simplificando al usuario su uso.
5. Provee asignación de los pesos a nivel de campo basado en la frecuencia de los mismos ó mediante el método EM.
6. Permite proveer más de un método de blocking que se componen mediante un operador OR.

Puntos débiles :

1. No es código abierto.
2. No es extensible.
3. No provee asignación única.
4. No provee comparación a nivel registro.
5. Es obligatorio elegir un método de blocking.
6. Sólo se puede ejecutar en entornos Windows.
7. No provee ejecución desde línea de comandos.
8. No soporta internacionalización.

2.3.1. Discusión

Se han presentado algunas herramientas de record linkage con sus puntos fuertes y débiles. En la mayoría de los casos se encuentra una limitación o dificultad a la hora de querer extender la herramienta en diferentes direcciones, por ejemplo el simple hecho de querer agregar una nueva función de comparación de atributos. Otras posibilidades de extensión, no ofrecidas por las herramientas presentadas, son:

1. Incluir un modo de procesar los resultados. Si bien la salida a un archivo luego puede ser tratada con diferentes herramientas, sería deseable añadir acciones que puedan procesar los resultados de manera sencilla.

2. Si se deseara realizar una comparación de registros no triviales, sería, en algunos casos, más simple implementar una función de comparación "a medida". Incluir funciones de comparación entre registros debería ser una opción y delegar en la herramienta su ejecución y procesamiento. Además si la comparación basada en los campos de los registros de forma individual no es suficiente, se recurre a procesos de limpieza e integración de

campos con el fin de unificarlos y poder compararlos. Esta limitación se debe al modelo de comparación basados en atributos. El modelo debería basarse en la comparación entre registros directamente, logrando más flexibilidad y a su vez abarcando también al modelo basado sólo en los atributos de forma individual. Podrían realizarse funciones más complejas, por ejemplo, tomar dos campos de un registro, y en algunos casos lo procesen y luego lo comparen con otro atributo y en otros casos se ignoren ó procesen de manera diferente. Esta flexibilidad no es fácil de modelar e implementar si no se usa el registro completo en la comparación.

3. El modelo propuesto en [EVE02] (Figura 2.3) no admite incluir etapas intermedias. Etapas intermedias permitirían extender y/o reducir el comportamiento y adaptarse mejor a las diferentes necesidades.

4. En los casos que se debe comparar dos archivos que no poseen registros duplicados, las herramientas deberían ofrecer la opción de garantizar una asignación única entre registros, pero teniendo en cuenta que cada par de registros seleccionado, debe ser asociado con el par que tenga mejor peso. Esta funcionalidad no se ofrece en las herramientas analizadas.

5. Las herramientas descriptas no soportan datos espaciales.

6. Piet provee herramientas que a través de plugins se integran en Jump [Jum], que es la plataforma integradora por la que accede el usuario. De las herramientas analizadas, FRIL es la única implementada en Java y posible candidata a la integración. Lamentablemente esta integración es complicada debido que dentro de las clases del modelo se incluye código relacionado a la interfaz propuesta por FRIL, no al modelo que sigue Jump.

Estas puntos son resueltos en *Matcher*, el framework para record linkage implementado en esta tesis que permite la integracón entre GIS y OLAP para la plataforma Piet. En el siguiente capítulo se describe el framework cómo resuelve estas limitaciones, se utiliza y se configura.

Capítulo 3

Solución propuesta

En este capítulo se describe la solución propuesta al problema de relacionar datos espaciales y no espaciales.

3.1. Descripción

La solución propuesta para la integración de GIS y OLAP implementa una herramienta de integración de datos, denominada *Matcher*, utilizando las técnicas descriptas en el Capítulo 2 pero con algunas diferencias con respecto a las herramientas convencionales de integración de datos:

1. Sólo resuelve el problema de *linkage*, no *deduplication*, ya que no es necesario para resolver la integración entre GIS y OLAP. Las herramientas en general ofrecen resolver la solución a los dos tipos de problemas.

2. Permite eliminar gran cantidad de falsos positivos[1] al aplicar un algoritmo que garantiza *asignación única*(ver Sección 3.5.5). Las herramientas analizadas en la Sección 2.3 no ofrecen un modo de garantizar que un registro no sea asignado con dos o más registros. Por ejemplo, un registro de una tabla de GIS puede ser asignado con dos o más registros de una tabla del data warehouse, esto produce gran cantidad de falsos positivos ya que en las tablas se conoce de antemano que no hay duplicados.

3. Deja abierta la posibilidad de comparar registros utilizando información GIS ya que las funciones de comparación están parametrizadas. En otras herramientas los datos GIS no son soportados y agregar una nueva función de comparación requiere modificar el código de la aplicación que afecta a la interfaz. En *Matcher*, la extensibilidad fue el factor de mayor importancia al momento del diseño e incorporar funciones de comparación[2] desde la misma interfaz fue contemplado.

[1] Registros emparejados que el proceso de integración asumen que representan a una misma entidad pero en realidad son diferentes, más información en la Sección 2.2.1.

[2] Se permite incorporar y/o modificar la mayoría de los componentes que interactúan en el proceso de integración.

3.2. Arquitectura conceptual

Matcher se utiliza en la etapa de instalación de la plataforma Piet. En la Figura 3.1 se describe la comunicación con el sistema GIS, el data warehouse, la estructura auxiliar y el usuario que va a crear la estructura:

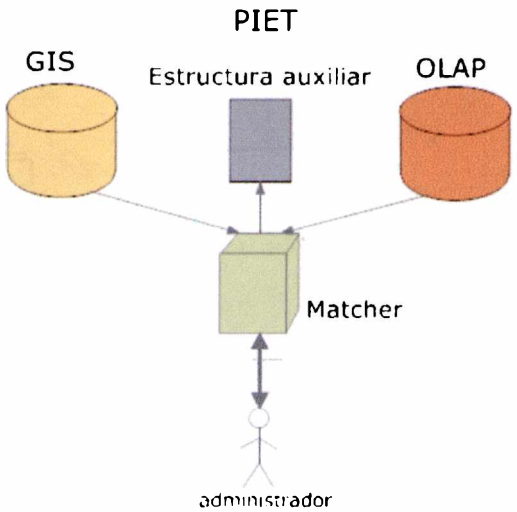


Figura 3.1: Interacción entre los componentes y el usuario en el momento de crear la estructura auxiliar.

El administrador interactuará con la herramienta indicando las tablas que se quieren relacionar del GIS y del data warehouse y, en donde se almacenará el resultado del proceso de integración.

3.3. Arquitectura implementada

En esta sección se resumen los componentes que forman parte de Matcher, incluyendo la plataforma de ejecución y las librerías externas que se utilizan para llevar adelante los servicios ofrecidos. En la Figura 3.2 se muestran los componentes y su interacción.

Matcher está implementado en el lenguaje Java y está compuesto por cuatro archivos Jar(Java ARchive)[3]

- **matcher-api:** Contiene interfases Java donde se define el funcionamiento de Matcher. No se incluye este componente en el diagrama porque no cumple un rol en la interacción con el usuario, sino es una práctica utilizada en el diseño que permite separar las interfases de la implementación.

- **matcher-core:** Implementación de los componentes definidos en matcher-api.

[3]Archivos JavaTM. Clases empaquetadas en un archivo. Más información en `http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html`

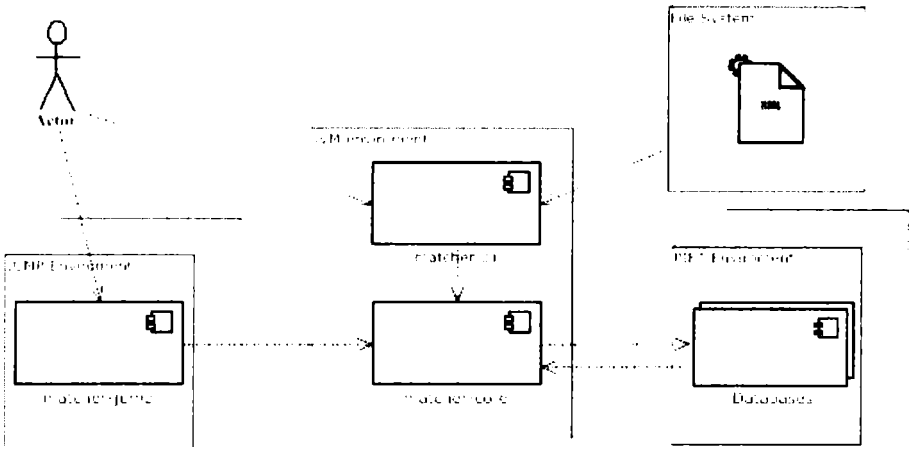


Figura 3.2: Diagrama de componentes de Matcher y su interacción

- **matcher-jump:** Es un plugin Jump[Jum] que provee una interfaz de ventanas para utilizar a Matcher.
- **matcher-cli:** Es un archivo que permite operar con Matcher desde línea de comandos.

La parte central de la implementación (matcher-core) es un Jar que se ejecuta en una Java Virtual Machine (JVM)[jvm] a través de alguna de las interfases implementadas. La librería matcher-core implementa las interfases definidas en matcher-api (JAR) y utiliza una serie de librerías externas para llevar adelante sus tareas. Las librerías más importantes son: Plexus[ple], commons-configuration[comb], commons-logging[comc], Castor[cas] y ostermiller-utils[ost].

La interfaz matcher-cli es un Jar que permite ejecutar procesos de integración mediante el archivo de configuración de Matcher. La librería más importante es args4j[arg].

La interfaz matcher-jump debe ser utilizado en una instalación JUMP, junto con una serie de librerías externas, como por ejemplo: JUMP[Jum], commons-logging, etc.

3.4. Funcionamiento

En la Figura 3.3 se muestra los procesos implementados en Matcher durante la ejecución de una tarea.

A diferencia del diagrama planteado en en [EVE02] (Figura 2.3), incorpora el proceso de *Asignación única* y un proceso de *Ejecución de Acciones*[4]. Otra diferencia es que no se incluye el proceso de *Normalización*, se decidió no implementar este proceso porque asumimos que los datos pueden ser procesados previamente.

[4]Este proceso podría ser el proceso denominado *Measurament* en TAILOR.

Figura 3.3: Diagrama de procesos de la ejecución de una tarea

La asignación única es un proceso opcional en Matcher. Se decidió así para brindar mayor flexibilidad y convertir a Matcher en una herramienta de integración de datos general (que no solo aplica a OLAP y GIS) y por lo tanto, no se puede garantizar que en las fuentes de datos a comparar no existan registros duplicados. Los algoritmos de asignación única utilizados se describen en la Sección 4.3.

3.5. Etapas de ejecución

La implementación actual no soporta iteraciones en la ejecución, a diferencia de algunas herramientas, como por ejemplo Febrl (ver Sección 2.3.0.1) que ejecuta una etapa y luego el proceso de integración se detiene presentando información asociada. Esta cualidad permite tomar decisiones para las siguientes etapas ó repetir su ejecución con diferentes parámetros, pero más importante es que permite realizar varias veces la clasificación de un mismo resultado de comparación.

A continuación se describen en detalle los objetivos y funcionamiento de las etapas de la Figura 3.3.

3.5.1. Lectura de datos

Es la primera etapa de la ejecución de la integración. Como su nombre indica, lee las fuentes de datos previamente configuradas y las almacena temporalmente en memoria. Soporta la lectura desde:

- Base de datos: Se puede leer la información de una base de datos vía JDBC[5].
- Ficheros CSV (del inglés comma-separated values).
- Ficheros dbf (archivos dBase[dba]).

Pero puede ser extendido como con otros tipos de fuentes como se describirá más adelante.

[5] *Java Database Connectivity* (JDBC) es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

3.5.2. Armado de bloques (blocking)

La implementación de blocking se basa en el uso de filtros[6]. Los filtros son funciones de comparación de registros (ver Sección 3.5.3) que cuando la función indica que no son un *match*[7] el par es descartado. De esta manera se impide que las siguientes etapas procesen al par descartado logrando reducir procesamiento y por ende el tiempo de ejecución final. En el caso de que alguna de las funciones indique lo contrario (es decir no clasifica el par como *no match*), el par es luego procesado por las siguientes etapas. Esta etapa es opcional.

3.5.3. Comparación

La comparación entre los pares de registros es el punto crítico y más importante de un proceso de integración. En Matcher la comparación se realiza mediante las funciones de comparación de registros. Una *función de comparación de registros* cumple dos roles:

1. **Comparar** los registros devolviendo un valor.
2. **Clasificar** el resultado de la comparación indicando si el par comparado es o no un match. (Ver la Sección 3.5.4)

Formalmente una función de comparación se define como:

Definición Sea f una función de comparación de registros y $r_1 \in A$ y $r_2 \in B$, donde A, B son las fuentes de datos, f se define como:

$$f : AXB \longmapsto \mathbb{R}$$

La función aplicada a los registros r_1 y r_2 devuelve un número que indica la similitud entre ambos registros. Las funciones deben retornan un valor mayor cuando exista más probabilidad que los registros representen a la misma entidad[8].

3.5.4. Clasificación

La clasificación es la etapa encargada de tomar cada par de registros y clasificarlos en una de las siguientes opciones:

- *Match*: Esta clasificación indica que el par es un match.
- *Possible match*: Esta clasificación indica que probablemente sea un match, pero aconseja una revisión por parte del usuario para asegurarse[9].
- *No match*: Esta clasificación indica que el par no es un match.

[6]En realidad en el armado de bloques también interviene el *método de indexación*, en la Sección 3.7 se describe el mismo

[7]Un *match* entre registros significa que ambos registros representan la misma entidad

[8]Si no cumplen esta reestricción no se puede ejecutar con resultados favorables la Asignación Única. En caso de no ejecutarse esta etapa, no es necesario satisfacer la restricción.

[9]Esta clasificación surge del modelo de decisión propuesto por Fellegi-Sunter, para más información ver el Apéndice A

El componente encargado de esta clasificación es la función de comparación de registros descripta en la Sección 3.5.4. La función además de asignar un valor al par, es responsable de clasificar el mismo. Formalmente lo expresamos como:

Definición Sea f una función de comparación de registros, f se define como:

$$f : \mathbb{R} \longmapsto \begin{cases} match \\ possible\ match \\ no\ match \end{cases}$$

En Matcher cada función de comparación debe clasificar el resultado de la comparación entre dos registros. No es posible clasificar las comparaciones entre diferentes funciones porque el resultado de las comparaciones no requieren que devuelvan el valor entre un rango definido, sino que mayor sea el resultado de la comparación, mayor sea la probabilidad de que el par sea un match. Formalmente si f_1 y f_2 dos funciones de comparación de registros, no es posible asegurar que $f_1(f_2(x,y)) = f_2(f_1(x,y))$, donde x, y registros de las diferentes fuentes de datos.

Notar que una función de comparación de registros representa un *modelo de decisión*, por lo tanto en Matcher habrá tantos modelos de decisión como funciones de comparación entre registros se hayan implementado.

3.5.5. Asignación única

La etapa denominada *Asignación Única* asegura que al finalizar su ejecución cada registro clasificado como *match* o *possible match* está asignado con exactamente un sólo registro y la asignación es la mejor alternativa posible. Formalmente decimos que:

Definición Sea M el conjunto de pares de registros resultado del proceso de asignación única y, A y B fuentes de datos; decimos que $\forall(r_1, r_2) \in M$ y $\forall(r_1, y)$, $(x, r_2) \in AXB - \{(r_1, r_2)\}$ se debe cumplir lo siguiente:

1. $\nexists(x, y) \in M$

2. (r_1, r_2) es la mejor alternativa posible para r_1 y/o r_2.

La segunda condición representa *la mejor alternativa posible* y depende del proceso de asignación utilizado.

3.5.6. Ejecución de acciones

La etapa *ejecución de acciones* es opcional y se ejecuta al finalizar la clasificación (ó la asignación única si esta presente) se encarga de ejecutar las acciones, previamente configuradas, asociadas a un proceso de integración. Las acciones pueden utilizarse para exportar los resultados a diferentes tipos de almacenamientos, realizar mediciones, etc.

3.6. Modelo de decisión en Matcher

Como se menciona en la Sección 2.2.6, el modelo de decisión es, dentro de una herramienta que implementa *Record Linkage*, quien se encarga de clasificar los **links** en **match** ó **no match**.

En Matcher se implementa el modelo de decisión mediante *las funciones de comparación de registros*. A diferencia de las herramientas analizadas en la Sección 2.3, un modelo de decisión es configurable dentro de un *proceso de integración de datos* ejecutado en Matcher y, el mismo está basado en el registro completo y no limitado a los atributos en forma individual.

3.7. Ejecución de las etapas Armado de bloques y Comparación

En Matcher, las etapas de *Armado de bloques* y de *Comparación*, en conjunto son las más costosas en relación al procesamiento de datos, ya que procesan cada par de registros. En el caso que se procesen tablas de gran volumen de datos, es probable según las funciones de comparación utilizadas, el proceso de integración sea muy lento. Si bien el Armado de bloques por medio de filtros permite reducir la cantidad de pares de registros que se compararán en la etapa Comparación, igualmente procesarán todos los registros de una fuente de datos contra todos los registros de la segunda fuente de datos.

En caso de procesamiento de grandes volúmenes de datos existen técnicas que, dadas ciertas circunstancias, permiten reducir considerablemente la cantidad de pares de registros a procesar. Por ejemplo la técnica conocida como *Sorted neighborhood*[HS95] ordena las tablas por algún criterio, y luego compara todos los registros que se encuentran en una *ventana* de tamaño w que va "avanzando" a medida que se comparan los registros. En la Figura 3.4 se grafica este modo de procesamiento de registros.

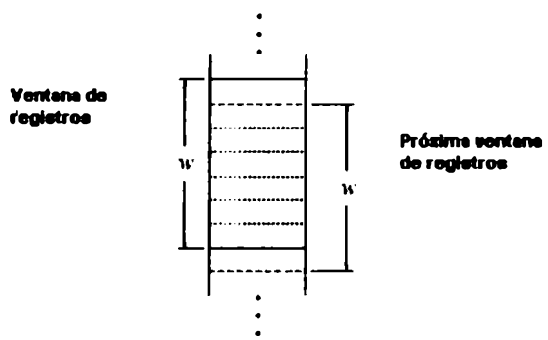


Figura 3.4: Procesamiento de los registros a través de una ventana de tamaño w

3.7.1. Método de indexación

En Matcher se soportan las técnicas, conocidas como *Blocking method*[10] (en algunas herramientas se denomina *Index*), mediante el *Método de indexación* o *Método de comparación*. El *método de indexación* es el encargado de procesar los registros de ambas fuentes y, de aplicar los filtros y funciones (si los filtros así lo indican) a cada par de registros.

[10]Tanto la aplicación de filtros como el método de comparación implementan el Armado de de bloques

Capítulo 4

Descripción de la implementación

En este capítulo se presenta un framework extensible, denominado *Matcher*, que permite resolver el problema de integrar fuentes de datos de distinto tipo, en particular, para el caso que nos interesa en esta tesis, fuentes de datos espaciales y multidimensionales.

4.1. Introducción

Matcher es un framework que provee una infraestructura para ejecutar procesos de integración, proveyendo un modelo de configuración, estático y dinámico, de componentes. El componente central es la *tarea* (MTask), que representa la configuración de un proceso de integración de datos. El rol de Matcher es "orquestar" la ejecución, el pasaje de datos y resultados entre los diferentes componentes involucrados en la ejecución de una tarea.

4.1.1. Proceso de integración

La configuración de un proceso de integración se modela en Matcher por el componente `MTask`. Este componente contiene la información necesaria para que Matcher lleve a cabo su ejecución. La información que incluye una tarea es:

1. Un identificador.
2. Las diferentes fuentes de datos.
3. Las funciones que se utilizan para comparar los registros.
4. Opcionalmente, los filtros y el método de indexación.
5. Opcionalmente, un proceso que garantiza la asignación única.
6. Opcionalmente, acciones que se ejecutan después de la etapa de Clasificación.

Con esta información, una *tarea* puede ser ejecutada dentro del ambiente Matcher en la serie de pasos definidas en la Figura 3.3.

4.2. Integración de datos entre dos archivos

Esta sección, mediante un caso práctico, introduce las carácteristicas de Matcher. Mediante el ejemplo se facilita la introducción de los diferentes componentes y los puntos de extensión.

El ejemplo se basa en la integración de dos archivos, **paises1** y **paises2**, que contienen datos sobre países obtenidos de diferentes fuentes[1].

4.2.1. Iniciando *Matcher*

En el ejemplo se utiliza la interfaz gráfica implementada en el módulo **matcher-jump**[2]. La instalación se describe en el Apéndice B. Una vez instalado y abierto Jump se selecciona en el menú la opción **Piet->tools->Matcher** que muestra la pantalla principal indicada en la Figura 4.1.

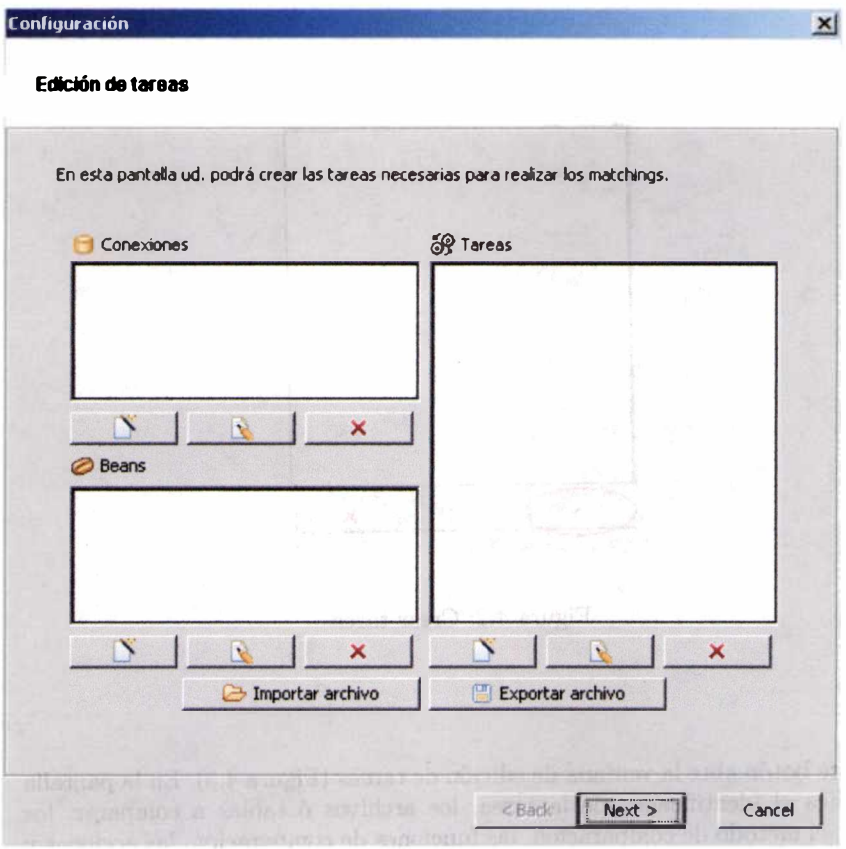


Figura 4.1: Pantalla inicial

En la pantalla se encuentran las opciones de importación y exportación

[1] En el Apéndice C se detallan los contenidos de los archivos.
[2] Ver la sección 3.3

de archivos XML de configuración[3], edición de tareas y la configuración de Matcher. En el ejemplo se muestra como crear una tarea identificada como `rl-paises`, para integrar ambos archivos.

4.2.2. Creación de una tarea

Con el botón Nueva tarea (Figura 4.2) se abre la pantalla de edición de tareas para configurar la nueva tarea.

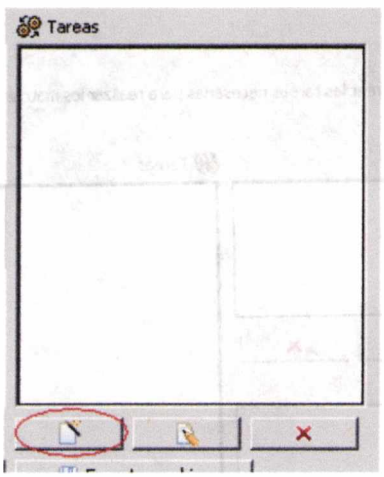


Figura 4.2: Crear tarea

Este botón abre la ventana de edición de tareas (Figura 4.3). En la pantalla se indica el identificador de la tarea, los archivos ó tablas a comparar, los filtros, el método de comparación, las funciones de comparación, las acciones y opcionalmente el método utilizado para la asignación única.

[3]Este tipo de archivos se detalla en el Apéndice B

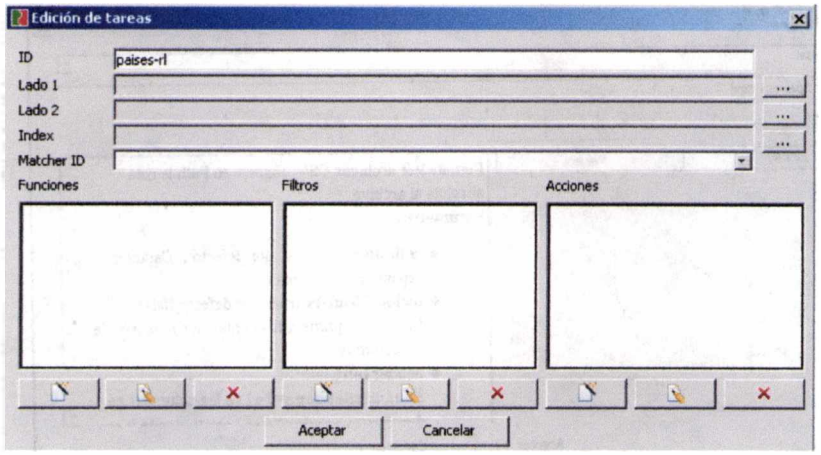


Figura 4.3: Pantalla de edición de tarea

4.2.2.1. Selección de los archivos a comparar

Para seleccionar los archivos a comparar, se debe presionar el botón ... para cada uno de los archivos, como se muestra en la Figura 4.4.

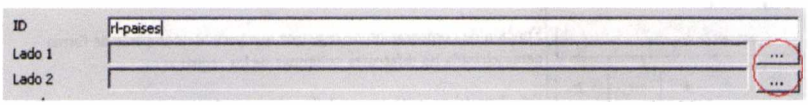


Figura 4.4: Abrir pantalla de fuente de datos

Al presionarlo se abre la ventana denominada *Fuente de datos*, Figura 4.5. En el cuadro de texto *Bean* se indica el tipo de fuente de datos y en *Path* la ruta de acceso al mismo. Las opciones para *Bean* se describen en el panel *Descripción*[4]

4.2.3. Incorporación de funciones de comparación

Una vez seleccionados los archivos, se debe indicar como comparar los registros. Los registros se comparan con las funciones que se hayan agregado a la tarea. El botón nueva función[5] abre la pantalla de la Figura 4.6.

Las funciones deben ser identificadas unívocamente dentro de una tarea. En este ejemplo se utiliza la función `IndividualColumnScorer`[6]. Esta función requiere especificar qué columnas comparar y cómo. Para ello, el botón *Mapeo de columnas* abre la pantalla de *Edición de mapeo de columnas* que completa como se muestra en la Figura 4.7. Para completar el panel de parámetros se utilizan el botón + para agregar, el botón - para quitar el que este seleccionado y el botón ? (autocompletar) ayuda y agiliza la carga de los mismos. Este modo de carga se repite en otras pantallas.

[4]En la Sección `Internacionalización` del Apéndice B se describe como configurar el texto que describe al bean y los pasos para que soporte multiidioma.

[5]El botón nuevo se representa por un papel y una varita.

[6]Las funciones de comparación entre registros se detallan en el Apéndice B

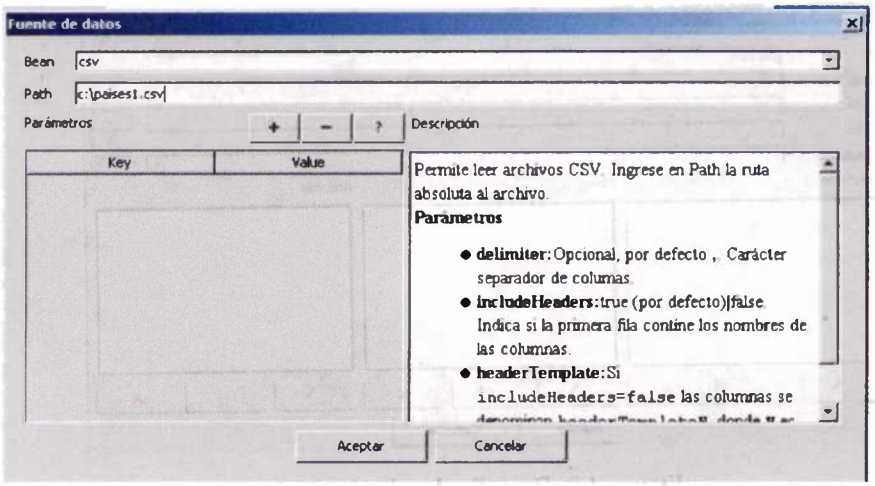


Figura 4.5: Pantalla de fuente de datos

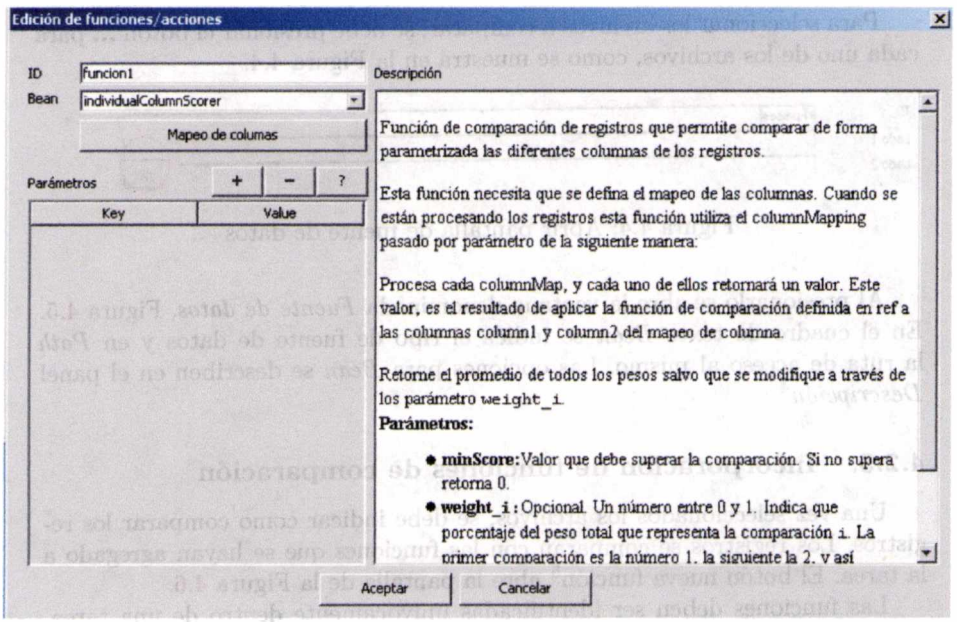


Figura 4.6: Pantalla de edición de función de comparación entre registros

En este ejemplo se selecciona la función `jaroWinklerTFIDF` que se utiliza para comparar campos de registros que contienen palabras donde no importa el orden de las mismas. En esta pantalla las funciones que se despliegan en la lista *Bean*[7] son funciones de comparación de atributos. En el Apéndice B se

[7]El nombre *Bean* se puede tomar como sinónimo de componente. El nombre proviene del ámbito Java, que es el lenguaje de programacón en la que se implementó Matcher.

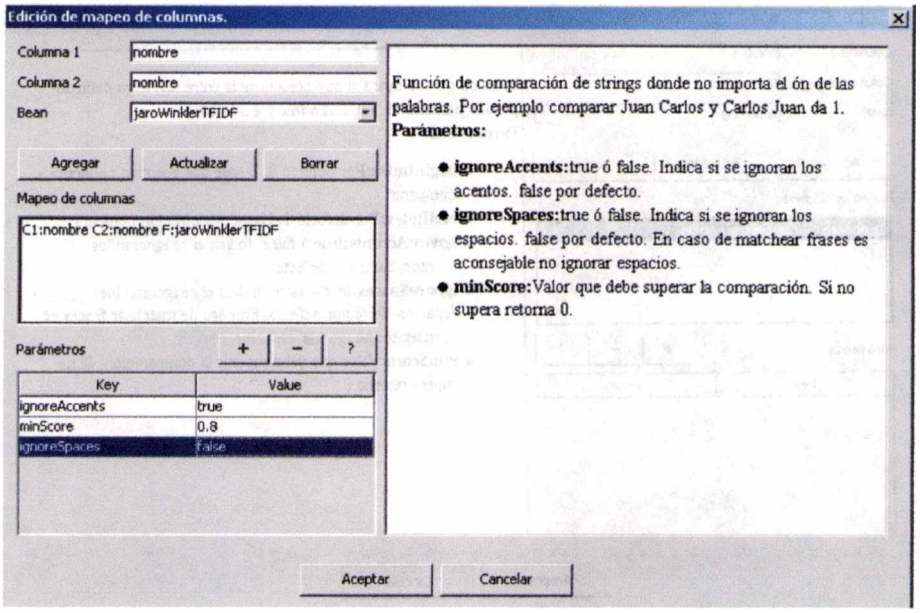


Figura 4.7: Pantalla de edición de función de comparación entre atributos

especifican las funciones provistas en *Matcher*.

4.2.4. Agregando filtros

En Matcher los filtros se utilizan para minimizar la cantidad de comparaciones que se realizarán entre los dos archivos. Si no se aplican filtros, la comparación se realiza en la forma "todos contra todos". En caso de pocos registros los filtros son innecesarios, pero no cuando entran en juego muchos registros. La Figura 4.8 presenta la pantalla de edición de filtros de la tarea.

Los filtros son en realidad funciones de comparación entre registros, pero en general son más simples y rápidas que las que se utilizan para la clasificación de los registros. Los filtros son procesados por el componente seleccionado en el campo *Index* de la pantalla descripta en la Figura 4.3. Este componente es el que realiza las comparaciones entre los registros y aplica los filtros.

4.2.5. Modo de comparación

El modo de comparación, como se describió en la Sección 3.7.1, se utiliza para minimizar la cantidad de comparaciones a realizar y por lo tanto el tiempo de ejecución. Matcher soporta actualmente el método de "todos contra todos" implementado en en el componente *fullIndex*, pero puede ser extendido con otros. El método *fullIndex* aplica el/los filtro/s (si los hay) a cada par de registros. Si la comparación devuelve `MATCH`, los registros son comparados con la función (puede haber más de una) que clasifica el par; en cambio si la comparación no es `MATCH` el par es descartado. En la Figura 4.9 se muestra la pantalla de selección del modo de comparación.

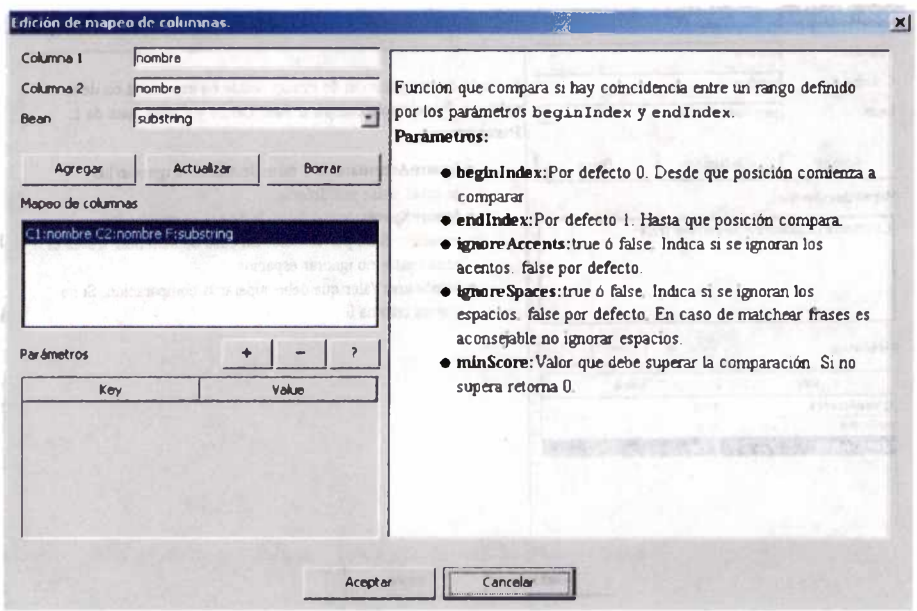


Figura 4.8: Mapeo de columnas para el filtro *subString*

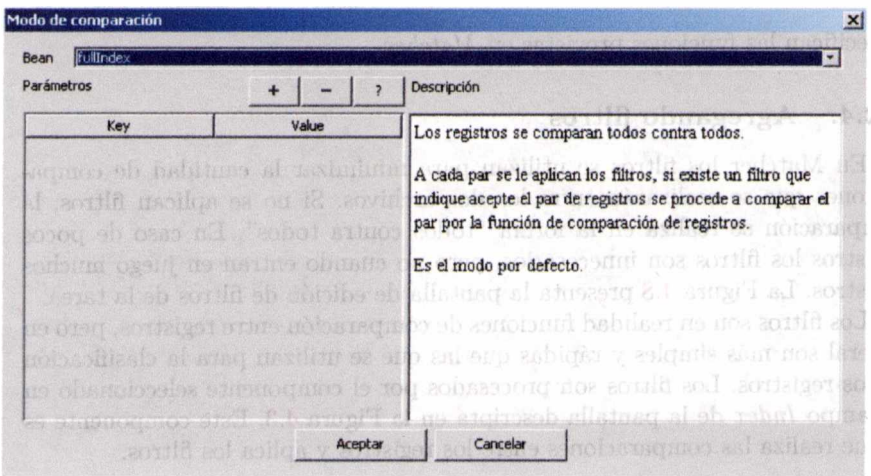


Figura 4.9: Pantalla donde se elige el modo de comparación

En el Apéndice B se describen los modos de comparación provistos en Matcher.

4.2.5.1. Agregando acciones

Las acciones son procesos que se ejecutan al finalizar la comparación de los archivos. La acción seleccionada en el ejemplo de la Figura 4.10 es `exportToFile`,

que sirve para exportar el resultado a un achivo. Esta acción requiere el parámetro **path** que indica la ruta al archivo a crear con los resultados. Los resultados que exporta a un archivo corresponden a los pares que se clasificaron como `MATCH`. En el Apéndice B se describen los diferentes tipos de resultado y las acciones provistas por *Matcher*.

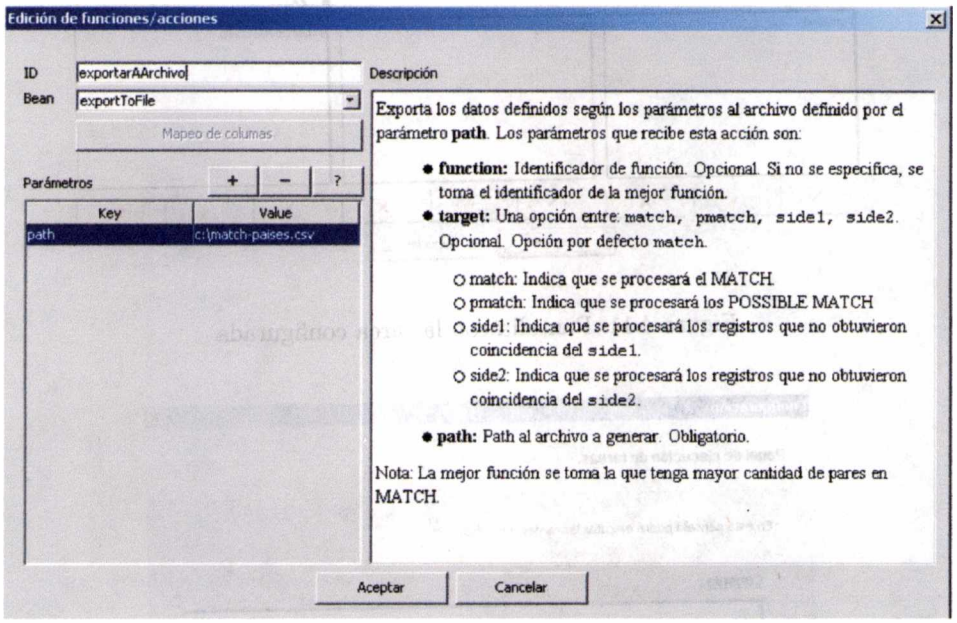


Figura 4.10: Pantalla de edición de acciones

4.2.6. Ejecución una tarea

Una vez configurada la tarea como se muestra en la Figura 4.11, estamos en condiciones de ir al panel de ejecución (mediante el botón *Next* >).

El panel de ejecuón de tareas (Figura 4.12) permite seleccionar la tarea, ejecutarla y detenerla cuando durante su ejecución. Asimismo, despliega en pantalla los mensajes de ejecución, que ayudan a detectar posibles errores y determinar en que etapa del proceso de ejecución se encuentra[8]. Al ejecutar la tarea se visualizá los mensajes como se muestra en la Figura 4.13.

4.2.7. Resultados de la ejecución

Al ejecutar esta tarea en los resultados se presentan los siguientes falsos positivos[9], que es común en las herramientas de record linkage:

```
nombre,id,nombre,id,SCORE
...
```

[8]Las etapas de ejecución de una tarea se describen en el Apéndice B

[9]Los falsos positicos son links clasificados, erroneamente, como matchs. Para más información ver la Sección 2.2.1

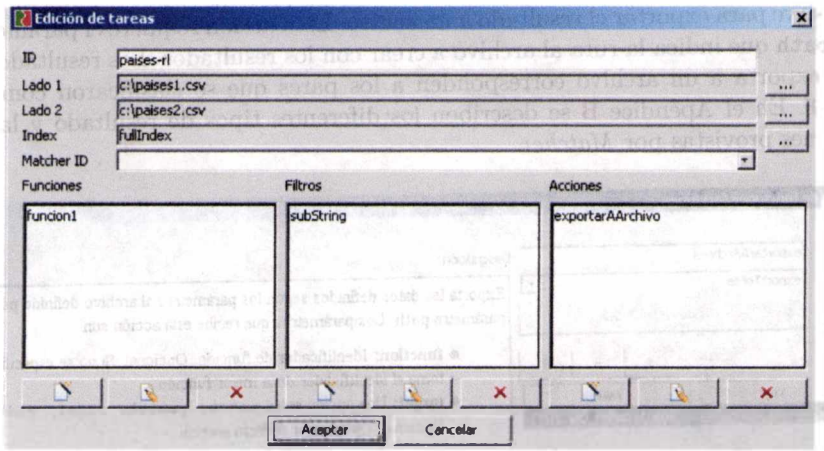


Figura 4.11: Pantalla con la tarea configurada

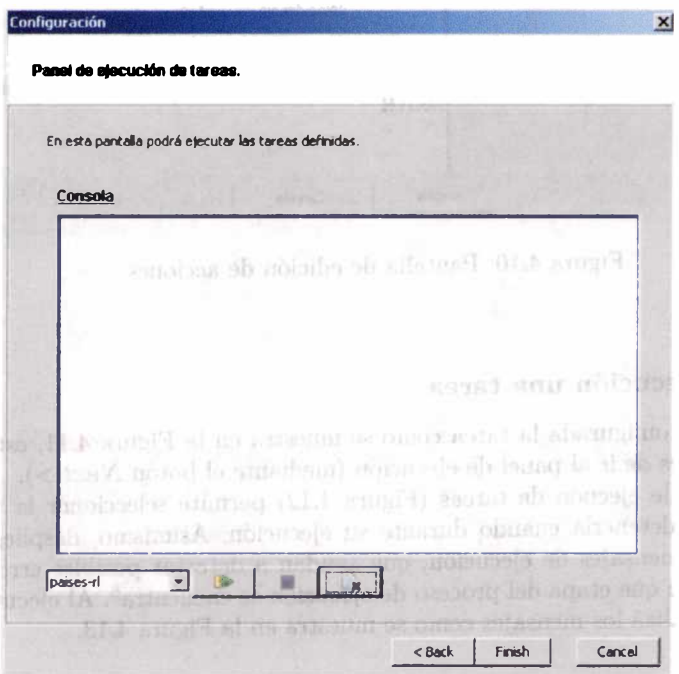


Figura 4.12: Panel de ejecución

```
"MALASIA","3260","Mali","138",0.9
"MALASIA","3260","Malasia","136",1.0
"MALASIA","3260","Malawi","135",0.9095238
...
"MALAWI","1250","Mali","138",0.9222222
```

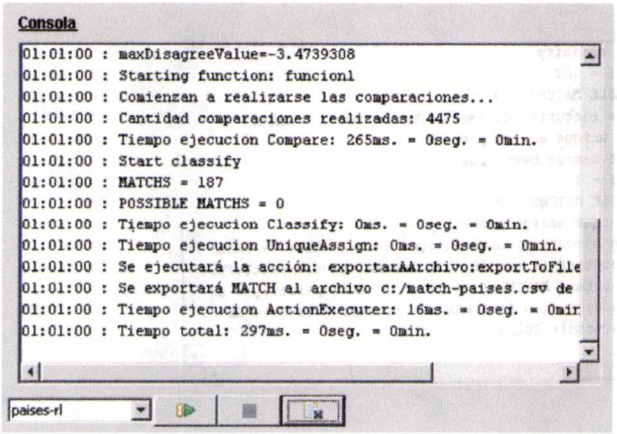


Figura 4.13: Panel de ejecución

```
"MALAWI","1250","Malasia","136",0.9095238
"MALAWI","1250","Malawi","135",1.0
...
```

para evitar este tipo de problemas se puede asignar a la tarea un proceso que garantize la asignación única.

4.2.7.1. Asignación única

Si se desea que la aplicación devuelva una asignación única en el resultado, Matcher provee dos componentes (pqm y munkres)[10] que se pueden seleccionar como se muestra en la Figura 4.14. Estos algorimtmos se describen en la Sección 4.3.

Figura 4.14: Selección del proceso de asignación única

Al ejecutar nuevamente la tarea, configurando pqm, el resultado obtenido se muestra en la Figura 4.15.

Y en la salida desaparecen los falsos positivos:

```
nombre,id,nombre,id,SCORE
...
"MALASIA","3260","Malasia","136",1.0
...
"MALAWI","1250","Malawi","135",1.0
```

[10]En el apéndice B se describen pqm y munkers con más detalle.

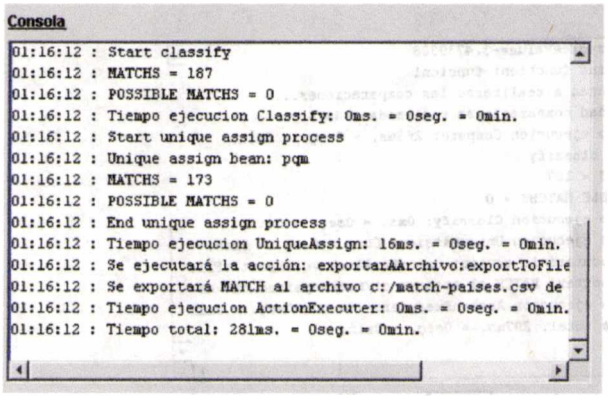


Figura 4.15: Ejecución con asignación única

...

4.3. Algoritmos para garantizar asignación única

En Matcher se ofrecen algoritmos para garantizar asignación única. El algoritmo de Munkres[Mun57], también denominado algoritmo húngaro [Kuh55], y el desarrollado exclusivamente para el problema de asignación única denominado *PQM*. A continuación se detallan ambos algoritmos.

4.3.1. Algoritmo PQM

El algoritmo procesa los registros clasificados como `MATCH` y `POSSIBLE_MATCH`. Cada par es almacenada en una cola de prioridad donde el orden es definido por el resultado de la comparación entre los registros[11].

Una vez almacenados se comienza a desencolar las comparaciones. Cada vez que se desencola se verifica que los elementos de la comparación no hayan sido previamente desencolados, y si es así se agregan a la salida del proceso. En el Algoritmo 1 describe el funcionamiento de PQM.

4.3.1.1. Ejemplo

Para presentar el ejemplo supongamos que se comparan dos fuentes de datos *Side*1 y *Side*2, donde *Side*1 tiene los registros $x_1, ..., x_5$; y *Side*2 los registros $y_1, .., y_4$. El proceso de asignación se ejecuta una vez comparados y clasificados los pares de registros. El grafo de la Figura 4.16 muestra la clasificación entre los pares, donde los vértices representan a los registros y las aristas representan al par y el peso asociado es valor obtenido a partir

[11] Si la función de comparación no cumple con la condición de que un par clasificado con mayor valor no indique mayor probabilidad de match, este algoritmo no funciona correctamente

Input: Cola de prioridad p conteniendo pares de registros ponderados por una función de comparación de registros
Output: El conjunto *Match* donde se cumple asignación única entre registros

```
filas = ∅ ;
columnas = ∅ ;
while ¬(vacía p) do
    elem := desencolar un elemento de p ;
    r1 := registro del lado 1 de elem ;
    if ¬ (r1 ∈ filas) then
        r2 := registro del lado 2 de elem ;
        if ¬ (r2 ∈ columnas) then
            agregar r1 a filas ;
            agregar r2 a columnas ;
            agregar elem a Match ;
        end
    end
end
```

Algorithm 1: Algoritmo PQM

de las funciones de comparación. Se asume que todas las aristas son MATCH ó POSSIBLE_MATCH.

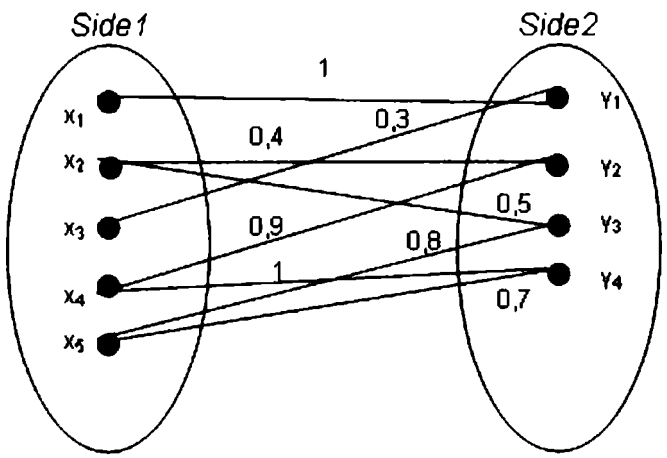


Figura 4.16: Grafo de costos

Estado inicial

$heap = \{(x_1, y_1, 1), (x_4, y_4, 1), (x_4, y_2, 0{,}9), (x_5, y_3, 0{,}8), (x_5, y_4, 0{,}7), (x_2, y_3, 0{,}5), (x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3)\}$

filas $= \emptyset$

columnas $=\emptyset$

resultado = $\emptyset$

1°

elemento := *heap.pop*(). *elemento* = $(x_1, y_1, 1)$

Luego del chequeo:

$heap = \{(x_4, y_4, 1), (x_4, y_2, 0{,}9), (x_5, y_3, 0{,}8), (x_5, y_4, 0{,}7),$
$(x_2, y_3, 0{,}5), (x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3)\}$
$filas = \{x_1\}$

$columnas = \{y_1\}$

$resultado = \{(x_1, y_1, 1)\}$

2°

elemento := *heap.pop*(). *elemento* = $(x_4, y_4, 1)$

Luego del chequeo:

$heap = \{(x_4, y_2, 0{,}9), (x_5, y_3, 0{,}8), (x_5, y_4, 0{,}7), (x_2, y_3, 0{,}5),$
$(x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3)\}$
$filas = \{x_1, x_4\}$

$columnas = \{y_1, y_4\}$

$resultado = \{(x_1, y_1, 1), (x_4, y_4, 1)\}$

3°

elemento := *heap.pop*(). *elemento* = $(x_4, y_2, 0{,}9)$

Luego del chequeo: No se agrega a resultado $(x_4, y_2, 0{,}9)$ porque $x_4 \in filas$.

$heap = \{(x_5, y_3, 0{,}8), (x_5, y_4, 0{,}7), (x_2, y_3, 0{,}5),$
$(x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3)\}$
$filas = \{x_1, x_4\}$

$columnas = \{y_1, y_4\}$

$resultado = \{(x_1, y_1, 1), (x_4, y_4, 1)\}$

4°

elemento := *heap.pop*(), *elemento* = $(x_5, y_3, 0{,}8)$

Luego del chequeo:

$heap = \{(x_5, y_4, 0, 7), (x_2, y_3, 0{,}5), (x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3))\}$
$filas = \{x_1, x_4, x_5\}$

$columnas = \{y_1, y_4, y_3\}$

$resultado = \{(x_1, y_1, 1), (x_4, y_4, 1), (x_5, y_3, 0{,}8)\}$

5°

$elemento := heap.pop()$, $elemento = (x_5, y_4, 0{,}7)$

Luego del chequeo: No se agrega a resultado $(x_5, y_4, 0{,}7)$ porque $x_5 \in filas$.

$heap = \{(x_2, y_3, 0{,}5), (x_2, y_2, 0{,}4), (x_3, y_1, 0{,}3))\}$
$filas = \{x_1, x_4, x_5\}$

$columnas = \{y_1, y_4, y_3\}$

$resultado = \{(x_1, y_1, 1), (x_4, y_4, 1), (x_5, y_3, 0{,}8)\}$

Siguiendo este procedimiento se obtiene :

$heap = \emptyset$
$filas = \{x_1, x_4, x_5, x_2\}$

$columnas = \{y_1, y_4, y_3, y_2\}$

$resultado = \{(x_1, y_1, 1), (x_4, y_4, 1), (x_5, y_3, 0{,}8), (x_2, y_2, 0{,}4)\}$

El resultado obtenido satisface los requerimientos necesarios para ser una solución.

4.3.2. Munkres

El algoritmo de asignación de Munkres (conocido también como algoritmo Húngaro), resuelve en realidad **el problema de asignación**. Este problema se puede definir de la siguiente manera:

Definición Hay un número de agentes y un número de tareas. Cualquier agente puede ser asignado para desarrollar cualquier tarea, contrayendo algún coste que puede variar dependiendo del agente y la tarea asignada. Es necesario para desarrollar todas las tareas asignar un solo agente a cada tarea para que el coste total de la asignación sea minimizado.

Como se puede apreciar, el algoritmo no resuelve directamente nuestro problema. Por tal motivo, se describirá cóomo aprovechar Munkres para aplicarlo al problema que nos incumbe.

4.3.2.1. Conceptos y definiciones

Un *grafo bipartito* $G = (V, E)$ se denomina a un grafo no dirigido cuyos vértices se pueden separar en dos conjuntos disjuntos V_1 y V_2 y las aristas siempre unen vértices de un conjunto con vértices de otro:

- $V_1 \cup V_2 = V$
- $V_1 \cap V_2 = \emptyset$
- $\forall x_1, x_2 \in V_1, \forall y_1, y_2 \in V_2$ no existe ninguna arista $e = x_1, x_2$ ni $e = y_1, y_2$.

El problema de asignación es un caso especial del problema denominado *matching*.

Un *matching* ó conjunto independiente de arcos, es un grafo es un conjunto de arcos sin vértices en común.

Dado un grafo $G = (V, E)$ un matching M en G es un conjunto de arcos disjuntos no adyacentes. Decimos que un vértice está *matched acoplado* si es incidido por un arco en el matching. Sino el vértice está *unmatched libre*.

Un *matching máximo* es un matching que contiene el número máximo posible de arcos. Puede haber muchos matchings máximos. El número de matching de un grafo es el tamaño del matching máximo. Un *matching maximal* es un matching M de un grafo G con la propiedad de que si algún arco que no pertenece a M es añadido a M, no será ya un matching. Nótese que todos los matching máximos deben ser maximales, pero no todos los matching maximales deben de ser máximos.

Un *matching perfecto* es un matching que cubre todos los vértices del grafo. Esto es, cada vértice del grafo es incidido por solo un arco del matching. Cada matching perfecto es máximo y maximal.

4.3.2.2. Matchings en grafos bipartitos ponderados

Encontrar un matching máximo bipartido ponderado en un grafo bipartido $G = (V = (X, Y), E)$ está definido como un matching perfecto donde la suma de los valores de sus arcos en el matching tiene un valor maximal. Si el grafo no es completamente bipartido, los arcos ausentes son introducidos con valor cero. Encontrar tal matching es conocido como **problema del asignación**.

4.3.2.3. Algoritmos que resuelven el problema de asignación

El problema del asignación es un caso especial del *problema del transportador*[Mon81], que es un caso especial del *problema del flujo de coste mínimo*. Es posible resolver cualquiera de estos problemas mediante el *algoritmo simplex*[KM72], aunque cada especialización tiene algoritmos más eficientes aprovechando su estructura particular. Algunos algoritmos que resuelven el problema del asignación son:

1. Algoritmo Húngaro [Kuh55] ó Algoritmo de Munkres para el problema de asignación (Algoritmo Khun-Munkres) [Mun57]
2. Algoritmo Simplex
3. Algoritmo Edmonds-Karp [EK72]
4. Algoritmo Ford-Fulkerson [For56]

Se optó por el algoritmo de Munkres como base para nuestra solución por ser el más especializado, es decir, el algoritmo requiere grafos bipartitos pesados (un conjunto de vértices representa a los agentes y el otro a las tareas) y en nuestro caso cada conjunto de vértices representa la fuente de datos y las aristas el peso calculado a partir de la función de comparación. Al aplicarse a esta estructura particular, el algoritmo de Munkres aprovecha la misma y es más rápido que los demás algoritmos que no requieren que el grafo sea bipartito ponderado.

La complejidad del algoritmo es $O(n^4)$. La demostración se puede ver en [Kuh55].

4.3.2.4. Adaptación del algoritmo de asignación de Munkres al problema de asignación única

El problema a resolver es, dados dos conjuntos de datos A y B, encontrar la asignación maximal entre los pares de registros. Si los agentes son los elementos del lado A, las tareas son los elementos del lado B y el coste es definido por la función de peso tendríamos:

Problema adaptado Hay un número de elementos $a \in A$ y un número de elementos $b \in B$. Cualquier a puede ser asignado a cualquier $b \leftrightarrow f(f(a,b)) \in \{MATCH, POSSIBLE\_MATCH\}, f$, función de comparación de registros, contrayendo algún coste que es definido por $f(a,b)$. Es necesario asignar un solo a a cada b para que el costo total del asignación sea **maximizado**.

Nótase que en nuestro caso es necesario que el costo sea maximizado y no minimizado. El algoritmo de Munkres se adapta fácilmente a la maximización en vez de la minimización.

4.3.2.5. Implementación del algoritmo húngaro provista en Matcher

La implementación del algoritmo húngaro utilizada es una modificación y optimización de la implementación del mismo incluida en el proyecto JigCell [jig]. La modificación realizada se debe a el algoritmo original no es performante en presencia de grandes volúmenes de datos y en nuestro caso es de vital importancia el tiempo ejecución.

La implementación varía las estructuras internas según la cantidad de datos para mejorar los tiempos y la utilización de recursos. Para ello se aplicó el patrón Strategy [EGV95] que selecciona un algoritmo u otro según la cantidad de datos.

Si el número de datos no es significativo[12] se utilizan matrices nativas del lenguaje para modelar el grafo. En cambio si son muchos datos se utilizan grafos dispersos. Para la implementación de grafos dispersos se utilizó la librería Universal Java Matrix Package UJMP [ujm] modificada para que sea compatible con Java 1.5. La versión original de UJMP es sólo compatible con Java 1.6 o superior, lo cuál es una restricción fuerte si se quiere lograr mayor compatibilidad. El motivo de la decisión de proveer 2 (dos) implementaciones se describe en el Apéndice F.

La implementación provista por JigCell y luego modificada se puede ver la explicación en los documentos públicos provistos por Iowa State University Iowa State University, específicamente en `http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html`.

[12]En nuestra implementación, el límite está definido en la propiedad `ar.edu.unlp.matcher.limitToUseNativeMatrixInByte` en el archivo `matcher.properties`.

4.3.3. Diferencias entre el algoritmo PQM y Munkres

La principal diferencia entre ambos algoritmos es que el algoritmo de Munkres devuelve un grafo maximal y PQM en cambio no lo asegura, aunque probablemente retorne un grafo maximal. El algoritmo PQM realiza todas las comparaciones (al igual que Munkres), las ordena y retorna los pares teniendo en cuenta que cada par incluido en el resultado no se encuentre en la salida del algoritmo. PQM comienza a armar el resultado con los pares con mayor probabilidad de acierto, pero es probable que sumando los pesos no sea maximal.

Esta diferencia es importante porque Munkres puede quitar pares de registros con valor alto por otros con menor valor para lograr que el grafo sea maximal, pero puede ser probable que al devolver un grafo maximal reduzca los posibles errores de las funciones de comparación. Como contempla grafos completos y no pares aislados los posibles errores a nivel dato sean menos significativos que en PQM. Por el contrario es probable que al intentar de devolver el grafo maximal quite pares de registros correctos y con valor alto.

Otra diferencia importante son los tiempos de ejecución y consumo de recursos. PQM al requerir menos cálculos es más rápido y consume menos recursos que el algoritmo de Munkres ya que necesita pocas estructuras internas para resolverlo.

Capítulo 5

Experimentos

A efectos de evaluar en forma práctica el comportamiento de la solución, se presentan una serie de experimentos realizados sobre diferentes conjunto datos. La idea fundamental es verificar el funcionamiento de los algoritmos de asignación única ya que las funciones de comparación de atributos son en general muy simples ó adaptaciones de otras librerías de funciones de comparación (secodstring, commons-codec).

5.1. Objetivos, equipo e información especificada en cada experimento

A continuación brindamos los objetivos que se plantean en esta sección, las características del equipo donde se realizaron las pruebas, y por último que se presenta en cada uno de los experimentos.

5.1.1. Objetivos de la experimentación

Los objetivos de los experimentos son:

1. Verificar que los tiempos sean compatibles con un uso razonable de la herramienta desarrollada.
2. Demostrar la flexibilidad de la herramienta.
3. Visualizar la calidad del resultado.

5.1.2. Ambiente para experimentación

El equipo utilizado para realizar la experimentación tiene las siguientes características:

CPU AMD 64 bits.

Sistema operatico Windows XP sp3

Memoria RAM 1Gb

Disco rígido 160Gb

5.1.3. Información asociada a los experimentos

En cada uno de los experimentos se especifica:

1. Los datos que se integran, que denominaremos lado 1 y lado 2. Es decir, en el uso previsto serán la componente GIS y la componente OLAP a integrar.
2. Las funciones de comparación que se utilizarán.
3. El algoritmo de asignación única utilizado.
4. El tiempo de ejecución.
5. Calidad del resultado, esto significa de todos los datos contenidos en el porcentaje de resultados, cuales son correctos (positivos), falsos positivos y falsos negativos, es decir, datos que deberían formar parte del resultado.
6. En caso de falsos negativos se indica de qué manera se podrían evitar y/o reducir.
7. Archivo de configuración XML utilizado.

Se denominará *Match* al conjunto resultado del proceso de integración de los datos del lado 1 y el lado 2.

5.2. Experimento 1

El primer experimento se quiere obtener el conjunto *Match* entre los condados[1] del data warehouse `FoodMart` y los condados del archivo `http://giscenter-sl.isu.edu/other/USA/USA_25M/CITIES/PAT.DBF`. El data warehouse `FoodMart` se encuentra en el proyecto Mondrian de Pentaho[mon].

5.2.1. Lado 1

El archivo PAT.dbf esta compuesto por condados de Estados Unidos, el cual tiene las siguientes columnas:

columna	tipo de dato	descripción
AREA	Número	área de la ciudad
PERIMETER	Número	périmetro de la ciudad
CTY2M_	Número	
CTY2M_ID	Número	
STATE_FIPS	Número	
CNTY_FIPS	Número	
FIPS	Texto	
STATE_NAME	Texto	Nombre del estado
CNTY_NAME	Texto	Nombre del condado
SUB_REGION	Texto	
STAT_FLAG	Númerico	

[1] Estados Unidos se divide en estados, cada estado se divide en condados y cada condado se divide en ciudades.

Los campos CTY2M_, CTY2M_ID, STATE_FIPS, CNTY_FIPS, FIPS, STAT_FLAG no serán utilizados en este experimento. La tabla contiene **4410 registros**.

5.2.2. Lado 2

Los condados de FoodMart se encuentran en la tabla del data warehouse region que tiene los siguientes campos:

columna	tipo de dato	descripción
region_id	Número	clave primaria
sales_city	Texto	nombre de la ciudad
sales_state_province	Texto	estado o provincia a la que pertenece la ciudad
sales_district	Texto	distrito de la ciudad
sales_country	Texto	país al cual pertenece la ciudad
sales_district_id	Texto	identificador de distrito.

Esta tabla contiene **110 registros** de ciudades pertenecientes a Estados Unidos, Méjico y Cánada. En este caso sólo nos interesa el condado asoicado a la ciudad que se encuentra en la columna sales_district.

La columna donde se definen los condados no se denomina sales_county debido a que contiene ciudades de diferentes países y cada uno con diferentes divisiones políticas. Se define sales_district como término medio.

5.2.3. Primer prueba

Para obtener el *Match* se debe comparar la columna CNTY_NAME de PAT.dbf con la columna sales_district de la tabla region del datawarehouse FoodMart. En esta primer prueba se ejecuta el proceso de integración utilizando las columnas CNTY_NAME y sales_district para verficar el resultado. A continuación especificamos la información asociada a esta prueba.

5.2.3.1. Función de comparación

IndividualColumnScorer con la función de comparación de atributos jaroWinklerTFIDF aplicada a CNTY_NAME del lado 1 y sales_district del lado 2.

5.2.3.2. Tiempo de ejecución

El tiempo total fue de 11 segundos utilizando el algoritmo PQM para la asignación única y de 13 segundos cuando se utiliza el algoritmo de Munkres. Se detallan los tiempos en cada etapa del proceso.

- **Lectura de datos**: $656ms$
- **Comparación de datos**: $10515ms$. Se realizaron 485100 comparaciones.
- **Clasificación**: 16ms
- **Asignación única**:
  - PQM: $16ms$
  - Munkres: $2391ms$

5.2.3.3. Resultado

Tanto PQM como Munkres retornaron exactamente los mismos resultados. En el Cuadro 5.1 se listan los resultados correctos mientras que en el Cuadro 5.2 se listan los resultados incorrectos (falsos positivos). No hubo falsos negativos; los condados no asignados de FoodMart no se encuentran en PAT.dbf, estos son:

- Bellingham, del estado Washington.
- Bremerton, del estado Washington. En realidad está bien que no se encuentre en PAT.dbf porque Bremerton no es un condado, sino una ciudad.

Los falsos positivos se deben a:

1. Hay dos registros que no son condados de Estados Unidos.
2. Los condados no pertenecen al mismo estado y el nombre no es igual.

La Figura 5.1 resume esta información y nos permite ver la calidad del resultado.

Lado1		Lado2			SCORE
STATE	CNTY	STATE	CNTY	COUNTRY	
Washington	Yakima	WA	Yakima	USA	1.0
Washington	Spokane	WA	Spokane	USA	1.0
Washington	Walla Walla	WA	Walla Walla	USA	1.0
California	San Francisco	CA	San Francisco	USA	1.0
California	Los Angeles	CA	Los Angeles	USA	1.0
California	San Diego	CA	San Diego	USA	1.0

Cuadro 5.1: Correctos

Lado1		Lado2			SCORE
STATE	CNTY	STATE	CNTY	COUNTRY	
Texas	Hidalgo	Zacatecas	Hidalgo	Mexico	1.0
Texas	Victoria	BC	Victoria	Canada	1.0
New Jersey	Salem	OR	Salem	USA	1.0
Virginia	Salem	OR	Salem	USA	1.0
Iowa	Bremer	WA	Bremerton	USA	0.93333334
New York	Cortland	OR	Portland	USA	0.9166667
Iowa	Tama	WA	Tacoma	USA	0.9111111
Idaho	Bingham	WA	Bellingham	USA	0.91

Cuadro 5.2: Falsos positivos

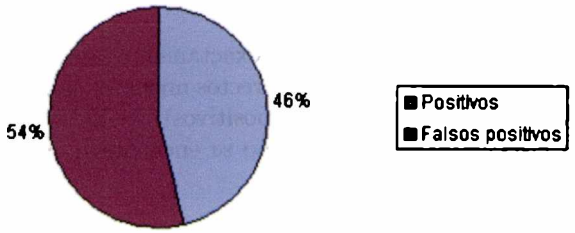


Figura 5.1: Calidad de los datos.

5.2.3.4. Conclusiones

Este experimento ha retornado todos los pares correctos, pero por el contrario ha retornado muchos falsos positivos. Es importante aclarar que la ejecución sin asignación única devuelve 267 pares de registros en el *Match* y cuando aplica asignación única se reduce a 25, es decir ha eliminado el 90 % falsos positivos (247 pares clasificados como correctos, pero no lo son), lo que demuestra la utilidad de la asignación única. Hay que aclarar que el total de registros que suman los resultados correctos (Cuadro 5.1) y los falsos positivos (Cuadro 5.2) suman 14, y no 25. Esto se debe a que se quitaron los repetidos de la salida original del proceso. Los repetidos producen pues en una tabla hay ciudades y en la otra condados. En el Cuadro 5.3 se muestran cinco filas que representan la misma información para este experimento.

Lado 1		Lado 2		
STATE	CNTY	STATE	CNTY	CITY
California	San Francisco	CA	San Francisco	Concord
California	San Francisco	CA	San Francisco	San Francisco
California	San Francisco	CA	San Francisco	Redwood City
California	San Francisco	CA	San Francisco	Burlingame
California	San Francisco	CA	San Francisco	Daly City

Cuadro 5.3: Tres filas que representan lo mismo para este experimento

En el Apéndice C se encuentran los archivos de configuración utilizados en esta prueba. En el próximo experimento se muestra cómo reducir los falsos positivos.

5.2.4. Prueba 2

En esta prueba demostraremos cómo quitar los falsos positivos de la prueba anterior. Básicamente se realizará:

- Descartar los registros que no sean de Estados Unidos, ya que en PAT.dbf se conoce que hay solamente condados de Estados Unidos.
- Utilizar el estado asociado al condado en las comparaciones.

5.2.4.1. Función de comparación

En esta prueba utilizamos la misma función que en la prueba anterior, pero en cambio se incluye un filtro para lograr quitar los falsos positivos de la prueba anterior. El filtro realiza lo siguiente: *Si coincide el primer carácter del estado o si el país (columna* `sales_country`*) de la tabla* `region` *es igual a USA, el par de registros procede a compararse y a clasificarse mediante las función de comparación. Caso contrario el valor asignado al par es 0 y por lo tanto no es tenido en cuenta.*

5.2.4.2. Tiempo de ejecución

El tiempo total es 1906 milisegundos utilizando el algoritmo PQM para la asignación única y de 2578 milisegundos cuando se utiliza el algoritmo de Munkres. Se detallan los tiempos en cada etapa del proceso.

- **Lectura de datos**: 656ms
- **Comparación de datos**: 1156ms. Se realizaron 16681 comparaciones. En esta prueba se redujo un 96.56 % la cantidad de comparaciones realizadas debido al filtro.
- **Clasificación**: 16ms
- **Asignación única**:
  - PQM: 15ms
  - Munkres: 656ms

5.2.4.3. Resultado

Este experimento retornó todos los resultados correctos sin falsos positivos.

5.2.4.4. Conclusiones

Este experimento ha retornado solo los pares correctos mediante el uso adecuado de los filtros. Hay que aclarar que solamente se puede realizar estos ajustes cuando ya se tiene un conocimiento previo de las características de los datos. En este caso el conocimiento de que en PAT.dbf sólo se refieren a condados de Estados Unidos y que en la tabla `region` los estados asociados coinciden en su primer carácter y no hay dos estados que comienzen con la misma letra. Por ejemplo si en `region` se encontraban estos estados:

1. Washington (WA)
2. West Virginia (WV)
3. Wisconsin (WI)
4. Wyoming (WY)

No hubiera alcanzado el filtro, aunque si reduce el tiempo de ejecución. Este problema es el que se resuelve en la siguiente prueba. En el Apéndice C se encuentran el archivo de configuración utilizados en esta prueba.

5.2.5. Prueba 3

En esta prueba demostraremos como quitar los falsos positivos de la prueba anterior si en la tabla region existen más de un estado que comienze con el mismo carácter.

5.2.5.1. Función de comparación

En esta prueba utilizamos los mismos filtros, la misma función que en la prueba anterior pero con dos funciones de comparación de atributos, es decir:

`IndividualColumnScorer` con la función de comparación de atributos `jaroWinklerTFIDF` aplicada a `CNTY_NAME` del lado 1 y `sales_district` del lado 2; y la función `SLIM` aplicada a la columna `STATE_NAME` y `sales_state_province`.

La función `SLIM` se utiliza para ponderar en la comparación el estado asociado a los condados. Como el estado en PAT.dbf esta completo y en la tabla `region` es el acrónimo, la función no tiene un peso muy confiable, sino que sirve para afectar al peso total asociado al par de registros. Por tal motivo en esta prueba se la función SLIM representará el 30 % del peso del valor calculado. En el Apéndice C se describe el archivo en donde se indica como realizar esta customización.

5.2.5.2. Tiempo de ejecución

El tiempo total es 2281ms milisegundos utilizando el algoritmo PQM para la asignación única y de 2937ms milisegundos cuando se utiliza el algoritmo de Munkres. Se detallan los tiempos en cada etapa del proceso.

- **Lectura de datos**: 656ms
- **Comparación de datos**: 1531ms. Se realizaron 15325 comparaciones.
- **Clasificación**: 16ms
- **Asignación única**:
  - PQM: 15ms
  - Munkres: 671ms

5.2.5.3. Resultado

Este experimento retornó todos los resultados correctos sin falsos positivos.

5.2.5.4. Conclusiones

Este experimento ha retornado los pares correctos y describe cómo resolver el problema en caso de que en la tabla region exista más de un estado que comienze con el mismo carácter. Otras alternativas son:

- Implementar una función de comparación de atributos que dado el nombre del estado retorne el acrónimo, ya que hay solamente 50 estados, no es una función compleja de desarrollar.
- Armar una vista SQL que aplica dicha función.

Lo que muestran estos experimentos es la flexibilidad de la herramienta para adaptarse a distintas situaciones y características de los datos. Las alternativas descriptas requieren trabajo por parte del usuario, pero serían más exactas.

5.3. Experimento 2

El siguiente experimento buscará obtener la relación *Match* entre los condados obtenidos del National Atlas of the United States[2] y los condados de `FoodMart`. En este caso, el número de registros a comparar es mucho mayor que en el experimento anterior.

5.3.1. Lado 1

Los condados de Estados Unidos obtenidos de National Atlas of the United States se encuentra en `http://coastalmap.marine.usgs.gov/GISdata/basemaps/usa/cities/citiesx020.htm`. El archivo `citiesx020.dbf` en realidad contiene ciudades, pero en este experimento relacionaremos los condados. El archivo cuenta con los siguentes campos:

columna	tipo de dato	descripción
CITIESX020	Número	
FEATURE	Texto	
NAME	Texto	nombre de la ciudad del condado
POP_RANGE	Texto	
POP_2000	Texto	población de la ciudad censado en el 2000
FIPS55	Texto	
COUNTY	Texto	nombre del condado
FIPS	Texto	código FIPS
STATE	Texto	estado del condado
STATE_FIPS	Texto	código FIP del estado
DISPLAY	Número	

La tabla de condados contiene **35432 registros**.

5.3.2. Lado 2

Ciudades del datawarehouse `Foodmart`.

5.3.3. Prueba 1

Esta prueba se realiza luego de una visualización previa de los datos. Los estados en `citiesx020.dbf` se expresan utilizando su acrónimo. La estategia es aplicar un filtro a los estados, que si comienzan con el mismo carácter prosiga, caso contrario descarte el par de registros analizados. De este modo se logra reducir la cantidad de comparaciones. La función de comparación se aplica por lo tanto tanto al campo condado como al campo estado de cada una de las tablas, como se describirá en la siguiente sección.

[2]National Atlas `http://www.nationalatlas.gov/index.html`

5.3.3.1. Función de comparación

Se aplica la siguiente función de comparación y el siguiente filtro:

Función de comparación: `IndividualColumnScorer` con las siguientes funciones de comparación de atributos:

- `jaroWinklerTFIDF` aplicada a `COUNTY` del lado 1 y `sales_district` del lado 2.
- `jaroWinkler` aplicada a `STATE` del lado 1 y `sales_state_province` del lado 2.

Filtro: `IndividualColumnScorer` con la siguiente función de comparación de atributos:

- `substring` aplicada a `STATE` del lado 1 y `sales_state_province` del lado 2.

5.3.3.2. Calidad del resultado

La calidad del resultado se muestra en la Figura 5.2. Los resultados fueron todos correctos (Cuadro 5.4), pero hubo falsos negativos (Cuadro 5.5). Tanto el algoritmo de Munkres como PQM retornaron el mismo resultado.

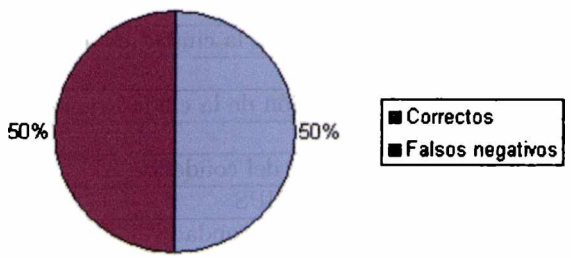


Figura 5.2: Calidad de los datos.

Lado 1		Lado 2		Score
State	County	County	State	
CA	San Francisco County	San Francisco	CA	0.9082483
CA	San Diego County	San Diego	CA	0.9082483
CA	Los Angeles County	Los Angeles	CA	0.9082483

Cuadro 5.4: Resultados correctos

5.3.3.3. Tiempo de ejecución

El tiempo total fue de 14 segundos utilizando el algoritmo PQM para la asignación única y el algoritmo de Munkres no finaliza porque utiliza más memoria de lo permitido (ver más adelante).

Lado 1		Lado 2	
State	County	County	State
WA	Spokane County	Spokane	WA
WA	Walla Walla County	Walla Walla	WA
WA	Yakima County	Yakima	WA

Cuadro 5.5: Falsos negativos

- **Lectura de datos**: 2438ms
- **Comparación de datos**: 12203ms. Se realizaron 15325 comparaciones.
- **Clasificación**: 31ms
- **Asignación única**:
  - PQM: 78ms
  - Munkres: No finaliza por un error relacionado al uso de memoria.

El error que ocurre en tiempo de ejecución del algoritmo de Munkres es:

```
java.lang.OutOfMemoryError: Java heap space
```

El error indica que el proceso en ejecución solicita más memoria de lo permitido. El máximo permitido es un parámetro de la máquina virtual de Java JVM[jvm], que es la encargada de la ejecución de las clases compiladas de Java. La memoria máxima permitida en esta prueba fue de 256M, como se muestra a continuación.

```
-vmargs -Xms40m -Xmx256m
```

Hay dos formas de ejecutar el proceso de Munkres[3]:

1. Indicar a Matcher la cantidad el límite en bytes que se puede utilizar en el algoritmo de Munkres. Este valor[4] se encuentra definido en el archivo matcher.properties.
2. Configurar la máquina virtual para que se pueda utilizar más memoria.

Los tiempos en cada caso fueron:

- **Modificando el valor en matcher.properties**: 246seg
- **Modificando los parámetros de la máquina virtual**: 78ms

[3]Esto se debe a la dos implementaciones del algoritmo, una tiene en cuenta la velocidad y la otra la minimización del uso de recursos

[4]Por defecto el valor es: ar.edu.unlp.matcher.limitToUseNativeMatrixInBytes=160000000

Es importante notar que a medida que hay más datos, el algoritmo de Munkres requiere más recursos. La implementación que utiliza menos recursos utiliza grafos dispersos[5] para modelar el grafo. La implementación de los grafos dispersos utilizada es de la librería UJMP[ujm] que lamentablemente es muy lenta. No se recomienda utilizar Munkres para muchos datos utilizando los grafos dispersos, para ello dejar un valor alto en el archivo matcher.properties. En las próximos experimentos solamente se usa la implementación de Munkres que no utiliza los grafos dispersos.

5.3.3.4. Conclusiones

En esta prueba, aunque no fue del todo satisfactoria debido a los falsos negaticos, es importante destacar que el proceso de integración genera 3420 pares de registros clasificados como links[6] y mediante la asignación única reduce a 30, es decir elimina el 99.12 % de pares que son falsos positivos. La cantidad de resultados es 30 y en el Cuadro 5.4 se listan sólo 3; el motivo es el mismo que el comentado en el primer experimento (ver Cuadro 5.3).

En la siguiente prueba se analiza y se demuestra como eliminar los falsos negativos.

5.3.4. Prueba 2

El problema de la prueba anterior se debe a que la clasificación entre los condados en algunos casos no superar el valor 0.8[7]. Para solucionarlo se indica que si la comparación entre dos atributos supera el valor 0.5, la misma es aceptable.

5.3.4.1. Función de comparación

Se aplica la misma función y el mismo filtro que en la prueba anterior, pero con la diferencia descripta en el párrafo anterior. El archivo de configuración utilizado se encuentra en el Apéndice C.

5.3.4.2. Calidad del resultado

Los resultados fueron todos correctos y no hubo falsos positivos ni falsos negativos. Tanto el algoritmo de Munkres como PQM retornaron el mismo resultado.

5.3.4.3. Tiempo de ejecución

El tiempo total fue de 14 segundos utilizando el algoritmo PQM para la asignación única y utilizando Munkres 18 segundos.

[5]En el Apéndice F se discute una solución a la excesiva utilización de recursos en la implementación del algoritmo de Munkres. Si bien se utiliza menos recursos, se ha utilizado una implementación lenta para modelar los grafos.

[6]Pares de registros que el proceso de integración indica que es probablemente un match.

[7]Valor por defecto utilizado en la función de comparación JaroWinklerTFIDF para determinar si el resultado de la comparación entre dos columnas es aceptable. En caso que el valor de la comparación sea menor retorna 0. Para más información leer Funciones de comparación de atributos en el Apéndice B.

- **Lectura de datos**: 2438ms
- **Comparación de datos**: 12203ms. Se realizaron 15325 comparaciones.
- **Clasificación**: 31ms
- **Asignación única**:
  - PQM: 31ms
  - Munkres: 1828ms

5.3.4.4. Conclusiones

En esta prueba se ha demostrado la flexibilidad de la herramienta para eliminar los falsos negativos. Hay que tener en cuenta que al reducir los valores límite de las funciones, existe una mayor probabilidad de falsos positivos al tener más pares para clasificar. Otro tema importante en estas últimas dos pruebas es la clara diferencia en el uso de recursos entre los dos algoritmos.

5.4. Experimento 3

El siguiente experimento buscará obtener la relación *Match* entre los condados obtenidos del National Atlas of the United States y los del archivo PAT.dbf.

5.4.1. Lado 1

Condados de USA tomados de National Atlas of the United States (citiesx020.dbf) que contiene 35432 registros.

5.4.2. Lado 2

Condados de USA tomados del giscenter-sl (PAT.dbf) que contiene 4410 registros.

5.4.3. Prueba 1

Esta prueba intenta probar la herramienta con volúmenes de datos más reales y probar la calidad de los datos, como así también los tiempos. Un análisis de los datos previo indica que en PAT.dbf hay 1804 condados. Esto nos servirá para saber cuantos datos aproximadamente debería devolverse en *Match*.

5.4.3.1. Función de comparación

La función de comparación aplicada es `IndividualColumnScorer` con las siguientes funciones de comparación de atributos:

- `jaroWinklerTFIDF` aplicada a `COUNTY` del lado 1 y `CNTY_NAME` del lado 2.
- `slim` aplicada a `STATE` del lado 1 y `STATE_NAME` del lado 2, ya que en un lado se utiliza acrónimos de estados y en el otro no.

Se aplica el filtro **substring** para comparar estados y condados que comienzen con la misma letra. En el Apéndice C se encuentra el archivo con la configuración mencionada.

5.4.4. Calidad del resultado

Tanto la ejecución de la integración con el algoritmo PQM y con Munkres retornaron exactamente el mismo resultado. Esta prueba fue exitosa, retornaron todos los resultados correctos, de los 1804 condados de PAT.dbf sólo uno no fue asignado provocando un falso negativo. El resto no asignado se debe a que no se encuentra en el archivo citiesx020.dbf. En la Figura 5.3 se resume la calidad; en este caso indica 0 %, pero en realidad hubo 1 (un) falso negativo (ver Cuadro 5.6).

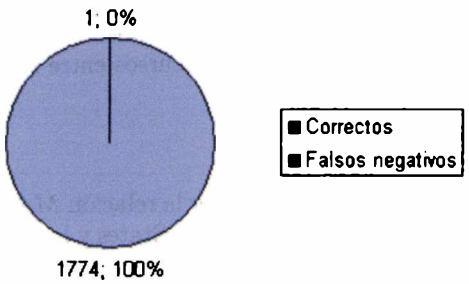


Figura 5.3: Calidad de los datos.

Lado 1		Lado 2	
State	County	State	County
IL	DuPage County	Illinois	Du Page

Cuadro 5.6: Falso negativo

5.4.5. Tiempo de ejecución

El tiempo de ejecución aproximado fue de 6 minutos (401359ms).

- **Lectura de datos**: $1172ms$
- **Comparación de datos**: $399547ms$. Se realizaron 233993 comparaciones (sin filtros se hubieran realizado $35432x4410 = 156255120$, es decir se redujo en 99,85 % la cantidad de comparaciones).
- **Clasificación**: $46ms$
- **Asignación única**:

- PQM: $313ms$[8]
- Munkres: $297ms$[9]

5.4.5.1. Conclusiones

En esta prueba se ha demostrado que la herramienta se ejecuta en tiempos aceptables y también la gran potencia de la asignación única, ya que previo a la asignación única habían 59519 pares de registros clasificados como MATCH[10], y luego del proceso de asignación única quedaron clasificados como MATCH solamente 3381.

Los falsos negativos cuando en el *Match* contiene casi la totalidad de registros de un lado (en este caso PAT.dbf), es sencillo detectarlos. Tomamos los resultados que se encuentran en el archivo original (del lado donde haya menos registros, en este caso PAT.dbf) y que no están en el Match para verificar cuales se encuentran en el otro archivo (citiesx020.dbf). En este caso había que revisar solamente 30 condados que estaban en PAT.dbf pero no en el *Match*. El problema del falso negativo se debe a que el condado *DuPage* y *Du Page* para la función JaroWinklerTFIDF son diferentes ya que en un caso es una palabra y en el otro dos. Estos errores se pueden solucionar potenciando el resultado con alguna función (por ejemplo slim). Debido a que solamente se obtuvo 1 falso negativo, se omite esta alternativa.

[8]Se ejecutó con los siguientes parámetros de la máquina virtual: `-Xmn40M -Xmx256M`

[9]Se ejecutó con los siguientes parámetros de la máquina virtual: `-Xmn100M -Xms800M -Xmx800M`

[10]Para más información acerca de la clasificación entre los pares ver la Sección 3.5.4

Capítulo 6

Integración de datos espaciales y de objetos móviles

En este capítulo se describe el problema que se resuelve en la segunda parte de esta tesis, esto es la integración entre datos espaciales y base de datos de trayectorias. Se introducirán también algunos conceptos asociados.

6.1. Introducción

La siguiente sección brinda una descripción del modelo de datos y el lenguaje y el lenguaje de consulta que utilizaremos en el resto de esta tesis.

6.1.1. Trayectorias semánticas

Los objetos móviles equipados con dispositivos electrónicos tales como GPS o RFID pueden reportar información sobre su ubicación espacio-temporal. Una trayectoria esta dada por una secuencia de posiciones en diferentes instantes. Dicha secuencia esta ordenada por la componente temporal. Los objetos móviles y sus trayectorias pueden almacenarse en lo que se denomina una base de datos de trayectorias, cuyas tuplas contienen la siguiente información:

o identificador del objeto móvil.

t un instante.

p una coordenada espacial (x,y).

En esta tesis trabajaremos con el conceto *trayectoria semántica*. Para ello la base de datos de trayectorias descripta es transformada, utilizando las ideas de [SPD+07], que asume que los objetos se mueven sobre un mapa que contiene geometrías disjuntas, que poseen información semántica en forma de atributos. Estas geometrías se denominan *Lugares de Interés de una Aplicación* (del inglés Places of Interest of an Application (PoIs)), porque dependen de la aplicación, ya que para un mismo mapa, dos aplicaciones podrían tener diferentes PoIs. Por ejemplo en una aplicación de turismo, los PoIs podrían ser restaurantes, hoteles y atracciones turísticas. Cuando un objeto permanece un tiempo mayor a un mínimo suficiente dentro de un PoI, el PoI es considerado como una parada

(stop) para la trayectoria y todas las coordenadas (x, y) que pertenecen al PoI se reemplazan por el identificador del POI. Las trayectorias así reescritas se denominan *trayectorias semánticas* ya que no solo están comprimidas sino que estan enriquecidas por la información adicional sobre los PoIs.

6.1.2. RE-SPaM

Regular Expression for Sequential Pattern Mining (RE-SPaM) [GV09] es un lenguaje basado en expresiones regulares aplicando algoritmos de descubrimientos de patrones secuenciales. Aplicado a trayectorias semánticas, los patrones secuenciales a descubrir son las secuencias de paradas (stops) seguidas habitualmente por los objetos móviles estudiados. El lenguaje se basa en expresiones regulares sobre restricciones aplicadas a las paradas. Las expresiones pueden incluir atributos, funciones y variables.

6.1.3. Modelo de datos en RE-SPaM

Para facilitar la explicación del modelo de datos tomaremos un caso en cual los PoIs corresponden a una aplicación de turismo en la ciudad de París. Los elementos a los cuales se aplican las consultas, precisamente secuencia de paradas que se componen de PoIs visitados por turistas, el intervalo de tiempo para cada objeto móvil en cada parada, y los atributos asociados a cada PoI.

El modelo de datos de RE-SPaM soporta la categorización de elementos, es decir que cada elemento puede ser clasificado como perteneciente a una categoría, descripta por un conjunto de atributos. Esto permite intituitivamente, presentar el esquema (la estructura), ocurrencia (una instancia concreta de una categoría, por ej. un hotel), e instancias (un conjunto de ocurrencias) de las categorías.

En los ejemplos se presentan cuatro categorías: hoteles, restaurantes, la torre Eiffel y zoológicos, con diferentes atributos y número de ocurrencias. En el Cuadro 6.1 se muestran las categorías y sus esquemas. Una instancia (el conjunto de ocurrencias) se muestra en el Cuadro 6.2. Cada vez que un turista atraviesa un PoI y se detiene en el mismo, esta información es almacenada. Más precisamente, cada parada de un objeto móvil es un objeto compuesto de atributos temporales (indicando la duración de la parada), y los atributos de la categoría. Cada elemento puede pertenecer exactamente a una sola categoría RE-SPaM se asocian a un algoritmo de mining que tiene el objeto de decubrir patrones frecuentes de trayectorias de objetos móviles, restringiendolos a aquellos que satisfagan ciertas restricciones expresadas por medio del lenguaje basado en expresiones regulares definidas sobre atributos de las paradas. Estas limitaciones son de la forma "Trayectorias que visitaron primero un restaurante económico, luego se detuvieron en un hotel 3 estrellas, y finalmente terminaron en el restaurante que comenzó la trayectoria".

Las instancias de las categorías contienen la geometría asociada (el atributo *geom*) que la define. Cuando los objetos móviles se detienen en dicha geometría durante un cierto tiempo, es considerada un stop y es cuando se registra un nuevo elemento en la trayectoria del objeto.

Nombre de la categoría	Esquema
hotels	[ID, categoryName, geom, star]
restaurans	[ID, categoryName, geom, foodType, price]
Eiffel Tower	[ID, categoryName, geom]
zoos	[ID, categoryName, geom, price]

Cuadro 6.1: Esquema de las categorías utilizadas en los ejemplos

Nombre de la categoría	Instancia
hotels (2)	[(ID,H1), (categoryName,hotel), (geom,pol1), (star,3)] [(ID,H2), (categoryName,hotel), (geom,pol2), (star,5)]
restaurants (3)	[(ID,R1),(categoryName,restaurant),(geom,pol3), (foodType,French),(price,cheap)] [(ID,R2),(categoryName,restaurant),(geom,pol4), (foodType,French),(price,expensive)] [(ID,R3),(categoryName,restaurant),(geom,pol5),(foodType,Italian),(price,cheap)]
Eiffel Tower (1)	[(ID,E), (categoryName,EiffelTower), (geom,pol6)]
zoos (1)	[(ID,Z), (categoryName,zoo), (geom,pol7), (price,cheap)]

Cuadro 6.2: Conjunto de instancias de las categorías del Cuadro 6.1

6.1.4. El lenguaje RE-SPaM

Para describir el lenguaje se presentarán algunos ejemplos.

Trayectorias utilizadas en los ejemplos

Basándose en las categorías del Cuadro 6.1 y en la instancia de la misma definida en el Cuadro 6.2, se utiliza en los ejemplos las trayectorias registradas dos objetos O_1 y O_2, en el Cuadro 6.3 se representan los recorridos para estos objetos, expresados como trayectorias semánticas.

Ejemplo 1

Q1: Trayectorias de turistas que visitaron el hotel H1, opcionalmente pasaron por el restaurante R3 y el zoológico, y al final visitaron el hotel H1 ó la torre Eiffel.

```
[ID='H1'].([ID='R3'].[ID='Z'])*.([ID='E']|[ID='H1'])
```

Notar que Q1 utiliza solamente atributos ID en todas las subexpresiones.

Ejemplo 2

Q2: Trayectorias que pasaron por el hotel H1, después, opcionalmente visitaron diferentes lugares, y finalizaron en la torre Eiffel ó regresaron a H1.

```
[ID='H1'].[]*.([ID='E']|[ID='H1'])
```

OID	Items
O_1	([(ts_date, 04/08/2008), (ts_time, 14:05), (tf_date, 04/08/2008), (tf_time, 14:33), (ID, R2), (categoryName, restaurant), (geom, pol4), (foodType, French), (price, expensive)]); ([(ts_date, 04/08/2008), (ts_time, 15:10), (tf_date, 04/08/2008), (tf_time, 16:05), (ID, E), (categoryName, EiffelTower), (geom, pol6)]); ([(ts_date, 04/08/2008), (ts_time, 17:30), (tf_date, 04/08/2008), (tf_time, 18:48), (ID, R3), (categoryName, restaurant), (geom,pol5), (foodType, Italian), (price,cheap)]); ([(ts_date, 08/08/2008), (ts_time, 06:22), (tf_date, 08/08/2008), (tf_time, 07:05), (ID, R1), (categoryName, restaurant), (geom, pol3), (foodType, French), (price, cheap)]); ([(ts_date, 08/08/2008), (ts_time, 10:00), (tf_date, 08/08/2008), (tf_time, 13:00), (ID, E), (categoryName, EiffelTower),(geom, pol6)]); ([(ts_date, 08/08/2008), (ts_time, 17:10), (tf_date, 08/08/2008), (tf_time, 18:17), (ID, R1), (categoryName, restaurant), (geom, pol3), (foodType,French),(price,cheap)]);
O_2	([(ts_date, 03/08/2008), (ts_time, 11:00), (tf_date, 03/08/2008), (tf_time, 11:15), (ID,Z), (categoryName, Zoo), (geom,pol7), (price, cheap)]); ([(ts_date, 08/08/2008), (ts_time, 18:30), (tf_date, 08/08/2008), (tf_time, 21:00), (ID, Z), (categoryName, Zoo), (geom, pol7), (price, cheap)]); ([(ts_date, 19/08/2008), (ts_time, 09:00), (tf_date, 19/08/2008), (tf_time, 10:20), (ID, R1), (categoryName, restaurant), (geom, pol3), (foodType, French), (price, cheap)]); ([(ts_date, 19/08/2008), (ts_time, 17:00), (tf_date, 19/08/2008), (tf_time, 18:12), (ID, R2), (categoryName, restaurant), (geom, pol4), (foodType, French), (price,expensive)]);

Cuadro 6.3: Trayectorias

Ejemplo 3

Q3: Trayectorias que visitaron el hotel H1 y después algún lugar económico, el 10/10/2007.

```
[ID='H1'].[ts_date='10/10/2007' ∧ price='cheap']
```

Q3 contiene una subexpresión con un atributo temporal.

Ejemplo 4

Q4: Trayectorias que comenzaron en un lugar categorizado con precio (por ejemplo, un lugar que tiene el precio como un atributo), luego se detuvo en la torre Eiffel ó el zoológico y finalmente se detuvo en un restaurante que sirve comida francesa y con el mismo precio que el lugar donde comenzó la trayectoria (@x).

```
[price=@x].( [ID='Z'] | [ID='E'] ).[foodType='French' ∧
                                    price=@x]
```

Ejemplo 5

Q5: Trayectorias que visitaron un lugar categorizado como económico durante el tercer trimestre.

```
[rollup(ts_date, 'quarter', 'Time')='Q3' ∧ price='cheap']
```

RE-SPaM permite aplicar funciones a los atributos. Por ejemplo la función `rollup(attribute, 'level_i', 'dimension_k') = 'c'` y `rollup(attribute, 'level_i', 'dimension_k') = @x` es una función del estilo OLAP sobre una jerarquía. La función mapea un elemento en el nivel inferior de la dimensión *dimensión_k*, a un elemento c en el nivel *level_i* de dicha dimensión.

6.1.5. Funciones en RE-SPaM

El lenguaje de RE-SPaM soporta funciones aplicado a atributos de un stop. La sintaxis de una función de n argumentos es:

$$fn(atributo, 'ct_1', 'ct_2', \ldots, 'ct_{n-1}'), n \geq 1$$

Donde fn es el identificador de la función, *atributo* es una referencia a un atributo de un stop y $ct_i, 1 \leq i \leq n$ son constantes. Las funciones devuelven un valor que puede ser comparado con una constante ó con una variable[1]. Ejemplo:

```
rollup(ts_date, 'quarter', 'Time')='Q3'
```

En el Apéndice E se encuentra detallada la sintaxis del lenguaje RE-SPaM.

6.2. Descripción del problema

El enfoque tradicional del análisis de trayectorias ignora el entorno geográfico en el cual aquellas se desarrollan. En esta parte de la tesis resolveremos el problema de integrar el análisis de patrones (expresado por RE-SPaM) a aquel entorno de modo de poder incluir restricciones sobre la geometría de las expresiones.

Técnicamente RE-SPaM utiliza un modelo de datos que admite elementos geométricos. Estos elementos geométricos no pueden ser consultados en la actual implementación mediante funciones geométricas, por ejemplo del tipo $containsBy(g1, g2)$, donde devuelve verdadero si $g1$ contiene a $g2$. Como consecuencia de esto, una consulta del estilo "Trayectorias que visitaron algún lugar de interés que está contenido en alguna región que contiene un río y finalizaron visitando un zoológico ó una atracción turística." sería imposible de expresar con la sintaxis actual de RE-SPaM.

Se propone en esta parte de la tesis una extensión del lenguaje RE-SPaM que permita, no sólo el soporte de funciones geométricas, sino integrarla en la plataforma de Piet y, reutilizar su modelo de datos (ver la Figura 2.2) mediante el lenguaje de consulta PietQL. Integrando ambos lenguajes se aporta a RE-SPaM la posibilidad de filtrar mediante funciones geometrícas y mediante funciones y atributos OLAP.

Antes de abocarnos a este problema, debemos describir las características de PietQL.

6.3. Sintaxis y semántica de PietQL

Como se menciona en el Capítulo 2, PietQL es un lenguaje de consulta que utiliza el modelo de datos Piet el cual integra información en un data warehouse

[1] Para más información acerca de las funciones, constantes y variables ver [GV09].

con información en GIS. En un GIS, la información típicamente se organiza en layers. Por ejemplo un layer contiene información de ríos, otro información de ciudades. PietQL fue diseñado para expresar consultas GIS típicas, OLAP típicas o combinadas. Es decir, permite obtener cubos filtrados por condiciones geométricas o información de layers filtrados por cubos.

6.3.0.1. Ejemplos

La sintaxis completa de PietQL se detalla en el Apéndice E. Se describe la semántica del lenguaje mediante ejemplos de los distintos tipos de consultas que soporta PietQL. Se utiliza la palabra reservada `SELECT GIS` o `SELECT OLAP` para identificar consultas que devuelven un cubo o las que devuelven un resultado GIS, respectivamente.

Puras OLAP Las consultas denominadas OLAP son aquellas que se realizan sobre un cubo y retornan información relacionada al mismo. En este tipo de consultas se utiliza una sintaxis similar a Multidimensional Expressions(MDX)[GSSHCW06]. Ejemplo: *Productos ofrecidos por almacenes en provincias de Bélgica.*

```
SELECT CUBE
[Product].[All Products] ON ROWS
FROM [Sales]
WHERE [Store].[All Stores] IN(
   SELECT CUBE [Store].[All Stores].[BELGIUM].Children
   FROM [SALES] )
```

Puras GIS Las consultas denominadas GIS son aquellas que se realizan sobre tablas que contienen objetos geográficos y retornan información que puede a su vez contener estos objetos. Las consultas GIS se basan en la sintaxis provista por PostGIS[pos]. Ejemplo: *Distritos de Bélgica atravesados por al menos un río.*

```
SELECT GIS bel_dist
FROM bel_dist, bel_river
WHERE crosses(bel_river,bel_dist)
```

OLAP-GIS Las consultas denominadas OLAP-GIS permiten combinar la ambos tipos de consultas. La limitación es que no se pueden retornar simultáneamente objetos geográficos y del cubo a consultar en la proyección. Por lo tanto este tipo retorna información relacionada al cubo. Ejemplo: *Unidades vendidas, costo y ventas de los productos y promociones realizadas por almacenes en distritos atravesados por ríos en el año 2007.*

```
SELECT CUBE
[Measures].[Unit Sales],
[Measures].[Store Cost],
[Measures].[Store Sales] ON COLUMNS,
([Promotion Media].[All Media],
Product.[All_Products]) ON ROWS

FROM [Sales]

WHERE [Store].[All Stores] IN(
 SELECT GIS bel_dist
 FROM bel_dist,bel_river
 WHERE intersects(bel_river,layer.bel_dist))

 slice [Time].[2007]
```

GIS-OLAP Permiten recuperar información almacenada en layers de GIS, filtradas por información contenida en CUBOS OLAP. Ejemplo: *Nombre de las ciudades con ventas en provincias atravesadas por el río Dijle, tal que las ciudades hayan tenido ventas superiores a 5000 unidades.*

```
SELECT GIS lc1.name
FROM
 bel_city lc1,
 bel_prov lp2,
 bel_river lr2
WHERE
 contains(lp2, lc1) AND
 intersects(lp2, lr2)
 AND lr2.name='Dijle'
 AND lc1 IN(
  SELECT CUBE
  filter([Store].[Store City].Members,
  [Measures].[Unit Sales]>5000)
  FROM [Sales])
 AND lp2 IN(
  SELECT CUBE
  filter([Store].[Store City].Members,[Measures].
  [Unit Sales]>5000)
  FROM [Sales]) )
```

En esta consulta primero tomamos las ciudades y las provincias en donde las ventas superan las 5000 unidades. Luego filtramos las provincias atravaseadas por el río *Dijle* (intersects(lp2, lr2) AND lr2.name='Dijle') y finalmente se filtran las ciudades contenidas en esas provincias (contains(lp2, lc1)).

Para el lector interesado en [GVZ08] se encuentra en detalle la sintaxis y semántica de PietQL.

Capítulo 7

Extendiendo de *RE-SPaM*: RE-SPaM++

Como se expresó en el Cápitulo 6, RE-SPaM no permite explotar la potencia del sistema GIS-OLAP de la plataforma Piet. Para ello debemos extenderlo a efectos de permitir integrar PietQL en RE-SPaM, logrando el objetivo de vincular en un único marco, GIS, OLAP y objetos móviles. La extensión de RE-SPaM se denomina RE-SPaM++. Se describe a continuación esta integración, la sintaxis de RE-SPaM++ y funcionamiento.

7.1. Integración con PietQL

La solución propuesta para integrar RE-SPaM y PietQL es vincular el resultado de una consulta PietQL (en el caso que devuelve elementos espaciales) con la geometría de los PoIs, a través de funciones. Por ejemplo, una función del estilo *containedBy* parece ser suficiente para consultar si la geometría de un stop en una trayectoria está contenida en geometrías restringidas en una consulta definida PietQL. Sin embargo, las funciones en RE-SPaM, tal como se especificaron en la Sección 6.1.5, precisan como primer argumento un atributo del stop de una trayectoria y luego parámetros adicionales que deben ser constantes. El problema es que las consultas PietQL no devuelven una constante, sino un cursor para navegar el resultado. Por tal motivo se necesita definir un *un nuevo tipo de función* para poder realizar la integración de RE-SPaM con PietQL. El nuevo tipo de función tiene la siguiente sintaxis:

$$fn(atributo, nombreTabla.nombreAtributo)$$

Donde *nombreTabla.nombreAtributo* es el atributo identificado como *nombreAtributo* de la tabla *nombreTabla*, la cual es una *tabla temporal* con alcance local a una consulta RE-SPaM++. Una tabla temporal en RE-SPaM define un alias a una consulta PietQL, permitiendo de esta manera integrar PietQL y RE-SPaM. La tabla temporal se define mediante la cláusula `WITH TABLE`. A continuación describiremos la sintaxis y la semántica de la extensión a RE-SPaM que permite la integración entre información geográfica y OLAP.

7.2. Sintaxis y Semántica

La sintaxis de RE-SPaM++ se divide en dos partes:

Parte PietQL : La parte PietQL (opcional) contiene definiciones de alias a consultas PietQL. Las consultas PietQL definidas se reestringen a aquellas que retornan información geográfica.

Parte RE-SPaM : En ella se definen las reestricciones mediante el lenguaje RE-SPaM y, opcionalmente, incluyendo reestricciones que hacen referencias a las tablas temporales definidas en la sección PietQL.

Para definir tablas temporales en RE-SPaM++ se utiliza la cláusula `WITH TABLE` seguida del identificador de la tabla temporal, una serie de identificadores de atributos que ligan las columnas del resultado de la ejecución de la consulta PietQL a nombres simbólicos y, la definición de una consulta PietQL. Por ejemplo, en la siguiente consulta RE-SPaM++ se definen dos tablas temporales que luego son referenciadas en la parte RE-SPaM:

```
WITH TABLE T1 (at1) as
SELECT GIS [...];

WITH TABLE T2 (at2) as
SELECT GIS [...];

[containedBy(geom1, T1.at1)='true'
 and price='cheap']?.[in(geom2,T2.at2)='true']
```

Las consultas PietQL no son ejecutadas en RE-SPaM++ sino son referenciadas en la parte RE-SPaM. En el Apéndice E se encuentra la sintaxis completa de RE-SPaM++.

7.3. RE-SPaM++ en ejemplos

En esta sección se describirá el lenguaje mediante ejemplos.

7.3.1. Ejemplos

Las trayectorias utilizadas en los siguientes ejemplos representan travesías de turistas en Bélgica.

Ejemplo 1

Q1: Trayectorias que visitaron algún lugar de interés que está contenido en alguna region que contiene un río y finalizaron visitando un aeropuerto ó alguna atracción turística.

```
WITH TABLE regRiver(the_geom) AS
SELECT GIS DISTINCT(bel_regn.the_geom)
FROM bel_regn,bel_river
WHERE contains(bel_regn.the_geom,bel_river.the_geom);

[containedBy(geom, regRiver.the_geom)='true'].
([categoryName='Airport']|[categoryName='Tourist Attraction'])
```

La tabla **bel_regn** contiene las regiones[1] de bélgica y **bel_river** contiene los ríos (polílineas). Con la cláusula **WITH TABLE regRiver(the_geom)** se define una tabla temporaria con alcance local. Así, la invocación de la función **containedBy(geom, regRiver.the_geom)** navega el cursor asociado a dicha tabla.

Ejemplo 2

Q2: Trayectorias que visitaron algún lugar de interés económico, cercano a un distrito que pertenece a una región atravesada por algún río, y finalizaron visitando alguna atracción turística.

```
WITH TABLE district(the_geom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist,bel_regn, bel_river
WHERE intersects(bel_regn.the_geom,bel_river.the_geom) and
contains(bel_regn.the_geom,bel_dist.the_geom);
```

```
[price='cheap' ∧ smallDistance(geom,district.the_geom)='true'].
[categoryName='tourist attraction']
```

La tabla **bel_dist** contiene los distritos de Bélgica.

Ejemplo 3

Q3: Trayectorias que visitaron algún lugar de interés económico cercano a un "distrito" el cual pertenece a una ciudad que contiene algún almacén donde se vendió más de 100 unidades en el año 2007, y finalizaron visitando alguna atracción turística.

[1] Aquí las regiones se representan por polígonos (si se utiliza el sistema vectorial para representar los cuerpos, como es el caso).

```
WITH TABLE district(thegeom) AS
SELECT GIS DISTINCT(bel_dist.the_geom)
FROM bel_dist,bel_city
WHERE contains(bel_dist,bel_city) AND
bel_city IN(
SELECT CUBE filter([Store].[Store District].
[Nijvel].Children,[Measures].[Unit Sales]>100)
FROM [Sales]
slice [Time].[2007]);
```

```
[price='cheap' ∧ smallDistance(geom,district.the_geom)='true'].
[categoryName='tourist attraction']
```

En la parte PietQL estamos tomando las ciudades con más de 100 unidades venidadas en el año 2007 (`bel_city IN(...)`). Luego tomamos los distritos que contienen esas ciudades (`contains(bel_dist,bel_city)`). En la parte RE-SPaM restringimos PoIs económicos y cercamos a los distritos definidos en la parte PietQL (`[price='cheap' ∧ smallDistance(geom,district.the_geom)='true']`) y terminando en una atracción turística (`[categoryName='tourist attraction']`).

Capítulo 8

RE-SPaM++: Implementación

En este capítulo se describe la implementación RE-SPaM++, los objetivos que se persiguieron en el diseño y algunos ejemplos.

8.1. Objetivos y requisitos de la solución

En esta sección se detallan los objetivos y requisitos que se persiguieron en el diseño.

Los requisitos de la solución fueron:

1. *No intrusiva:* La extensión en primer lugar no deberá afectar la librería que implementa PietQL y RE-SPaM. No debería ser modificada para acoplarse a la solución, sino que la misma se debe adaptar.

2. *Las gramáticas deberían ser reutilizables, si es posible:* Cada lenguaje trae un parser que deberá mantenerse.

3. *Customizable:* Es decir, flexible en cuanto a su configuración.

4. *Requerir mínimo esfuerzo de configuración:* Desde RE-SPaM la configuración debería ser simple.

5. *Permitir puntos de extensión:* En caso de ser necesario dejar la posibilidad de poder extender la solución de forma simple.

6. *Transparente para PietQL:* Debería desconocer que se está utilizando desde la solución.

8.2. Introducción

La implementación de esta parte de la tesis consiste una librería diseñada desde dos puntos de vista. El primero se basa en crear y ejecutar consultas RE-SPaM++ y, el segundo, en integrar dos lenguajes de consultas ya implementados en librerías separadas e independientes. Cada punto de vista tiene un propósito:

1. Desde el punto de vista del uso, la implementación debe simplificar la creación y ejecución de consultas, ocultando el hecho que finalmente las consultas se ejecutarán en las librerías que implementen PietQL y RE-SPaM.

2. Desde el punto de vista de la integración, la implementación debe minimizar lo más posible el esfuerzo necesario para lograr la adaptación entre ambas librerías.

Este capítulo se divide en dos partes, cada una describiendo la implementación desde los diferentes puntos de vista. La Sección 8.3 se basa en el primer punto de vista y la sección 8.4 en el segundo.

8.3. Uso y configuración de RE-SPaM++

En esta sección se describe el API de RE-SPaM++ para la creación y ejecución de consultas. Se muestran algunos ejemplos y luego se describen las interfases más importantes del API.

8.3.1. Ejemplos

8.3.1.1. Ejemplo 1

En este ejemplo se muestra como crear y ejecutar una consulta RE-SPaM++.

```
1 Session s = ClaseDeMiAplicacion.obtenerSession();
2 ERSPaMQuery qq = s.createERSPaMQuery(''...'');
3 Object result = qq.execute();
4 s.close();
```

Explicación

Línea 1 Se obtiene la sesión de trabajo. La sesión es la unidad de trabajo de RE-SPaM++.

Línea 2 Se construye una consulta en el lenguaje extendido RE-SPaM.

Línea 3 Se ejecuta la consulta y se almacena en `result` el resultado.

Línea 4 Se cierra la sesión.

8.3.1.2. Ejemplo 2

En este ejemplo se muestra como crear y ejecutar una consulta PietQL embebida dentro de una consulta RE-SPaM++.

```
1 Session s = ClaseDeMiAplicacion.obtenerSession();
2 ERSPaMQuery qq = s.createERSPaMQuery(''
3
4 WITH TABLE t1 (at1) AS
5   SELECT GIS bel_city.name
6   FROM bel_city
```

```
7    WHERE bel_city IN(
8     SELECT GIS bel_city
9     FROM bel_city , bel_river , bel_prov
10    WHERE intersects(bel_prov , bel_river) AND
11     contains(bel_prov , bel_city)
12     );
13
14 [price='cheap'].([ID='Eiffel'] | [ID='Zoo']).[price='cheap']
15 '');
16 Object result = qq.getTable(''t1'').runAssociatedQuery();
17 s.close();
```

Explicación

Línea 1 Se obtiene la sesión de trabajo.

Línea 2 Se construye una consulta en el lenguaje extendido RE-SPaM.

Línea 16 Se ejecuta la consulta PietQL definida en la cláusula `WITH TABLE` nombrada como `t1` y se almacena en `result` el resultado de su ejecución.

Línea 17 Se cierra la sesión.

8.3.1.3. Ejemplo 3

En este ejemplo se muestra como crear y ejecutar una consulta PietQL. La librería permite ejecutar consultas PietQL directamente.

```
1 Session s = ClaseDeMiAplicacion.obtenerSession();
2 ResultSet r = (ResultSet)s.createPietQLQuery(''SELECT GIS ...'')
3                                .execute();
4 s.close();
```

Explicación

Línea 1 Se obtiene la sesión de trabajo.

Línea 2 Se construye, ejecuta y almacena una consulta PietQL GIS[1]. Las consultas GIS retornar objetos que implementan la interfaz `java.sql.ResultSet`[2]

Línea 4 Se cierra la sesión.

[1]En la implementación actual no se soporta consultas que retornen objetos tipo OLAP, pero su soporte es simple de incorporar como se verá mas adelante

[2]En la implementación actual de la librería que implementa la ejecución de consultas GIS PietQL retorna `java.sql.ResultSet` por eso es seguro el casteo. El método `execute()` retorna objetos tipo `java.lang.Object`, debido a que el tipo de dato que retorna la librería está por fuera de la implementación y podría cambiar.

8.3.1.4. Ejemplo 4

En este ejemplo se muestra como crear las sesiones.

```
1 SessionFactory sf = ConfigurationFactory.getConfiguration()
2                        .configure().buildSessionFactory();
3 Session s = sf.openSession();
4 s.close();
5 sf.close();
```

Explicación

Línea 1 Se crea una instancia de `SessionFactory`. En secciones posteriores se describe en detalle como funciona la configuracuón.

Línea 3 Se crea una nueva sesión. No es costosa esta operación.

Línea 4 Se cierra la sesión.

Línea 5 Se cierra la instancia y se liberan los recursos asociados a `sf`.

8.3.2. Sesiones

En RE-SPaM++, las sesiones permiten crear consultas RE-SPaM++ como también PietQL. Las sesiones son creadas a partir de una instancia `SessionFactory` y abstraen al programador de la manipulación de las conexiones JDBC necesarias para ejecutar las consultas. También abstrae atrapando posibles errores ocurridos en las librerías PietQL y RE-SPaM al encapsularlos en excepciones definidas en el API de RE-SPaM++.

Una sesión tiene un ciclo de vida determinado explícitamente mediante la creación desde el SessionFactory y luego invocando el método `close` sobre la misma. El método `close` tiene como objetivo liberar las conexiones y recursos utilizados. Una consulta creada puede ejecutarse mientras la sesión no se cierre.

8.3.3. Interfases Query

Las intefases Query representan las consultas. Las más importantes son:

- `PietQLQuery`: Esta interfaz representa una consulta PietQL.
- `RESPaMQuery`: Esta interfaz representa la parte RE-SPaM de una consulta RE-SPaM++ (ver la Sección 7.2).
- `ERESPaMQuery`: Esta interfaz representa una consulta RE-SPaM++.

Estos tres tipos de consultas, a excepción de `RESPaMQuery`, una vez creadas pueden ser ejecutadas mediante el método `execute`. La interfaz más importante es `ERESPaMQuery`, la que permite ejecutar consultas RE-SPaM++.

8.3.4. Creación de sesiones

Las sesiones son creadas mediante el método `openSession` de una instancia de *SessionFactory*. Este método crea y configura la sesión con la información necesaria para crear las consultas[3]. Una instancia de SessionFactory se crea una única vez en la aplicación que utiliza RE-SPaM++, ya que la misma es una operación costosa a comparación a la de una sesión.

Las instancias de `SesionFactory` al igual que las sesiones tienen un ciclo de vida asociado. Comienza al momento de creación y finaliza cuando se invoca al método `close` sobre la misma. Este método libera los recursos utilizados por la instancia. Una vez que se invoca `close`, no se pueden crear sesiones y la instancia debe ser desechada.

8.3.5. Creación de instancias SessionFactory

Las instancias de `SessionFactory` se crea de la siguiente forma:

```
ConfigurationFactory.getConfiguration()
    .configure().buildSessionFactory()
```

Mediante esta instrucción se leen archivos de configuración que inicializan las librerías que implementan PietQL y RE-SPaM, y si no hubo ningún error en ello, crea una instancia de `SessionFactory` con información necesaria para crear las sesiones. Los objetivos de este método modo de configuración son:

1. Garantizar que cuando se creen sesiones a partir de las instancias `SessionFactory`, las librerías que implementan PietQL y RE-SPaM están disponibles y configuradas

2. Disponer de un modo de configuración común.

8.3.6. Resumen

Debe notarse la similitud de nuestra solución con el framework Hibernate[hib]. La decisión de seguir esta línea se debió prinicipalmente a:

1. La rápida comprensión para los programadores que hayan usado el API de Hibernate, la cual es muy utilizada en la comunidad Java.

2. Es un modelo que permite extenderse fácilmente en diferentes áreas, como por ejemplo caching, optimización de consultas, etc.

En el apéndice D se profundiza en los detalles de la implementación.

8.4. Integración de RE-SPaM y PietQL

En esta sección se describe los aspectos de la implementación desde el punto de vista de la integración.

[3]También para la ejecución, ya que en la implementación actual, las sesiones son las responsables de la ejecución de las consultas

8.4.1. Visión general

En la Figura 8.1 se describen los componentes de la implementación de RE-SPaM++ (empaquetados en `librespam++`) y la interacción entre las librerías que implementan PietQL (`libpql`) y RE-SPaM (`librespam`), como así también que componente es visible para quien utilize RE-SPaM++.

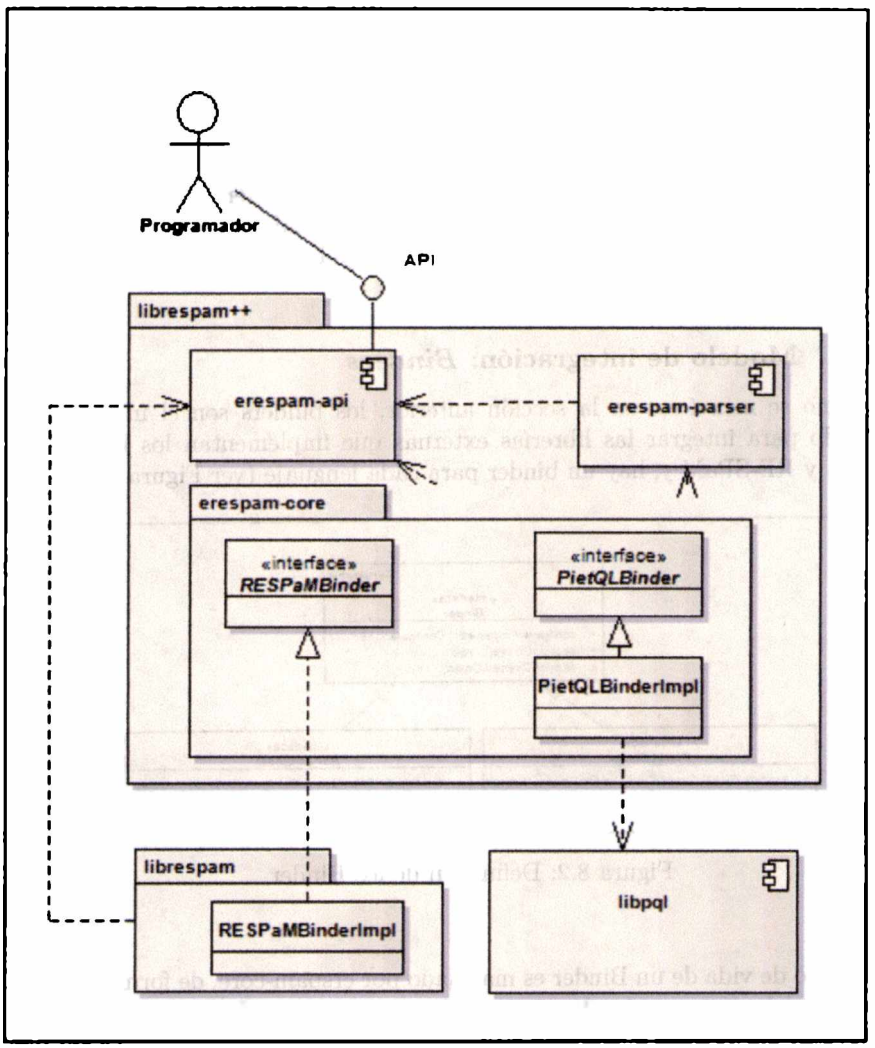


Figura 8.1: Diagrama de componentes de la implementación de Re-SPaM++

En el diagrama se muestra que la implementación de librespam++ se divide en tres componentes:

1. erespam-api: Contiene las interfases Query, Session, SesionFactory, etc.

2. erespam-parser: Implementa el parser del lenguaje RE-SPaM++.

3. erspam-core: Implementa las interfases definidas en erspam-api y orquesta la ejecución de las consultas delegando a los *binders* cuando es necesario.

El componente erspam-api provee el API de RE-SPaM++ para quien utilize la librería, mientras que erspam-core define un modelo de integración mediante lo que se denomina *binder*. Un *binder* es el mecanismo (una clase) por el cual desde librespam++ interactúa con las librerías que implementan PietQL y RE-SPaM. En la Figura 8.1 se aprecia que el binder para PietQL se encuentra en espam-core y el binder de RE-SPaM se encuentra implementado dentro de la misma librería. Implementando el binder de PietQL dentro de erspam-core logramos cumplir que la solución sea *Transparente para PietQL* como se definió en la Sección 8.1. El binder de RE-SPaM no se implementó en erspam-core porque RE-SPaM++ afecta funcionalmente a RE-SPaM y uno de los objetivos de librespam++ es ocultar la interacción con libpql. Por tal motivo se deja la implementación fuera de erspam-core.

8.4.2. Modelo de integración: *Binders*

Como se menciona en la sección anterior, los binders son el mecanismo utilizado para integrar las librerías externas que implementan los lenguajes PietQL y RE-SPaM y, hay un binder para cada lenguaje (ver Figura 8.2).

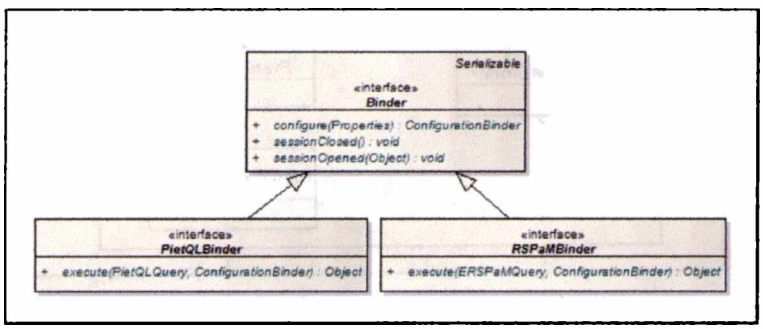


Figura 8.2: Definición de los Binder

El ciclo de vida de un Binder es manejado por erspam-core, de forma que lo único que debe hacer un implementador de un binder es indicar a erspam-core cuál es el binder de cada lenguaje. El binder se utiliza para:

1. Manejar eventos.

2. Ejecutar consultas.

Los eventos son dos, cuando se está creando una instancia de SessionFactory y cuando se está creando una instancia de una sesión. Para mayor detalle ver la Sección 8.4.4. A continuación se describe cómo se le indica a erspam-core cuál es el binder, el manejo de eventos y por último se describe que información recibe el binder al ejecutar las consultas.

8.4.3. Configuración de los binder

Para indicar a erspam-core cuales son los binder, se utiliza el archivo erspam-binders.properties.

```
erspam-binders.properties
-------------------------
ar.edu.unlp.erespam.binders.PietQLBinder=...
ar.edu.unlp.erespam.binders.RSPaMBinder=...
```

En cada línea se especifica la clase que implementa la interfase correspondiente.

8.4.4. Eventos

Los eventos que deben manejar los binder son dos, al momento que se está creando una instancia de SessionFactory como se describe en 8.3.5 y cuando se crea una sesión mediante el método `openSession` de SessionFactory. A continuación se describen estos eventos.

8.4.4.1. Creación de SessionFactory

Cuando se crea una instancia de SessionFactory se invoca el método `ConfigurationBinder configure(Properties archivoDePropiedades)` a cada binder. Este método permite a los binder saber que se está creando un SessionFactory y que, posiblemente, a continuación se ejecutarán consultas. El método devuelve un objeto que luego es pasado al binder cada vez que se ejecuta una consulta, como se describe en la Sección 8.4.5.

La idea de proveer este evento es que los binder realicen la configuración en el método `configure` y no al ejecutar las consultas. Si solamente se provee el método para ejecutar, cada implementación de los binder debe chequear si la configuración necesaria para la ejecución se realizó o no. El uso del método `configure` garantiza que cuando se está ejecutando las consultas, cada librería ya fue configurada.

El método `configure` recibe por parámetro un archivo del tipo clave/valor. Cada binder recibe un archivo diferente, definidos en erspam.properties.

```
pietql-properties=/piet-ql.properties
rspam-properties=/rspam-config.properties
```

Este evento es opcional. Cada implementación puede no realizar ninguna acción en el método configure y manejar internamente como se configura (en que momento).

8.4.4.2. Creación de sesiones

Cuando se crea una sesión, se crea una instancia del binder para PietQL y para RE-SPaM y se asocian a la misma. Por este motivo, el ciclo de vida de un binder es el mismo que el de una sesión. Al crear una sesión satisfactoriamente se invoca al método `sessionOpened(object)`. El parámetro es pasado al momento de creación de la sesión.

```
SessionFactory sf = ...;
//creamos una sesion
Object data =...;
Session s1 = sf.openSession(data);
//es lo mismo que sf.openSession(null);
Session s = sf.openSession();
```

8.4.5. Ejecución de consultas

Cuando una consulta (ERSPaMQuery o PietQLQuery) se ejecuta, internamente se invoca al método
`execute(ERSPaMQuery q, ConfigurationBinder c)`[4] para que finalmente ejecute la consulta ó eleve un error si falla. El parámetro `configuration` es el objeto creado en el método `configure` del binder (ver la Sección 8.4.4.1).

8.5. Resumen

En este capítulo se ha presentado el API de RE-SPaM++ y el modelo de integración. Tanto el API como el modelo de integración consta de muy pocas clases y métodos, y de esta manera cumplimos los requisitos, planteados al inicio de este capítulo, de requerir mínimo esfuerzo de configuración y permitir puntos de extensión (vía los binder).

[4]Si se ejecuta una consulta ERSPaMQuery.

Capítulo 9

Conclusión y temas abiertos

9.1. Conclusiones

La herramienta implementada (Matcher) para la creación de las estructuras auxiliares que integra datos espaciales y OLAP, facilita al administrador de Piet dicha labor de forma considerable. En los experimentos se demostró que realiza dicha tarea en tiempos razonables.

Del análisis de la solución implementada, se desprende que cumple con el objetivo inicial de simplificar y automatizar en lo posible la integración de GIS y OLAP, al completar la estructura auxiliar en la plataforma Piet reduciendo considerablemnte el tiempo y el costo de realizar esta labor manualmente.

La extensión al lenguaje RE-SPaM propuesta, RE-SPaM++, cumple con el objetivo inicial de aprovechar la estructura auxiliar y de esta manera permitir dentro del lenguaje el uso mediante PietQL de la información GIS/OLAP en la definición de restricciones a los patrones de trayectorias descubiertos por el algoritmo de RE-SPaM.

9.2. Trabajo a futuro

En esta sección se describen las extensiones y mejoras que se podrían realizar. A continuacin se listan las más importantes agrupadas en las que conciernen a la herramienta de integración y a RE-SPam++.

9.2.1. Mejoras y extensiones para Matcher

- Implementar una interfaz gráfica independiente de Jump.
- Implementar una librería de funciones de comparación de atributos.
- Soporte transparente a la ejecución en paralelo de la comparación entre los registros.
- Soporte de asignación de pesos a atributos de acuerdo a la frecuencia del mismo[1].
- Implementar la asignación automática de pesos a los campos por medio del método EM, el lector interesado puede profundizar en [DLR77].

[1] Ver sección 2.2.5.

9.2.2. Mejoras y extensiones para RE-SPaM++

- Unificar la creación de consultas en la plataforma Piet. Ahora se utilizan dos métodos: `session.createERSPaMQuery()` y `session.createPietQLQuery()`. Sería mejor tener un sólo método por ej. `session.createQuery()` que analizando el contenido detecte cómo se debe ejecutar.

- Implementar consultas parametrizadas al estilo JDBC, en donde los parámetros se especifican con el carácter ?. Por ejemplo:

```
ERSPaMQuery q = session.createERSPaMQuery(''
WITH TABLE t1 (at1) AS
SELECT GIS lc1.name
FROM
 bel_city lc1,
 bel_prov lp2,
 bel_river lr2
WHERE
 contains(lp2, lc1) AND
 intersects(lp2, lr2)
 AND lr2.name=? FROM [Sales]);


[price='cheap' ∧ smallDistance(geom,district.the_geom)='true'].
[categoryName=?]


q.setParameter(1,'Dijle').setParameter(2,'tourist attraction')
```

- Al igual que consultas JDBC, implementar el soporte a consultas precompiladas[2].

- Implementar métodos que permitan paginar los resultados.

- Integrar en forma completa PietQL. En esta implementación no se permiten ejecutar consultas que devuelvan información de cubos OLAP.

[2] El método `createPrepareStatement` de la clase `Connection` en el API de JDBC.

Bibliografía

[arg] *Args4j: java command line options parser*, `https://args4j.dev.java.net/`.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe, *A comparative analysis of methodologies for database schema integration*, ACM Comput. Surv. **18** (1986), no. 4, 323–364.

[cas] *Castor*, `http://www.castor.org/`.

[Chr08] Peter Christen, *A freely available record linkage system with a graphical user interface*, Proceedings of the Australasian Workshop on Health Data and Knowledge Management (HDKM). Conferences in Research and Practice in Information Technology (CRPIT), vol. 80. (Wollongong, Australia), Australasian Workshop on Health Data and Knowledge Management (HDKM), Enero 2008.

[coma] *Commons codec de apache commons*, `http://commons.apache.org/codec/`.

[comb] *Commons configuration*, `http://commons.apache.org/configuration/`.

[comc] *Commons logging*, `http://commons.apache.org/logging/`.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg, *A comparison of string distance metrics for name-matching tasks*, -, 2003, pp. 73–78.

[csv] *Common format and mime type for comma-separated values (csv) files*, `http://tools.ietf.org/html/rfc4180`.

[dba] *Archivos dbase*, `http://www.clicketyclick.dk/databases/xbase/format/index.html`.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the em algorithm*, Journal of the Royal Statistical Society. Series B (Methodological) **39** (1977), no. 1, 1–38.

[dme] *The double metaphone search algorithm*, `http://www.ddj.com/cpp/184401251`.

[EGV95] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides, *Design patterns: Elements of reusable object-oriented software*, Addison Wesley, 1995.

[EK72] Jack Edmonds and Richard M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM 19 (1972), 248–264.

[EVE02] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid, *Tailor: a record linkage toolbox*, Data Engineering, 2002. Proceedings. 18th International Conference on, 2002, pp. 17–28.

[For56] D. R. Ford, L. R.; Fulkerson, *Maximal flow through a network*, Canadian Journal of Mathematics (1956), 399–404.

[FS69] Ivan P. Fellegi and Alan B. Sunter, *A theory for record linkage*, Journal of the American Statistical Association **64** (1969), no. 328, 1183–1210.

[GBVR03] Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford, *Record linkage: Current practice and future directions*, Tech. report, CSIRO Mathematical and Information Sciences, 2003.

[GSSHCW06] Francesco Civardi George Spofford Sivakumar Harinath Chris Webb, Dylan Hai Huang, *Mdx-solutions: With microsoft sql server analysis services 2005 and hyperion essbase*, Wiley, Noviembre 2006.

[GV09] Leticia I. Gomez and Alejandro A. Vaisman, *Efficient constraint evaluation in categorical sequential pattern mining for trajectory databases*, EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology (New York, NY, USA), ACM, 2009, pp. 541–552.

[GVZ08] Leticia I. Gomez, Alejandro A. Vaisman, and Sebastián Zich, *Piet-ql: a query language for gis-olap integration*, GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (New York, NY, USA), ACM, 2008, pp. 1–10.

[hib] *Hibernate*, `https://www.hibernate.org/`.

[HS95] Mauricio A. Hernández and Salvatore Stolfo, *The merge/purge problem for large databases*, In Proceedings of the 1995 ACM SIGMOD, 1995, pp. 127–138.

[HSW07] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler, *Data quality and record linkage techniques*, Springer, July 2007.

[i18] *Internacionalización en aplicaciones java*, `http://java.sun.com/docs/books/tutorial/i18n/index.html`.

[jar] *Archivos javatm*, `http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html`.

[Jar97] M. Jaro, *Software demonstrations*, In Proc. of an International Workshop and Exposition - Record Linkage Techniques, 1997.

[jcc] *Java compiler compile*, `https://javacc.dev.java.net/`.

[jig] *Jigcell project*, `http://jigcell.biol.vt.edu/`.

[JLX+08] P Jurczyk, JJ Lu, L Xiong, JD Cragan, and A Correa, *Fril: A tool for comparative record linkage.*, AMIA Annu Symp Proc (2008).

[Jum] *Java unified mapping platform*, `http://www.jump-project.org`.

[jvm] *Java virtual machine*, `http://java.sun.com/docs/books/jvms/`.

[KCGS93] Won Kim, Injun Choi, Sunit Gala, and Mark Scheevel, *On resolving schematic heterogeneity in multidatabase systems*, Distrib. Parallel Databases **1** (1993), no. 3, 251–279.

[KM72] V. Klee and G.J. Minty, *How good is the simplex algorithm?*, vol. III, Inequalities, Academic Press, New York, 1972, 159-175.

[Kuh55] H. W. Kuhn, *The hungarian method for the assignment problem*, vol. 2, Copyright 1955 Wiley Periodicals, Inc., A Wiley Company, Bryn Mawr College, 1955, 83-97, DOI `http://dx.doi.org/10.1002/nav.3800020109`.

[LC94] Wen-Syan Li and Chris Clifton, *Semantic integration in heterogeneous databases using neural networks*, VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1994, pp. 1–12.

[LNE89] James A. Larson, Shamkant B. Navathe, and Ramez Elmasri, *A theory of attribute equivalence in databases with application to schema integration.*, IEEE Trans. Software Eng. **15** (1989), no. 4, 449–463.

[mon] *Mondrian pentaho. mondrian is an olap server written in java.*, `http://mondrian.pentaho.org/`.

[Mon81] G. Monge, *Mémoire sur la théorie des déblais et de remblais. histoire de l'académie royale des sciences de paris, avec les mémoires de mathématique et de physique pour la même année*, 1781, 666-704.

[Mun57] James Munkres, *Algorithms for the assignment and transportation problems*, vol. 2, J. Soc. Indust. and Appl. Math. 5, 32, 1957, DOI: 10.1137/0105003.

[mvn] *Maven*, `http://maven.apache.org/`.

[oCPC] DC's Division of Cancer Prevention and Control, *Link plus*, http://www.cdc.gov/cancer/npcr/tools/registryplus/lp.htm.

[ost] *Open source (gpl) java utilities maintained by stephen ostermiller with help from many contributors.*, `http://ostermiller.org/utils/`.

[pie] *Piet is an open source system that integrates geographic information systems (gis) and olap systems*, `http://piet.exp.dc.uba.ar/pietql/`.

[ple] *The plexus container*, `http://plexus.codehaus.org/`.

[pos] *Postgis*, `http://postgis.refractions.net/`.

[py] *Python*, `http://www.python.org/`.

[sec] *Secondstring project*, `http://secondstring.sourceforge.net/`.

[SPD+07] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. Fernandes de Macedo, F. Porto, and C. A. Vangenot, *Technical report*, Conceptual view of trajectories, 2007.

[ujm] *Universal java matrix package*, `http://www.ujmp.org/`.

[Win89] William E. Winkler, *Frequency-based matching in the fellegi-sunter model of record linkage*, Proceedings of the Section on Survey Research Methods, American Statistical Association, 1989, pp. 778–783.

[Zea06] Statistics New Zealand, *Data integration manual*, Statistics New Zealand, www.stats.govt.nz, Agosto 2006.

Apéndice A

Modelo de decisión Fellegi y Sunter

A continuación se presenta el modelo de decisión[1] planteado en [FS69].

A.1. Introducción

En este modelo, comparar dos registos implica comparar los atributos de los registros por separado asignando un valor a cada par que refleje su similitud. Este valor se denomina *peso del campo* y se calcula a partir del grado de fiabilidad de los datos (probabilidad m) y la frecuencia del valor (probabilidad u).

A.1.1. Probabilidad m

La probabilidad m es el grado de fiabilidad de los datos y se define como:

$$m = Pr(\textit{los dos campos coinciden} \mid \textit{los registros son un match})$$

La probabilidad m se puede expresar como: dado dos registros r_1 y r_2 que son un match, cuál es la probabilida que debido a errores en los campos comparados, los registros r_1 y r_2 no sean clasificados como un match. Es por este motivo que la probabilidad m está esta asociada con la calidad de los datos. Esto es:

$$m = 1 - Pr\ (\textit{los campos no coinciden} \mid \textit{los registros son un match})$$

Si por ejemplo, se comparan registros que contienen un campo género, un valor de probabilidad m de 0,98 podría ser adecuado ya que es un dato simple y podría considerarse fiable. En cambio un campo que contiene la dirección no es tan fiable y asignar una probabilidad m de 0,7 sería más adecuado.

[1] Como clasificar si un par de registros es o no un match.

A.1.2. Probabilidad u

La probabilidad u es el grado de frecuencia de los datos. Esta probabilidad se expresa como *la probabilidad de que los valores coincidan pero no sean la misma entidad.* Es decir:

$$u = Pr(los\ dos\ valores\ coinciden \mid los\ registros\ no\ son\ un\ match)$$

Podemos decir que cuanto más frecuente es el valor más probable que dos registros no relacionados contengan ese valor. Por lo tanto, la probabilidad u es comunmente definida como:

$$u = 1/(Número\ de\ valores\ diferentes)$$

Por ejemplo, el género puede ser tanto masculino como femenino con igual probabilidad, de donde se deduce que la probabilidad u es 0,5. Hay 12 meses en un año, entonces un campo mes podría tener una probabilidad u de $\frac{1}{12}$. La dirección en general no se repite, por lo que un posible valor para la probabilidad u sería 0,01 o menor. Cuando cada valor tiene la misma probabilidad de ocurrencia se conoce como *probabilidad u global.*

A.2. Cálculo del peso del campo

El peso del campo es el resultado de la comparación entre dos atributos y refleja su coincidencia. Si hay coincidencia[2] el peso tiene un valor positivo, si no, el peso es negativo. El peso se calcula de la siguiente manera:

$$coincidencia\ de\ los\ valores = \log_2\left(\frac{m}{u}\right)$$

$$no\ coincidencia\ de\ los\ valores = \log_2\left(\frac{1-m}{1-u}\right)$$

La aplicación de logaritmos se utilizan para simplificar cálculos posteriores ya que los pesos de campo[3] son aditivos.

A.3. Cálculo del peso asociado al par de registros

Una vez calculados los pesos de los campos, se calcula un peso para el par de regitros y se calcula sumando los pesos de cada uno de los campos.

Ejemplo: Sean A y B dos archivos con registros de personas y suponer los siguientes registros:

Archivo	**A**
Nombre	John Black
Fecha de nacimiento	23/11/63
Género	M
Direccón	112 Hiropi Street

[2]Para determinar la coincidencia se utilizan funciones de comparación.

[3]Asumiendo independencia de atributos, es decir que un valor de un atributo no depende del valor de otro atributo. La independencia entre atributos es un requerimiento del modelo propuesto por Fellegi-Sunter.

Archivo	B
Nombre	Jon Block
Fecha de nacimiento	23/11/65
Género	M
Direccón	89 Molesworth St

Para calcular el peso del par de registros, primero se asignan las probabilidad m y u, luego se calcula los pesos de los campos:

Archivo	prob. m	prob. u	valor coinc.	v. no coinc.
Nombre	0,95	0,01	6.57	-4,31
Fecha de nacimiento	0,9	0,01	6,49	-3,31
Género	0,95	0,5	0,93	-3,32
Dirección	0,7	0,01	6,13	-1,72

Luego, se comparan los registros

Archivo	¿Hay coincidencia?	Peso del campo
Nombre	No	-4,31
Fecha de nacimiento	No	-3,31
Género	Sí	0,93
Dirección	No	-1,72

Finalmente el peso compuesto asignado al par de registros es:

$$(-4{,}31) + (-3{,}31) + (0{,}93) + (-1{,}72) = -8{,}41$$

Como el valor es negativo, en el proceso de integración este par es descartado.

En la práctica, el cálculo del peso de los campos puede ser parcial. Es decir, el valor del peso del campo cuando hay coincidencia es número real que mientras mayor sea ese valor, mayor es la similitud. En el ejemplo anterior, cualquier par de registro que coincidan solamente en el género tiene un peso asociado de -8.41, mientras que con cálculos de pesos parciales, si el género en un caso son idénticos, el peso asociado al par es -8.41. Pero si no son idénticos y aún se consideran coincidencia, podría ser de -8.5.

A.4. Clasificación

Los pares se clasifican aplicando un umbral de corte en los pares analizados (Figura 2.6). El umbral se define con dos límites, el límite superior (upper cut-off) y el límite inferior (lower cutt-off). El límite superior indica el peso que un par de registros debe superar para ser considerado un link y el límite inferior (lower cutt-off) indica que si el peso de un par de registros es menor, se considerará como no link. Si los límites coinciden, divide al conjunto de pares de registros en dos. Sin embargo si no coinciden se dividen en tres conjuntos y el conjunto de pares definido entre los dos límites, se denomina *clerical review*. Los pares clasificados en este nuevo conjunto se aconsejan revisar manualmente. En la Figura A.1 se muestra un ejemplo de un umbral de corte.

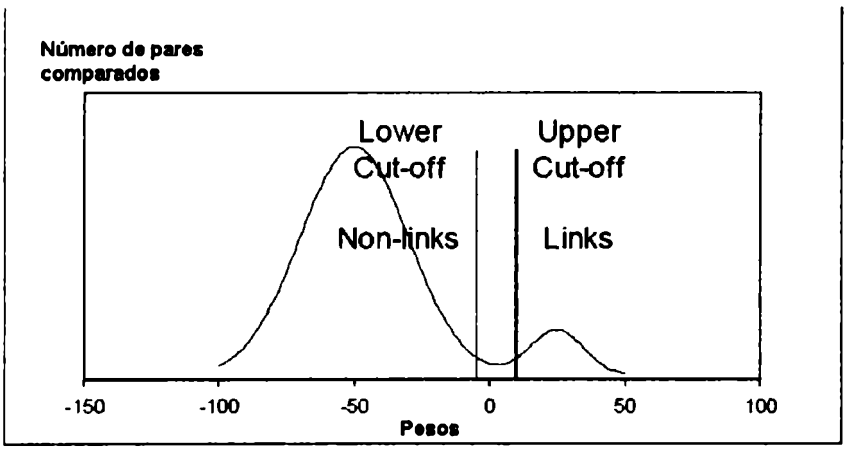


Figura A.1: Distribución de los pesos compuestos y los límites del umbral.

En [FS69] se describe como obtener los mejores límites de umbral, pero en la práctica es la persona que trabaje en el proceso de integración de datos quien decida estos límites [Zea06, p. 44].

A.5. Relación entre el peso asociado a un campo y las probabilidades m y u

Para comprender como afecta a los pesos del campo las variaciones de las probabilidades m y u se presentan las Figuras A.2 y A.3

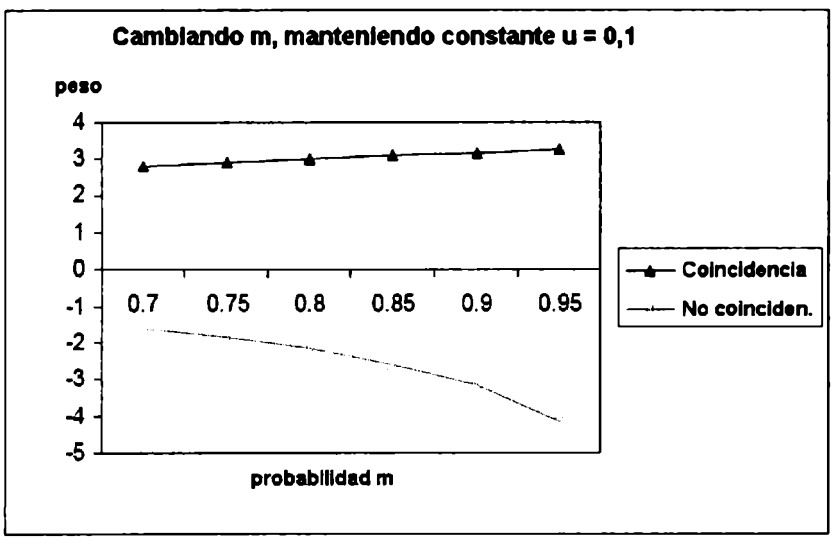


Figura A.2: Variación del peso variando m y manteniendo u

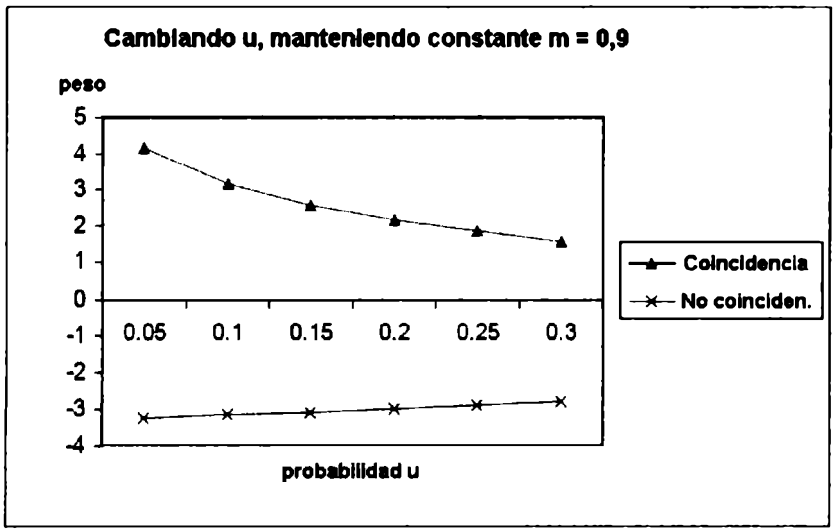


Figura A.3: Variación del peso variando u y manteniendo m

Si se mantiene constante u y varia m, *el valor de no coincidencia* disminuye a medida que **aumenta** m, mientras que si se mantiene m y varia u, *el valor de coincidencia* **disminuye**.

Apéndice B

Manual de referencia de Matcher

B.1. Instalación

Se describe en esta sección como utilizar Matcher desde línea de comandos (matcher-cli) y desde Jump (matcher-jump).

B.1.1. Instalar matcher-cli

La librería matcher-cli no requiere instalación. Es un archivo Jar ejecutable que necesita que se indique el classpath donde se encuentren las librerías externas que necesita. Para más información de como indicar el classpath a un archivo Jar puede consultar [jar].

B.1.1.1. Dependencias

```
+- args4j:args4j:jar:2.0.10:compile
+- log4j:log4j:jar:1.2.9:compile
\- edu.unlp.matcher:matcher-core:jar:0.2.1:compile
   +- edu.unlp.matcher:matcher-api:jar:0.2.1:compile
   |  \- trove:trove:jar:2.1.1:compile
   +- com.wcohen.secondstring:secondstring:jar:2006:compile
   +- com.wcohen.secondstring:secondstring:jar:2003:compile
   +- commons-logging:commons-logging:jar:1.1.1:compile
   +- org.ujmp:ujmp:jar:0.1.1:compile
   |  \- commons-math:commons-math:jar:1.2:compile
   +- commons-codec:commons-codec:jar:1.3:compile
   +- com.svcon:jdbf:jar:1.0:compile
   +- org.codehaus.castor:castor:jar:1.2:compile
   +- xerces:xercesImpl:jar:2.6.2:runtime
   +- org.ostermiller:utils:jar:1.07.00:compile
   +- commons-configuration:commons-configuration:jar:1.6:compile
   |  +- commons-collections:commons-collections:jar:3.2.1:compile
   |  +- commons-lang:commons-lang:jar:2.4:compile
   |  +- commons-digester:commons-digester:jar:1.8:compile
   |  |  \- commons-beanutils:commons-beanutils:jar:1.7.0:compile
   |  \- commons-beanutils:commons-beanutils-core:jar:1.8.0:compile
   \- org.codehaus.plexus:plexus-container-default:jar:1.1.0:compile
```

```
      +- org.codehaus.plexus:plexus-utils:jar:1.4.5:compile
      +- org.codehaus.plexus:plexus-classworlds:jar:2.0.0:compile
      +- org.apache.xbean:xbean-reflect:jar:3.4:compile
      |  \- commons-logging:commons-logging-api:jar:1.1:compile
      \- com.google.code.google-collections:google-collect:jar:
snapshot-20080530:compile
```

B.1.2. Instalar matcher-jump

La librería matcher-jump es un plugin Jump. Para poder utilizar la librería instalar Jump y luego el plugin como se indica en [Jum].

B.1.2.1. Dependencias

```
+- edu.unlp.matcher:matcher-core:jar:0.2.1:compile
|  ... ( dependencias de matcher-core )
+- com.vividsolutions:jump-core:jar:1.2:provided
|  +- com.vividsolutions:jts:jar:1.6.0:provided
|  +- com.vividsolutions:jump-workbench:jar:1.2:provided
|  +- jama:jama:jar:1.0.1:provided
|  +- jdom:jdom:jar:0.7:provided
|  \- bsh:bsh:jar:2.0b1:provided
\- log4j:log4j:jar:1.2.9:compile
```

B.2. Ejecución de tareas desde línea de comandos

Se provee un archivo Jar ejecutable (matcher-cli.jar). Mediante este Jar se puede ejecutar tareas. Ejemplo:

```
# java -jar mather-cli.jar -task=TASK_ID -file=PATH
```

El parámetro `task` es obligatorio e indica que tarea ejecutar. El parámatro file es opcional e indica el path absoluto del archivo XML de configuración de tareas descripto en la sección anterior. En caso de no especificarse intenta leer el archivo `match.xml` en el directorio actual.

B.3. Archivo XML de configuración

El archivo de configuración XML permite configurar tareas y componentes (beans). Comenzamos describiendo algunos ejemplos sencillos antes de brindar la referencia completa de que elementos se pueden incluir en el archivo XML.

B.3.1. Ejemplos

Ejemplo 1: Configuración de funciones Sean las siguientes tablas, sobre las cuales se necesita realizar el matching. Por un lado la tabla bel_city y por el otro store_city que contienen ciudades. Se quiere encontrar el matching utilizando los nombres de las ciudades. Por el lado de bel_city, el nombre se encuentra en la columna name. Por el otro en la columna store_city.

```
TABLE bel_city (
  gid INTEGER,
  bel_city_i INTEGER,
  id TEXT,
  name TEXT,
  the_geom public.geometry,
  piet_id SERIAL
)

TABLE store_city (
  store_city VARCHAR(30),
  city_id INTEGER
)
```

El archivo de configuración sería similar al siguiente:
match.xml

```
1  <matcher-config>
2
3   <mtask>
4    <id>cities</id>
5
6    <!— Lado 1 del match. —>
7    <source1>
8     <info>bel_city</info>
9     <params>
10     <param>
11      <key>conFactory</key>
12      <value>belgica</value>
13     </param>
14    </params>
15   </source1>
16
17   <!— Lado 2 del match —>
18  <source1>
19    <info>store_city</info>
20    <params>
21     <param>
22      <key>conFactory</key>
23      <value>belgica</value>
24     </param>
25    </params>
26   </source1>
27
28   <functions>
29    <function>
30     <id>f1</id>
31     <ref>individualColumnScorer</ref>
32     <columnMapping>
33      <columnMap>
```

```
34          <column1>name</column1>
35          <column2>store_city</column2>
36          <ref>jaroWinkler</ref>
37        </columnMap>
38      </columnMapping>
39    </function>
40  </functions>
41 </mtask>
42
43 <connections>
44  <conFactory>
45    <id>belgica</id>
46    <user>postgres</user>
47    <pass>postgres</pass>
48    <driver>org.postgresql.Driver</driver>
49    <url>jdbc:postgresql://localhost:5432/piet</url>
50  </conFactory>
51 </connections>
52 </matcher-config>
```

Dentro del elemento (ó tag) **matcher-config** se configura los elementos necesarios para poder ejecutar las tareas (procesos de integración). El tag **mtask** define una tarea. En el archivo puede haber uno o más de estos elementos. En el ejemplo contiene los siguientes elementos:

- *id:* un identificador de la tarea.
- *source1:* origen de los datos del lado 1.
- *source2:* origen de los datos del lado 2.
- *functions:* donde se declaran las funciones.

Dentro del tag **functions** de define una función mediante el tag **function**. A la función se debe asignar un identificador mediante el tag **id** y cómo comparar los registros. Esto se realiza con el tag **ref** en el cuál se asocia un identificador a un bean que compare registros, ver la sección B.7.

La función **individualColumnScorer** provista por Matcher, utiliza funciones de comparación de atributos, ver B.6. Esta necesita mapear columnas con funciones atributos. Esto se hace mediante el tag **columnMapping**, donde se definen uno o más elementos **columnMap**. Este último, como su nombre lo indica, asocia dos columnas, una del **source1** y otra del **source2**, con una función de comparación de atributo, que se define mediante el tag **ref**.

En **column1** ponemos la columna **bel_city.name** y en **column2** la columna **store_city.store_city**. En **ref** hemos definido la función de comparación **jaroWinkler**. Esta función provista sirve para comparar dos strings.

Ejemplo 2: Configuración de acciones Suponer que el resultado se quiere almacenar en una tabla. La tabla podría ser similar a la siguiente:

```
gis_olap_cities (
  gisid VARCHAR(32),
  olapid VARCHAR(32),
)
```

Donde en **gisid** se almacena la columnabel_city.gisid y en **olapid** la columna **store_city.city_id**. Para indicar esta acción (y acciones en general), se utiliza, dentro del tag **mtask** el elemento **actions**. Dentro del mismo se indican una o más acciones.

```
1  <matcher-config>
2
3   <mtask>
4    <id>cities</id>
5     [...]
6    <actions>
7     <action>
8      <id>accion1</id>
9      <ref>exportToTable</ref>
10     <params>
11      <param>
12       <key>table</key>
13       <value>gis_olap_cities</value>
14      </param>
15       <param>
16        <key>conFactory</key>
17        <value>belgica</value>
18       </param>
19       <param>
20        <key>C_gisid</key>
21        <value>S1_bel_city_id</value>
22       </param>
23       <param>
24        <key>C_olapid</key>
25        <value>S2_city_id</value>
26       </param>
27       <param>
28        <key>truncateTable</key>
29        <value>true</value>
30       </param>
31      </params>
32    </action>
33   </actions>
34  </mtask>
35 </matcher-config>
```

En el ejemplo se ve un solo **action**. En **ref** se puso **exportToTable**, el cual exporta el resultado a la tabla indicada en table, conectándose a la BD con conFactory belgica y en la columna **gisid** se pondrá el valor de la columna del bel_city_id del source1 y en la columna **olapid** se pondrá el de la columna city_id del source2.

Ejemplo 3: Incluir funciones y/o acciones definidas por el usuario En el ejemplo anterior tanto las funciones como las acciones se indican con el tag `ref`. En este elemento se indica un identificador de un bean, el cual está configurado en archivos dentro de Matcher ó puede ser configurado dentro del archivo XML.

Sea *fooFunction* función de comparación definida por el usuario implementada en la clase `foo.MiFuncion`, el archivo de configuración sería:

```
1  <matcher-config>
2
3   <mtask>
4    <id>task1</id>
5    [...]
6    <functions>
7     <function>
8      <id>f1</id>
9      <ref>fooFunction</ref>
10     </function>
11   </functions>
12  </mtask>
13  [...]
14  <bean>
15   <id>fooFunction</id>
16   <class>foo.MiFuncion</class>
17   <type>RECORD_FUNCTION</type>
18  </bean>
19
20 </matcher-config>
```

B.3.2. Elementos del archivo de configuración

En esta sección se detalla que elementos van en el archivo de configuración.

matcher-config Elemento raíz del archivo XML.

- **mtask** donde se define la tarea. Puede haber 1 ó más.
- **connections** donde se definen las conexiones. Puede haber 0 ó 1.
- **bean** donde se definen los bean. Puede haber 0 ó más.
- **taskExecuter** donde se define el componente ejecutador de tareas. Puede haber 0 ó 1. Por defecto taskExecuter.

mtask Tarea que realiza un match entre dos conjuntos de datos. Representa un requerimiento. Se pueden definir varias funciones para resolver el matching y acciones que serán ejecutadas secuencialmente al finalizar la ejecución de las tareas.

- **id** un identificador. Obligatorio.
- **source1** Conjuntos de datos. Obligatorio.
- **source2** Conjuntos de datos. Obligatorio.

functions elemento donde se indican las funciones. Obligatorio.

actions elemento donde se escriben uno o más elementos action. Opcional.

filter funciones utilizadas por el método de comparación definido en `blocking-method`.

blocking-method método para realizar las comparaciones.

matcherId identificador del bean que garantiza asignación única.

source1/source2 Elemento donde se especifica la fuente de datos desde se leen los registros a procesar.

ref identificador del componente. Obligatorio.

params parámetros. Opcional.

functions Elemento donde se indican las funciones que ejecutará la tarea de manera secuencial.

function donde se define la función. Debe definirse al menos 1 (una) función.

function Función utilizada para asociar un peso a la arista que asocia un elemento del side1 con otro del side2.

id identificador. Obligatorio.

ref identificador de un bean utilizado para la comparación. El bean debe implementar la interfase RecordFunction. Obligatorio.

columnMapping donde se definen los columnMap. Opcional.

params parámetros de la función. Opcional.

columnMapping Contenedor de elementos columnMap.

columnMap Asocia atributos de side1, side2 con una función de comparación.

columnMap Asocia un atributo del side1 con otro del side2 a una función de comparación de atributos.

column1 nombre del atributo del side1. Generalmente el nombre de la columna. Obligatorio.

column2 nombre del atributo del side2. Generalmente el nombre de la columna. Obligatorio.

ref identificador del bean función que compara y retorna un valor indicando que probabilidad hay de que tengan el mismo valor. Obligatorio. El bean debe implementar la interfase AttributeFunction.

params parámetros de la función. Opcional.

filter Conjunto de funciones.

functions elemento donde se indican las funciones. Obligatorio.

blocking-method Identificador del bean que realiza las comparaciones aplicando filtros silos hubiera.

ref identificador del componente. Obligatorio.

params parámetros de la acción. Opcional.

actions Elemento donde se definen las acciones de una tarea.

action acción. Debe definirse al menos 1 (una) acción.

action Acción.

id identificador de la acción dentro de la tarea. Obligatorio.

ref identificador del bean que se ejecutará. Obligatorio. El bean debe implementar la interfase Action.

params parámetros de la acción. Opcional.

connections Elemento donde se definen las conexiones.

connection Elemento donde se configura el acceso utilizado conexiones a motores de base de datos. Debe definirse al menos 1 (uno).

connection Elemento donde se definen los datos necesarios para conectarse a un motor de base de datos.

id un identificador que es utilizado desde los elementos conFactory. Obligatorio.

user nombre del usuario utilizado para conectarse a la base de datos. Opcional.

pass contraseña de acceso. Opcional.

url URL de la base de datos. Obligatorio

driver clase que implementa la interfase Driver. Utilizada para obtener la conexión. Obligatorio

bean Elemento donde define un bean que luego puede ser referenciado en los elementos ref.

id un identificador. Obligatorio

class nombre completo de la clase. Obligatorio

type El tipo del bean bean. Obligatorio. Una opción entre:

RECORD_FUNCTION: Es un bean que implementa la interfase RecordFunction.

ATTRIBUTE_FUNCTION: Es un bean que implementa la interfase AttributeFunction.

ACTION: Es un bean que implementa la interfase Action.

MATCHER: Es un bean que implementa la interfase IMatcher.

params Elemento donde se definen parámetros.

param Parámetro. Se debe definir al menos 1 (un) parámetro.

param Parámetro. Los parámetros son un par de elementos del estilo clave/-valor.

key clave del parámetro.

value valor del parámetro

taskExecuter Elemento donde define el componente ejecutador de tareas.

B.4. Archivo de definición de componentes

Matcher es una herramienta de record linkage extensible, es decir que se pueden agregar diferentes componentes sin necesidad de modificar las diferentes interfases.
Para definir los componentes se utilizan archivos XML:

- *META-INF/components.xml*: Archivo XML donde se definen los componentes de Matcher.
- *plexus.xml*: Archivo XML para definir ó redefinir componentes por parte del usuario.
- *archivo XML de configuración de tareas*: Mediante el tag bean se pueden definir componentes.

La precedencia de lectura es:

1. *archivo XML de configuración de tareas*
2. *plexus.xml*
3. *META-INF/components.xml*

Es decir, si en los tres archivos tiene un componente con el mismo nombre, se toma el que se define en el archivo de definición de tareas.

B.4.1. Configuración

Los componentes en el archivo de configuración se definen como se describió en B.3.2, mientras que en el archivo plexus.xml y componentes.xml los componentes se definen como:

```
<component>
 <instantiation-strategy>per-lookup</instantiation-strategy>
 <role>TIPO-DE-COMPONENTE</role>
 <role-hint>IDENTIFICADOR</role-hint>
 <implementation>CLASE-JAVA</implementation>
</component>
```

Por ejemplo el componente fuente de datos sql se define como:

```
<component>
 <instantiation-strategy>per-lookup</instantiation-strategy>
 <role>ar.edu.unlp.matcher.api.DataSource</role>
 <role-hint>sql</role-hint>
 <implementation>ar.edu.unlp.matcher.util.SQLDataSource</implementation>
</component>
```

Para instanciar los componentes definidos en estos dos últimos archivos se utiliza el framework Plexus, para más información ver [ple].

B.5. Fuentes de datos

Se describen en esta sección los componentes *Fuente de datos* provistos. Un componente es *Fuente de Datos* si implementa la interfase

ar.edu.unlp.matcher.api.DataSource

B.5.1. csv

Permite seleccionar un archivo CSV[csv] como fuente de datos. Los parámetros que acepta son:

- *delimiter:* Carácter delimitador de campos. Por defecto (,).
- *includeHeaders:* Indica si el archivo contiene cabecera. Por defecto (true). Cuando incluye cabecera, el título asociado a los atributos se toma de la primera fila y además es ignorada en la comparación. Cuando `includeHeaders=false`, la primera fila no es ignorada y el título de los campos es creado concatenando el valor del parámetro `headerTemplate` y un número secuencial ascendente a partir de uno. Por ejemplo si `headerTemplate=field-`, la primera columna se titula field-1, la segunda field-2, y así sucesivamente.
- *headerTemplate:* Nombre utilizado como título de las columnas cuando `includeHeaders=false`. Por defecto (field-).

Nota: La cantidad de columnas del archivo se toma de la primera fila. Si en el archivo alguna fila contiene más o menos columnas es ignorada y logeada en el log.

B.5.2. dbf

Permite seleccionar un archivo dBase[dba] como fuente de datos. No utiliza parámetros.

Precaución: No soporta campos MEMO.

B.5.3. sql

Permite leer los registros a partir de una tabla ó de una consulta SQL como fuente de datos. Los parámetros que acepta son:

- *conFactory:* Identificador del ConnectionFactory de donde debe obtener la conexión JDBC para conectarse a la base de datos.

El valor asociado a este bean si comienza con `select` se toma como una consulta, sino el texto ingresado, por ejemplo `t1` se transforma en la consulta `select * from t1`.
Este bean requiere conectarse a la base de datos, para ello utiliza el componente conFactory asociado.

Connection Factory este componente contiene la información necesaria para conectarse a la base de datos. Los datos asociados son:

- *id:* Identificador. Este nombre es el que se indica en el bean sql.
- *user:* Nombre del usuario utilizado en la autenticación.
- *pass:* Contraseña utilizada en la autenticación.
- *driver:* Clase del driver. Por ejemplo para conectarse a un motor PostreSQL es `org.postgresql.Driver`.
- *url:* URL de la base de datos. Por ejemplo en PostreSQL es de la forma: `jdbc:postgresql://<host>:<port>/<database>`

B.6. Funciones de comparación entre atributos

Se describen en esta sección los componentes *Función de comparación de atributos* provistos. Un componente es *F. comparación de atributos* si implementa la interfase

`ar.edu.unlp.matcher.api.AttributeFunction`

Este tipo de componentes se utilizan dentro de las funciones de comparación de registros.

B.6.1. jaroWinkler

Función de comparación de atributos. Es un Adapter [EGV95] a la función `com.wcohen.ss.JaroWinkler` de la librería secondstring [sec][CRF03] versión 20060615. Esta libería provee funciones de comparación de strings.

B.6.1.1. Parámetros

minScore Un número real. Si el resultado de la comparación no supera este valor, retorna 0. Opcional.

ignoreAccents Un valor booleano. `false` por defecto. Indica si existe diferencia entre vocales acentuadas en la comparación. Por ejemplo, si ignoreAccents=true, la comparación entre camión y camion retorna 1. Esta opción reemplaza los carácteres acentuados por el mismo sin acento. Se recomienda activar esta opción si se comparan palabras en un idioma en el cuál el alfabeto puede contener palabras acentuadas solamente.

ignoreSpaces Un valor booleano. `true` por defecto. Quita los carácteres espacio de las palabras.

Esta función es útil para nombres cortos y en donde importe el órden de las palabras.

B.6.2. jaroWinklerTFIDF

Función de comparación de atributos. Es un adapter a la función

`com.wcohen.ss.JaroWinklerTFIDF`

de la librería secondstring versión 20060615.

B.6.2.1. Parámetros

minScore Un número real. Si el resultado de la comparación no supera este valor, retorna 0. Opcional.

ignoreAccents Un valor booleano. `false` por defecto. Indica si existe diferencia entre vocales acentuadas en la comparación. Por ejemplo, si ignoreAccents=true, la comparación entre camión y camion retorna 1. Esta opción reemplaza los carácteres acentuados por el mismo sin acento. Se recomienda activar esta opción si se comparan palabras en un idioma en el cuál el alfabeto puede contener palabras acentuadas solamente.

ignoreSpaces Un valor booleano. `false` por defecto. Quita los carácteres espacio de las palabras.

Esta función es útil para nombres y frases en donde no importe el órden de las palabras.

B.6.2.2. Comparación entre jaroWinkler y jaroWinklerTFIDF

valor1	valor2	función	score
Gardella Juan	Juan Gardella	jaroWinklerTFIDF	1
Gardella Juan	Juan Gardella	jaroWinkler	0.7128205128205128

B.6.3. slim

Función de comparación de atributos. Compara palabras ignorando el orden de las mismas. Es una función efectiva pero devuelve casi siempre valores altos cuando existe pocas coincidencias. Su uso recomendado es para potenciar otra función de comparación.

B.6.3.1. Parámetros

minScore Un número real. Si el resultado de la comparación no supera este valor, retorna 0. Opcional.

ignoreAccents Un valor booleano. `false` por defecto. Indica si existe diferencia entre vocales acentuadas en la comparación. Por ejemplo, si ignoreAccents=true, la comparación entre camión y camion retorna 1. Esta opción reemplaza los carácteres acentuados por el mismo sin acento. Se recomienda activar esta opción si se comparan palabras en un idioma en el cuál el alfabeto puede contener palabras acentuadas solamente.

ignoreSpaces Un valor booleano. `false` por defecto. Quita los carácteres espacio de las palabras.

B.6.4. dmethapone

Función de comparación de atributos. Compara fonéticamente palabras utilizando el algoritmo de búsqueda Double Methapone[dme] para realizar la comparación.

Esta función es un adapter a la implementación del algoritmo provista en la librería commons-codec[coma].

B.6.5. numberDistance

Compara dos números $val1$ y $val2$ de la siguiente manera:

$$\frac{min(val1, val2)}{max(val1, val2)}$$

Retorna un valor de aproximación útil para datos no exactos como por ejemplo población, superficies, etc.

B.6.5.1. Parámetros

minScore Un número real. Si el resultado de la comparación no supera este valor, retorna 0. Opcional.

B.6.6. numberEqual

Función de comparación de atributos. Esta función compara dos números $val1$ y $val2$ de la siguiente manera:

$$numberEqual(val1, val2) = \begin{cases} val1 = val2 \rightarrow 1 \\ val1 \neq val2 \rightarrow 0 \end{cases}$$

Esta función es innecesaria si los datos se encuentran en una base de datos ya que realiza lo mismo que un join.

B.6.7. exactString

Compara dos strings por exactitud. Si son iguales retorna 1, sino 0.

B.6.7.1. Parámetros

caseSensitive Un valor booleano. `false` por defecto. Indica si la comparación debe ignorar o no las diferencias entre mayúsculas y minúsculas.

ignoreAccents Un valor booleano. `false` por defecto. Indica si existe diferencia entre vocales acentuadas en la comparación. Por ejemplo, si ignoreAccents=true, la comparación entre camión y camion retorna 1. Esta opción reemplaza los carácteres acentuados por el mismo sin acento. Se recomienda activar esta opción si se comparan palabras en un idioma en el cuál el alfabeto puede contener palabras acentuadas solamente.

ignoreSpaces Un valor booleano. `true` por defecto. Quita los carácteres espacio de las palabras.

B.6.8. substring

Compara secciones de strings por exactitud. Si son iguales retorna 1, sino 0. Por defecto compara sólo la primera letra. Esta función es útil para utilizarla como filtro en los métodos de indexación (blocking).

B.6.8.1. Parámetros

beginIndex Un valor entero. 0 por defecto. Indica desde que posición comienza la sección a comparar.

endIndex Un valor entero. 1 por defecto. Indica hasta que posición se compara.

caseSensitive Un valor booleano. `false` por defecto. Indica si la comparación debe ignorar o no las diferencias entre mayúsculas y minúsculas.

ignoreAccents Un valor booleano. `false` por defecto. Indica si existe diferencia entre vocales acentuadas en la comparación. Por ejemplo, si ignoreAccents=true, la comparación entre camión y camion retorna 1. Esta opción reemplaza los carácteres acentuados por el mismo sin acento. Se recomienda activar esta opción si se comparan palabras en un idioma en el cuál el alfabeto puede contener palabras acentuadas solamente.

ignoreSpaces Un valor booleano. `false` por defecto. Quita los carácteres espacio de las palabras.

B.7. Funciones de comparación entre registros

Se describen en esta sección los componentes *Función de comparación de registros* provistos. Un componente es *Función comparación de registros* si implementa la interfase

`ar.edu.unlp.matcher.api.RecordFunction`

Este tipo de componentes representan al modelo de decisión en el diagrama de la Figura 2.3 porque es el encargado de clasificar un par de registros en:

- `MATCH`
- `NO_MATCH`
- `POSSIBLE_MATCH`

Por lo tanto, una función de comparación de registros es el modelo de decisión en el proceso de integración de datos. Como Matcher permite definir este tipo de funciones por el usuario, no sigue un modelo específico, sino que soporta tantos modelos como funciones se hayan implementado.

B.7.1. IndividualColumnScorer

Función de comparación de registros que utiliza funciones de comparación de atributos para obtener el *score* entre ambos registros.

B.7.1.1. Parámetros

- *minScore:* Un número real. Si el resultado de la comparación no supera este valor, el par de registros comparado se clasifica como `NO_MATCH`. Opcional. Por defecto $minScore = 0{,}7$.
- *weight_i:* Donde $i \in 1..N$, N cantidad de columnas. Un número real entre 0 y 1. Representa el peso que representa la comparación i dentro del resultado total. Opcional, por defecto $weight\_i = 1/N$.

B.7.1.2. Funcionamiento

Sea F una función `IndividualColumnScorer` que tiene configurada tres funciones de comparación de atributos $f1$, $f2$, y $f3$. Suponer que se debe realizar la comparación de los registros $r_1[a_1, a_2, a_3]$ y $r_2[b_1, b_2, b_3]$, donde a_1, a_2, a_3, b_1, b_2 y b_3 son los atributos de los registros. Suponer que f_1 compara los atributos a_1 y b_1, f_2 los atributos a_2 y b_2 y la función f_3 los atributos a_3 y b_3.

$$F = \{(f_1, a_1, b_1), (f_2, a_2, b_2), (f_3, a_3, b_3)\}$$

La correspondiente configuración en XML sería:

```
1  <function>
2   <id>func1</id>
3   <ref>individualColumnScorer</ref>
4
5   <columnMapping>
6
7   <columnMap>
8    <column1>a1</column1>
9    <column2>b1</column2>
10   <ref>f1</ref>
11  </columnMap>
12
13  <columnMap>
14   <column1>a2</column1>
15   <column2>b2</column2>
16   <ref>f2</ref>
17  </columnMap>
18
19  <columnMap>
20   <column1>a3</column1>
21   <column2>b3</column2>
22   <ref>f3</ref>
23  </columnMap>
24
25  </columnMapping>
26  </function>
```

Para comparar los registros r_1 y r_2 se envia el mensaje `score()` a F.

```
float score = F.score(r1,r2);
```

Una vez enviado el mensaje, F utiliza la información del mapeo de columnas. Suponer la primera configuración que lee es (f_1, a_1, b_1). En este caso instancia la función f_1[1] y luego obtiene el nombre del atributo del registro del lado 1 a_1 y el nombre del atributo del registro del lado 2 b_1. Con esta información procede a la comparación de ambos atributos.

El resultado de la comparación es comparado con el valor del parámetro *minScore*, si es mayor o igual a *minScore* almacena el resultado, sino asume que el resultado de la comparación es 0. El proceso se repite para las demás funciones de comparación.

Los resultados son almacenados en un arreglo temporal, donde en el primer elemento del arreglo se almacena la comparación de la primera función, en el segundo elemento el resultado de la segunda comparación y así sucesivamente. El resultado retornado por F, es por defecto, el promedio entre los resultados. En este caso retorna:

$$\frac{f_1(a_1, b_1) + f_2(a_2, b_2) + f_3(a_3, b_3)}{3}$$

B.7.1.3. Peso de las funciones

Puede que el promedio de los resultados no sea conveniente en algunas circunstancias, por ejemplo cuando haya resultados con mayor "peso" que otros.

Basandose en el ejemplo anterior, se quiere que f_1 represente el 80 % del resultado de la comparación entre r_1 y r_2. En este caso el resultado de F debe ser:

$$0{,}8 * f_1(a_1, b_1) + 0{,}1 * f_2(a_2, b_2) + 0{,}1 * f_3(a_3, b_3)$$

Notar que $f1(a_1, b_1)$ representa el 80 % del resultado, $f_2(a_2, b_2)$ y $f_3(a_3, b_3)$ el 20 % restante, divido el peso entre ambas equitativamente. Se podría querer a cada una un peso distinto:

$$0{,}5 * f_1(a_1, b_1) + 0{,}2 * f_2(a_2, b_2) + 0{,}3 * f_3(a_3, b_3)$$

La única restricción es que la suma de los pesos asignados sea 1 para garantizar que el resultado de la función no quede fuera de rango.

`IndividualColumnScorer` permite asignar un peso a las funciones de comparación mediante el uso del parámetro *weight*. Por ejemplo en XML se traduce el primer ejemplo a:

```
1  <function>
2   <id>func1</id>
3   <ref>individualColumnScorer</ref>
4   <columnMapping>
5    <columnMap>
6     <column1>a1</column1>
7     <column2>b1</column2>
8     <ref>f1</ref>
9    </columnMap>
10   <columnMap>
```

[1] f_1 es el nombre del bean.

```
11      <column1>a2</column1>
12      <column2>b2</column2>
13      <ref>f2</ref>
14    </columnMap>
15    <columnMap>
16      <column1>a3</column1>
17      <column2>b3</column2>
18      <ref>f3</ref>
19    </columnMap>
20  </columnMapping>
21  <params>
22      <param>
23        <key>weight_1</key>
24        <value>0.8</value>
25      </param>
26      <param>
27        <key>weight_2</key>
28        <value>0.1</value>
29      </param>
30      <param>
31        <key>weight_3</key>
32        <value>0.1</value>
33      </param>
34    </params>
35 </function>
```

`IndividualColumnScorer` calcula el parámetro *weight* en los mapeos donde no se han definido los pesos, siempre que se hayan configurado pesos en el rango estipulado. En el caso anterior la distribución equitativa la puede calcular, por lo tanto se reduce a:

```
1  <function>
2   <id>func1</id>
3   <ref>individualColumnScorer</ref>
4   <columnMapping>
5     <columnMap>
6       <column1>a1</column1>
7       <column2>b1</column2>
8       <ref>f1</ref>
9     </columnMap>
10    <columnMap>
11      <column1>a2</column1>
12      <column2>b2</column2>
13      <ref>f2</ref>
14    </columnMap>
15    <columnMap>
16      <column1>a3</column1>
17      <column2>b3</column2>
18      <ref>f3</ref>
19    </columnMap>
20  </columnMapping>
```

```
21  <params>
22    <param>
23      <key>weight_1</key>
24      <value>0.8</value>
25    </param>
26  </params>
27 </function>
```

B.7.2. FellegiSunter

Función de comparación de registros que implementa el modelo de comparación propuesto en el Apéndice A. Utiliza funciones de comparación de atributos para comparar los campos.

B.7.2.1. Parámetros

- *upperLimit:* Un número real. Si el resultado de la comparación supera este valor, el par de registros comparado se clasifica como `MATCH`. Opcional. Por defecto 5.
- *lowerLimit:* Un número real. Si el resultado de la comparación no supera este valor, el par de registros comparado se clasifica como `NO_MATCH`. Opcional. Por defecto 1.
- *partialAgreement:* Un boolean, `true` ó `false`. Si es `true` se utiliza el peso de la comparación de los atributos en el resultado de la comparación para que tengan más peso si la función retorna mejor score. Opcional. Por defecto `true`.
- *uprob_i:* Donde $i \in 1..N$, N cantidad de columnas. Un número real. Representa la probabilidad u[2] asociada a la comparaciíon definida en el `columnMap` i. Opcional, por defecto 0.01.
- *mprob_i:* Donde $i \in 1..N$, N cantidad de columnas. Un número real. Representa la probabilidad m[3] asociada a la comparaciíon definida en el `columnMap` i. Opcional, por defecto 0.9.

Si el resultado de la comparación se encuentra entre *lowerLimit* y *upperLimit* el par se clasifica como `POSSIBLE_MATCH`.

B.7.2.2. Funcionamiento

El funcionamiento y fundamento de la función se explican en el Apéndice A. Para el cálculo del peso de la comparación se partialAgreement=false se realiza como se explica en la Sección A.2 y en la Sección A.3. En caso que `partialAgreement=true` el peso del campo que se asigna, según sea coincidencia o no, se multiplica por el peso de la comparación de los campo. Es decir, partialAgreement afecta como se calcula el peso de los campos y por lo tanto el peso asociado al par de registros. En el siguiente seudo-código se ejemplifica el cálculo de los campos:

[2] Ver Sección A.1.2 para más información.
[3] Ver Sección A.1.1 para más información.

```
atf = funcionComparacionAtributos()
//Se compara los atributos y el resultado lo devuelve en score
score = atf.score(at1, at2)
SI [partialAgreement=false]:
  result = score > 0 ? agrees[i] : disagrees[i]
SINO:
 result = score > 0 ? agrees[i]*score : disagrees[i]*score
```

Donde `agrees[i]` y `disagrees[i]` son los pesos asociados a la comparación del campo i como se especifica en la Sección A.2.

B.8. Modos de comparación

En esta sección se describe el componente `fullIndex` que es el único modo de comparación provisto por Matcher. `fullIndex` implementa la interfase

ar.edu.unlp.matcher.api.ComparationMethod

B.8.1. fullIndex

`fullIndex` realiza una comparación de todos los registros de source1 contra todos los registros de source2 definidos en una tarea. Si la tarea tiene configurado uno o más filtros los utiliza, sino realiza la comparación con las funciones directamente.

B.9. Acciones

Se describen en esta sección los componentes *Acción* provistos. Este tipo de componentes implementan la interfase

`ar.edu.unlp.matcher.api.Action`

B.9.1. Parámetros comunes

Todas las acciones provistas en Matcher tienen 3 (tres) parámetros.

function Identificador de función. Opcional. Indica el resultado de que función procesar. Si no se especifica, se procesa el resultado de la mejor función.

active true (opción por defecto) ó false. Indica si la acción debe ejecutarse.

target Una opción entre: `match`, `pmatch`, `side1`, `side2`. Opcional. Opción por defecto `match`.

- **match** Indica que se procesará los registros clasificados como `MATCH`.
- **pmatch** Indica que se procesará los registros clasificados como `POSSIBLE_MATCH`.
- **side1** Indica que se procesará los registros que no obtuvieron coincidencia del source1.
- **side2** Indica que se procesará los registros que no obtuvieron coincidencia del source2.

B.9.2. ExportToTable

Esta acción exporta a una tabla el resultado de la tarea a la cual está asociada.

B.9.2.1. Párametros

table tabla a la cual copiará los resultados.

conFactory nombre de la conexión definida en connections.

truncateTable true ó false. Por defecto false. Indica que previo a la inserción debe hacer o no un truncate a la table.

C_columnaX todos los parámetros que comiencen con 'C_' indican que en la columna columnaX de la tabla definida en table se copie el valor de la columna columnaY del registro. La columnaY es definido en el value del parámetro, donde también se indica si es de un registro del side1 o del side2. El valor de este parámetro varía según el valor del parámetro target.

Si *target* = *match* el valor debe ser de la forma

$$S[1|2]\_columnaY$$

Donde si es de la forma S1_*columnaY* indica que copie el valor de la columa columnaY de la tabla definida en side1. Si es de la forma S2_, realice lo mismo pero que copie de la columna de la tabla definida en side2.

Si *target* = *side*1 ó *target* = *side*2 el valor debe ser de la forma

$$columnaY$$

Donde *columnaY* indica que copie el valor de la columa columnaY de la tabla definida en el side correspondiente a definido en target. El valor de este parámetro debe ser de la forma: S[1|2]_columnaY. Donde si es de la forma S1_columnaY indica que copie el valor de la columa columnaY de la tabla definida en side1. Si es de la forma S2_columnaZ, realice lo mismo pero que copie de la columna de la tabla definida en side2.

B.9.3. PrinterAction

Esta acción imprime en pantalla el resultado. Es la acción que se ejecuta si no se ha indicado ninguna acción en la tarea. Para modificar la acción por defecto debe editar el archivo matcher.properties:

```
matcher.properties
--------------------
ar.edu.uba.matcher.defaultAction=printerAction
```

B.9.4. ExportToFile

Esta acción exporta a un archivo el resultado de la tarea a la cual está asociada. Al momento de escribir este documento soporta sólo exportar a formato CSV.

B.9.4.1. Párametros

path URI del archivo.

B.10. Asignación única

En esta sección se describen los componentes que garantizan asignación única. Este tipo de componentes no requieren parámetros e implementan la interfase

ar.edu.unlp.matcher.api.IMatcher

B.10.1. pqm

El componente `pqm` (Priority Queue Matcher) implementa el algoritmo definido en la Sección 4.3.1.

B.10.2. munkres

El componente `munkres` implementa el algoritmo definido en 4.3.2.

B.11. Ejecutador de tareas

En esta sección se describe el componente ejecutador de tareas provisto en Matcher.

B.11.1. taskExecuter

Este componente ejecuta una tarea en una serie de etapas que permiten ser modificadas mediante un archivo de configuración. Las etapas que ejecuta son:

1. `initProcess` Informa mediante el logger[4] la tarea a ejecutar e información asociada a la misma.
2. `readSources` Lee los registros de las fuentes de datos y los almacena en una lista.
3. `compare` Compara los registros con el componente correspondiente.
4. `classify` Clasifica los resultados.
5. `uniqueAssign` Si la tarea tiene asociado un componente para garantizar asignación única la ejecuta en esta etapa.
6. `actionExecuter` Lee las acciones asociadas a la tarea y las ejecuta.

[4]Objeto encargado de enviar mensajes de aplicación a dispositivos de salida, por ej. la pantalla, un archivo, etc. Algunos frameworks conocidos para esta tarea son log4j, commons-logging, etc.

B.11.2. Configuración

Las etapas pueden ser modificadas en el archivo de configuración de componentes de Matcher (META-INF/components.xml), por ejemplo taskExecuter se configura de la siguiente manera:

```
<component>
 <instantiation-strategy>per-lookup</instantiation-strategy>
 <role>ar.edu.unlp.matcher.api.TaskExecuter</role>
 <role-hint>taskExecuter</role-hint>
 <implementation>
  ar.edu.unlp.matcher.workflow.impl.WorkFlowImpl
 </implementation>
 <configuration>
  <commands>
   <command>initProcess</command>
   <command>readSources</command>
   <command>compare</command>
   <command>classify</command>
   <command>uniqueAssign</command>
   <command>actionExecuter</command>
  </commands>
 </configuration>
</component>

<component>
 <instantiation-strategy>per-lookup</instantiation-strategy>
 <role>ar.edu.unlp.matcher.workflow.Command</role>
 <role-hint>initProcess</role-hint>
 <implementation>
  ar.edu.unlp.matcher.workflow.impl.commands.InitProcess
 </implementation>
 <configuration>
  <order>99</order>
 </configuration>
</component>

[...]
```

Cada componente se asocia a un orden que representa el orden de ejecución. Por ejemplo si se quiere agregar una etapa al final que realize algún tipo de medición del resultado, se puede hacer de dos formas:

- Agregar una etapa a taskExecuter con el orden más alto.
- Crear un ejecutador de tareas con otro nombre y las etapas de taskExecuter más la nueva. Este opción tener 2 ejecutadores de tareas. Uno con el default y el otro con la etapa de medición, el ejecutador de tareas puede ser configurado en el archivo de configuración de tareas. Por ejemplo:

```
<component>
```

```
    <instantiation-strategy>per-lookup</instantiation-strategy>
    <role>ar.edu.unlp.matcher.api.TaskExecuter</role>
    <role-hint>fooTaskExecuter</role-hint>
    <implementation>
     ar.edu.unlp.matcher.workflow.impl.WorkFlowImpl
    </implementation>
    <configuration>
     <commands>
      <command>initProcess</command>
      <command>readSources</command>
      <command>compare</command>
      <command>classify</command>
      <command>uniqueAssign</command>
      <command>actionExecuter</command>
      <command>fooCommand</command>
     </commands>
    </configuration>
   </component>

   <component>
    <instantiation-strategy>per-lookup</instantiation-strategy>
    <role>ar.edu.unlp.matcher.workflow.Command</role>
    <role-hint>fooCommand</role-hint>
    <implementation>
     mypackage.Foo
    </implementation>
    <configuration>
     <order>1000</order>
    </configuration>
   </component>

   [...]
```

Las etapas son clases que implementan la interfase `Command`[5]

```
public interface Command extends Comparable<Command>{

 public void execute(Context context);

 [...]
}
```

El método más importante es **`execute(Context)`**, que es por el cuál se ejecuta una etapa. El parámetro **`context`** contiene información relacionada a la tarea y sirve también para el pasaje de datos a través de las diferentes tareas.

[5]La clase `Command` no pertenece al **API** de Matcher.

B.12. Internacionalización

El soporte a internacionalización tiene sentido sólo para la interfaz matcher-jump, el módulo matcher-cli no lo soporta.

Los componentes provistos que soportan internacionalización son aquellos que en la interfaz tienen el panel descripción:

- Fuente de datos
- Función de comparación de atributos.
- Función de comparación de registros.
- Acciones
- Métodos de comparación.

Cada componente es una clase implementando la interfaz correspondiente, pero para que tenga soporte a internacionalización debe:

1. Implementar la interfaz `ar.edu.unlp.matcher.api.ClassWithHelp`.
2. Escribir el archivo que se utiliza para autocompletar los parámetros.
3. Escribir el archivo de ayuda.

Por ejemplo el componente `ar.edu.unlp.matcher.util.CSVDataSource` está empaquetado en matcher-core.jar de la siguiente forma:

```
ar/edu/unlp/matcher/util/
CSVDataSource.class
CSVDataSource.properties
CSVDataSource.html
CSVDataSource_es.properties
CSVDataSource_es.html
```

Donde `CSVDataSource.properties` se utiliza para autocompletar los parámetros

```
delimiter=por defecto ",".
includeHeaders=true|false.
headerTemplate=field-
```

`CSVDataSource.html` es un archivo HTML donde se describe el componente.

Para determinar que archivos se utilizan se busca los archivos con el nombre de la clase seguido del lenguaje seteado en la computadora en donde se esté ejecutando[6]: si el idioma es español los archivos que terminen en `_es`, si es inglés en `_en`. Más información de internacionalización (i18n) en Java consultar [i18].

La interfase ClassWithHelp que se define en matcher-api, contiene los siguientes métodos:

[6]En realidad en que Java Virtual Machine (JVM) se ejecuta.

```
public interface ClassWithHelp {
 public String getDescription();
 public Map<String, String> getAutoCompletParameters();
}
```

Si se quiere seguir el modelo de los archivos del ejemplo anterior, queda a cargo del programador esta tarea (leer los archivos en la ubicación de la clase), pero si el componente extiende de la clase

`ar.edu.unlp.matcher.api.impl.AbstractParamsHolder`

que se encuentra en matcher-core, lo único que debe hacer es escribir los archivos.

B.13. Detalle de los componentes

Se describe en esta sección el conjunto de interfaces más importantes

B.13.1. MTask

Representa la tarea. Contiene métodos de edición y lectura.

B.13.2. RecordFunction

Representa la función de comparación de registros. Cada implementación de esta interfase debe definir el método `score()` que es el que retorna la probabilidad de igualdad.

B.13.3. AttributeFunction

Representa la función de comparación de atributos. Cada implementación de esta interfase debe definir el método `score()` que es el que retorna la probabilidad de igualdad.

B.13.4. ColumnMapping y ColumnMap

Representan el componente mapeador de columnas.

B.13.5. IMatcher

Representa el proceso de resolución.

B.13.6. Action

Representa la acción post-ejecución.

B.13.7. TaskExecuter

Representa el ejecutador de tareas.

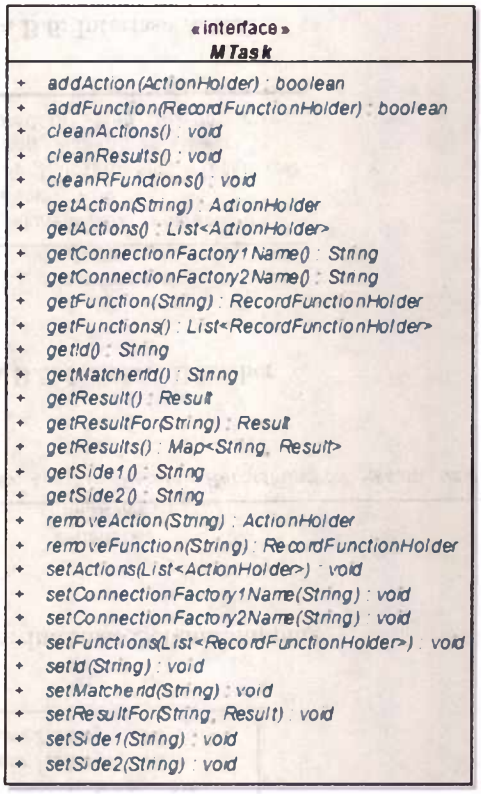


Figura B.1: Interfase MTask

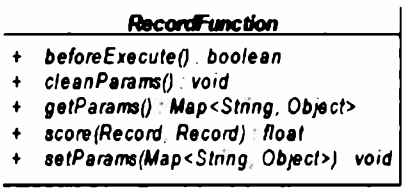


Figura B.2: Interfase RecordFunction

B.13.8. MatcherConfig

Representa la configuración principal. Contiene las tareas y provee el método `run(taskId)` que permite la ejecución de las tareas.

B.13.9. Result

Conjunto salida de la ejecución exitosa de la tarea. Representa a *Match*.

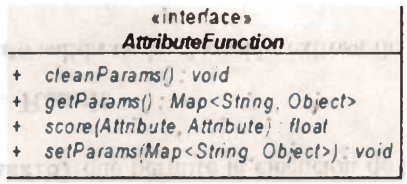


Figura B.3: Interfase AttributeFunction

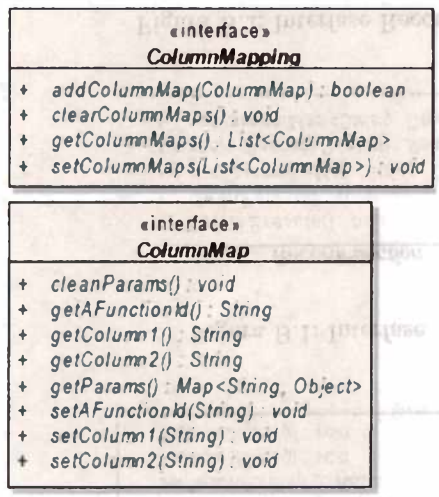


Figura B.4: Interfase ColumnMapping

Figura B.5: Interfase IMatcher

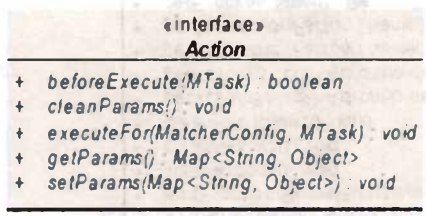


Figura B.6: Interfase Action

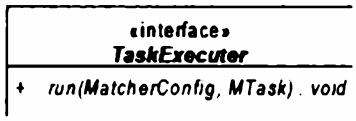


Figura B.7: Interfase TaskExecuter

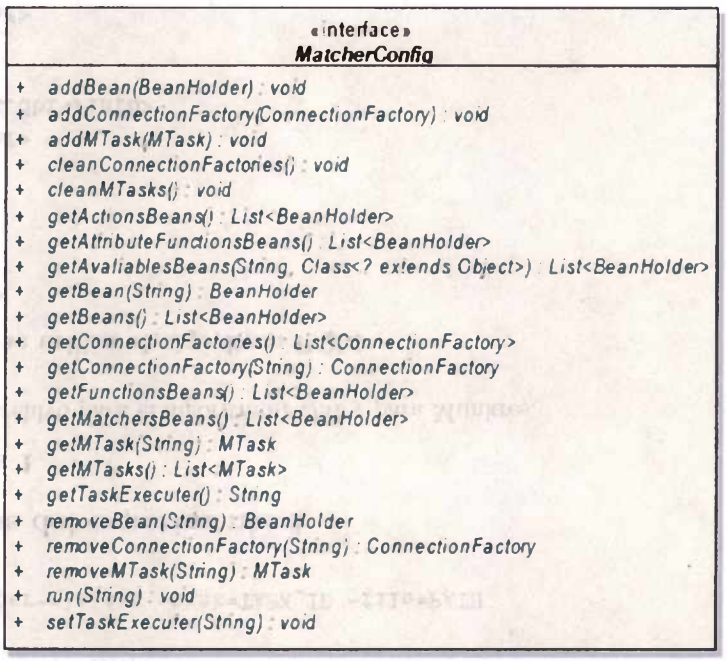


Figura B.8: Interfase MatcherConfig

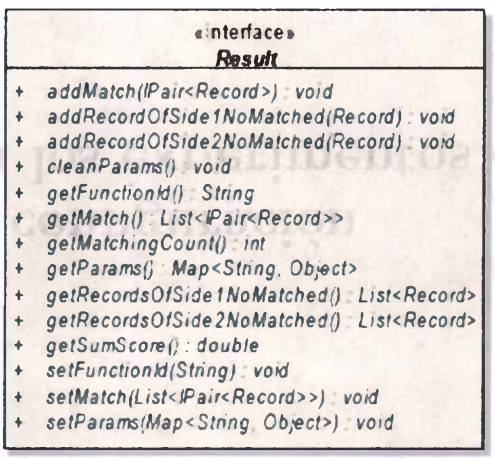


Figura B.9: Interfase Result

Apéndice C

Archivos de configuración utilizados en los experimentos de Matcher

En este apéndice se describen los archivos de configuración de Matcher de los experimentos descriptos en el Capítulo 5. Para ejecutarlos escribir en la línea de comando:

```
# java -jar mather-cli.jar -task=TASK_ID -file=PATH
```

C.1. Archivos del experimento 1

C.1.1. Prueba 1

Se muestra el archivo para el algoritmo PQM y para Munkres.

Archivo cuando se utiliza el algoritmo PQM

```
<matcher-config>
 <mtasks>
  <mtask>
   <id>t1</id>
   <source1>
    <ref>dbf</ref>
    <info>c:\pat.dbf</info>
   </source1>
   <source2>
    <ref>sql</ref>
    <info>region</info>
    <params>
     <param>
      <key>conFactory</key>
      <value>foodmart</value>
     </param>
    </params>
   </source2>
```

```
      <ref>sql</ref>
      <info>region</info>
      <params>
       <param>
        <key>conFactory</key>
        <value>foodmart</value>
       </param>
      </params>
     </source2>
     <matcherId>munkres</matcherId>
     <functions>
      <function>
       <id>f1</id>
       <ref>individualColumnScorer</ref>
       <columnMapping>
        <columnMap>
         <ref>jaroWinklerTFIDF</ref>
         <column1>CNTY_NAME</column1>
         <column2>sales_district</column2>
        </columnMap>
       </columnMapping>
      </function>
     </functions>
     <actions>
      <action>
       <id>a1</id>
       <ref>exportToFile</ref>
       <params>
        <param>
         <key>path</key>
         <value>c:\file-munkres.csv</value>
        </param>
       </params>
      </action>
     </actions>
     </mtask>
    </mtasks>

    <connections>
     <conFactory>
      <id>foodmart</id>
      <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
      <url>jdbc:odbc:foodmart</url>
     </conFactory>
    </connections>

</matcher-config>
```

NOTA: La única diferencia es el valor del elemento **matcherId**, en los próximos archivos se pone sólo para **pqm**.

C.1.2. Prueba 2

```
<mtask>
   <id>exp1-prueba2</id>

   <source1>
    <ref>dbf</ref>
    <info>c:\pat.dbf</info>
   </source1>

   <source2>
    <ref>sql</ref>
    <info>region</info>
    <params>
     <param>
      <key>conFactory</key>
      <value>foodmart</value>
     </param>
    </params>
   </source2>

   <matcherId>pqm</matcherId>

   <filter>
    <functions>
     <function>
      <id>f1</id>
      <ref>individualColumnScorer</ref>
      <columnMapping>
       <columnMap>
        <ref>substring</ref>
        <column1>STATE_NAME</column1>
        <column2>sales_state_province</column2>
       </columnMap>
       <columnMap>
        <ref>substring</ref>
        <column1>STATE_NAME</column1>
        <column2>sales_country</column2>
        <params>
         <param>
          <key>column1_value</key>
          <value>USA</value>
         </param>
        </params>
       </columnMap>
      </columnMapping>
     </function>
    </functions>
   </filter>
```

```
  <functions>
   <function>
    <id>f1</id>
    <ref>individualColumnScorer</ref>
    <columnMapping>
     <columnMap>
      <ref>jaroWinklerTFIDF</ref>
      <column1>CNTY_NAME</column1>
      <column2>sales_district</column2>
     </columnMap>
    </columnMapping>
   </function>
  </functions>
  <actions>
   <action>
    <id>a1</id>
    <ref>exportToFile</ref>
    <params>
     <param>
      <key>path</key>
      <value>c:\file-m.csv</value>
     </param>
    </params>
   </action>
  </actions>
 </mtask>
```

Prueba 3

```
<mtask>
  <id>exp1-prueba3</id>

   <source1>
    <ref>dbf</ref>
    <info>c:\pat.dbf</info>
   </source1>

   <source2>
    <ref>sql</ref>
    <info>region</info>
    <params>
     <param>
      <key>conFactory</key>
      <value>foodmart</value>
     </param>
    </params>
   </source2>

   <matcherId>pqm</matcherId>
```

```
<filter>
 <functions>
  <function>
   <id>f1</id>
   <ref>individualColumnScorer</ref>
   <columnMapping>
    <columnMap>
     <ref>substring</ref>
     <column1>STATE_NAME</column1>
     <column2>sales_state_province</column2>
    </columnMap>
    <columnMap>
     <ref>substring</ref>
     <column1>STATE_NAME</column1>
     <column2>sales_country</column2>
     <params>
      <param>
       <key>column1_value</key>
       <value>USA</value>
      </param>
     </params>
    </columnMap>
   </columnMapping>
  </function>
 </functions>
</filter>


<functions>
 <function>
  <id>f1</id>
  <ref>individualColumnScorer</ref>
  <columnMapping>
   <columnMap>
    <ref>jaroWinklerTFIDF</ref>
    <column1>CNTY_NAME</column1>
    <column2>sales_district</column2>
   </columnMap>
   <columnMap>
    <ref>slim</ref>
    <column1>STATE_NAME</column1>
    <column2>sales_state_province</column2>
    <params>
     <param>
      <key>minScore</key>
      <value>0.6</value>
     </param>
    </params>
   </columnMap>
```

```
      </columnMapping>
      <params>
       <param>
        <key>weight_1</key>
        <value>0.7</value>
       </param>
      </params>
     </function>
    </functions>
    <actions>
     <action>
      <id>a1</id>
      <ref>exportToFile</ref>
      <params>
       <param>
        <key>path</key>
        <value>c:\file-m.csv</value>
       </param>
       <param>
        <key>delimiter</key>
        <value>;</value>
       </param>
      </params>
     </action>
    </actions>
   </mtask>
```

C.2. Experimento 2

C.2.1. Prueba 1

```
<?xml version="1.0" encoding="UTF-8"?>
<matcher-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task2.xsd">
 <mtasks>

  <mtask>
     <id>exp2-prueba1</id>
   <source1>
    <ref>dbf</ref>
    <info>c:\citiesx020.dbf</info>
   </source1>
   <source2>
    <ref>sql</ref>
    <info>region</info>
    <params>
     <param>
      <key>conFactory</key>
      <value>foodmart</value>
     </param>
```

```
 </params>
</source2>

<matcherId>pqm</matcherId>


<filter>
 <functions>
  <function>
   <id>f1</id>
   <ref>individualColumnScorer</ref>
   <columnMapping>
    <columnMap>
     <ref>substring</ref>
     <column1>STATE</column1>
     <column2>sales_state_province</column2>
    </columnMap>
    <columnMap>
     <ref>substring</ref>
     <column1>STATE</column1>
     <column2>sales_country</column2>
     <params>
      <param>
       <key>column1_value</key>
       <value>USA</value>
      </param>
     </params>
    </columnMap>
   </columnMapping>
  </function>
 </functions>
</filter>

<functions>
 <function>
  <id>f1</id>
  <ref>individualColumnScorer</ref>
  <columnMapping>
   <columnMap>
    <ref>jaroWinklerTFIDF</ref>
    <column1>COUNTY</column1>
    <column2>sales_district</column2>
   </columnMap>
   <columnMap>
    <ref>jaroWinkler</ref>
    <column1>STATE</column1>
    <column2>sales_state_province</column2>
   </columnMap>
  </columnMapping>
 </function>
```

```
    </functions>
    <actions>
     <action>
      <id>a1</id>
      <ref>exportToFile</ref>
      <params>
       <param>
        <key>path</key>
        <value>c:\file-E2-p1.csv</value>
       </param>
       <param>
        <key>delimiter</key>
        <value>;</value>
       </param>
      </params>
     </action>
    </actions>
   </mtask>

  </mtasks>

  <connections>
   <conFactory>
    <id>foodmart</id>
    <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
    <url>jdbc:odbc:foodmart</url>
   </conFactory>
  </connections>

</matcher-config>
```

C.2.2. Prueba 2

```
<?xml version="1.0" encoding="UTF-8"?>
<matcher-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task2.xsd">
 <mtasks>

  <mtask>
     <id>exp2-prueba1</id>
    <source1>
     <ref>dbf</ref>
     <info>c:\citiesx020.dbf</info>
    </source1>
    <source2>
     <ref>sql</ref>
     <info>region</info>
     <params>
      <param>
```

```
    <key>conFactory</key>
    <value>foodmart</value>
   </param>
  </params>
 </source2>
 <matcherId>pqm</matcherId>


 <filter>
  <functions>
   <function>
    <id>f1</id>
    <ref>individualColumnScorer</ref>
    <columnMapping>
     <columnMap>
      <ref>substring</ref>
      <column1>STATE</column1>
      <column2>sales_state_province</column2>
     </columnMap>
    </columnMapping>
   </function>
  </functions>
 </filter>


 <functions>
  <function>
   <id>f1</id>
   <ref>individualColumnScorer</ref>
   <columnMapping>
    <columnMap>
     <ref>jaroWinklerTFIDF</ref>
     <column1>COUNTY</column1>
     <column2>sales_district</column2>
     <params>
      <param>
       <key>minScore</key>
       <value>0.5</value>
      </param>
     </params>
    </columnMap>
    <columnMap>
     <ref>jaroWinkler</ref>
     <column1>STATE</column1>
     <column2>sales_state_province</column2>
    </columnMap>
   </columnMapping>
  </function>

 </functions>
```


ht

```
    <actions>
     <action>
      <id>a1</id>
      <ref>exportToFile</ref>
      <params>
       <param>
        <key>path</key>
        <value>c:\file-E2-p1.csv</value>
       </param>
       <param>
        <key>delimiter</key>
        <value>;</value>
       </param>
      </params>
     </action>
    </actions>
   </mtask>

  </mtasks>

  <connections>
   <conFactory>
    <id>foodmart</id>
    <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
    <url>jdbc:odbc:foodmart</url>
   </conFactory>
  </connections>

</matcher-config>
```

C.3. Experimento 3

C.3.1. Prueba 1

```
<?xml version="1.0" encoding="UTF-8"?>
<matcher-config>
 <mtasks>
  <mtask>
   <id>exp3-prueba1</id>

   <source1>
    <ref>dbf</ref>
    <info>c:\citiesx020.dbf</info>
   </source1>
   <source2>
    <ref>dbf</ref>
    <info>c:\PAT.dbf</info>
   </source2>
   <matcherId>pqm</matcherId>
```

```
<filter>
 <functions>
  <function>
   <id>f1</id>
   <ref>individualColumnScorer</ref>
   <columnMapping>
    <columnMap>
     <ref>substring</ref>
     <column1>STATE</column1>
     <column2>STATE_NAME</column2>
    </columnMap>
    <columnMap>
     <ref>substring</ref>
     <column1>COUNTY</column1>
     <column2>CNTY_NAME</column2>
     <params>
      <param>
       <key>endIndex</key>
       <value>2</value>
      </param>
     </params>
    </columnMap>
   </columnMapping>
  </function>
 </functions>
</filter>
```

```
<functions>
 <function>
  <id>f1</id>
  <ref>individualColumnScorer</ref>
  <columnMapping>
   <columnMap>
    <ref>jaroWinklerTFIDF</ref>
    <column1>COUNTY</column1>
    <column2>CNTY_NAME</column2>
    <params>
     <param>
      <key>minScore</key>
      <value>0.5</value>
     </param>
    </params>
   </columnMap>
   <columnMap>
    <ref>slim</ref>
    <column1>STATE</column1>
    <column2>STATE_NAME</column2>
    <params>
     <param>
```

```
          <key>minScore</key>
          <value>0.5</value>
         </param>
        </params>
       </columnMap>
      </columnMapping>
      <params>
       <param>
        <key>minScore</key>
        <value>0.5</value>
       </param>
      </params>
     </function>

    </functions>
    <actions>
     <action>
      <id>a1</id>
      <ref>exportToFile</ref>
      <params>
       <param>
        <key>path</key>
        <value>c:\file-E3-p1-munkres.csv</value>
       </param>
       <param>
        <key>delimiter</key>
        <value>;</value>
       </param>
      </params>
     </action>
    </actions>
   </mtask>
  </mtasks>
</matcher-config>
```

D

DDT

Apéndice D

Manual de referencia de RE-SPaM++

D.1. Arquitectura

La solución propuesta está construida en tres módulos. Estos módulos y relaciones se muestran en la figura D.1.

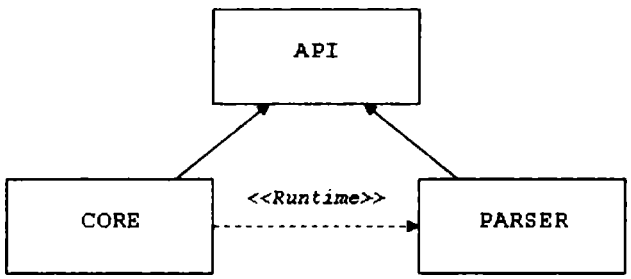


Figura D.1: Módulos que componen a la implementación de RE-SPaM++

Cada bloque es un archivo JAR[jar] y las líneas son las dependencias. La línea punteada significa que no es nesecaria para compilar la librería, sino que es necesaria que solamente en tiempo de ejecución.

D.2. API

El API consta de una serie de interfaces Java que se describirán a continuación.

D.2.1. Query

Interfaz común de las consultas.

```
public interface Query {
```

```
public String getQuery();
}
```

D.2.2. PietQLQuery

Interfaz de las consultas PietQL.

```
public interface PietQLQuery extends Query {
public Object execute();
}
```

D.2.3. RSPaMQuery

Interfaz de las consultas que se ejecutan en el entorno RE-SPaM.

```
public interface RSPaMQuery extends Query{}
```

D.2.4. ERSPaMQuery

Interfaz de las consultas RE-SPaM++. El método **getRSPaMQuery()** retorna la consulta RE-SPaM embebida.

```
public interface ERSPaMQuery extends RSPaMQuery{
        public Set<TemporalTable> getTables();
        public TemporalTable getTable(String id);
        public RSPaMQuery getRSPaMQuery();
        public Object execute();
        public void setMiningParameters(Map<String, Object> params);
        public Map<String, Object> getMiningParameters();
        public void setMiningParameter(String key, Object value);
        public Object getMiningParameter(String key);
}
```

D.2.5. TemporalTable

Interfaz de las consultas PietQL embebidas en las consultas RE-SPaM++. El método `getId()` retorna el nombre asociado, el método `runAssociatedQuery()` es un atajo a `getQuery().execute()` y `getAttributes()` retorna los atributos definidos.

```
public interface TemporalTable {

        public PietQLQuery getQuery();

        public String getId();

        public Object runAssociatedQuery();

        public Set<Attribute> getAttributes();
}
```

D.2.6. Session

La unidad de trabajo en RE-SPaM++. Permite construir consultas.

```
public interface Session extends Serializable {

        public PietQLQuery createPietQLQuery(String queryString);

        public ERSPaMQuery createERSPaMQuery(String queryString);

        public void close();

        public boolean isClosed();

}
```

D.2.7. SessionFactory

Clase que crea sesiones en base a la configuración dada. El método `openSession()` crea nueva una instancia de `Session`.

```
public interface SessionFactory extends Serializable  {

public Session openSession();

        public Session openSession(Object data);

        public void close();

        public boolean isClosed();

}
```

D.2.8. Configuration

Interfaz encargada de construir una nueva instancia de `SessionFactory`. Antes de invocar al método `buildSessionFactory()` se debe ejecutar `configure()`.

```
public interface Configuration {

   public SessionFactory buildSessionFactory();

        public Configuration configure();

        public Configuration configure(Settings settings);
}
```

D.2.9. Settings

Interfaz que se utiliza para configurar las librerías a la cual se delega la ejecución de las consultas.

```
public interface Settings {

        public Properties getPieqQlProperties();

        public Properties getRSPAMProperties();
}
```

D.2.10. ConfigurationFactory

Punto de entrada a RE-SPaM++. Es la clase encargada de crear una instancia que implemente `Configuration`. En las siguientes secciones se profundizará el funcionamiento.

```
public class ConfigurationFactory {

public static Configuration getConfiguration()

}
```

D.2.11. ParserFacade

Interfaz que debe ser provista por el módulo *Parser* que se describirá en la siguiente sección.

```
public interface ParserFacade extends Serializable{

        public PietQLQuery parsePietQL(String queryString);

        public ERSPaMQuery parseERSPaM(String queryString);
```

```
        public RSPaMQuery parseRSPaM(String queryString);
}
```

D.3. Parser

Este módulo es el encargado de realizar los parseos de los lenguajes y proveer una implementación de **ParserFacade**.

D.3.1. Gramáticas

Como se definió en la sección D.1, este módulo debe re-usar los parser de las otras gramáticas[1], pero no se pudo cumplir en su totalidad debido a:

Dependencias Los parsers de cada una de las gramáticas contienen dependencias a internas propieas de cada implementación.

Validaciones semánticas Los parsers realizan validaciones que dependen de cierta configuración que es realizada en la implementación. Por ejemplo:

1. En el parser incluído en PietQL chequea que las funciones definidas en el where[2] estén definidas en un archivo. No se puede incluir ese archivo de configuración dentro de esta librería porque no se puede asegurar que no se vayan a quitar ó agregar funciones.
2. En el parser incluído en RE-SPaM valida que las jerarquías y los atributos de las mismas en base a un archivo de configuración. El parser provisto no debe realizar estos chequeos por los mismos motivos descriptos en el item anterior.

Finalmente, por los motivos descriptos, se tomaron las siguientes medidas:

1. Se valida la solamente la sintaxis del lenguaje piet-ql, y si es correcta se valida nuevamente al momento de ejecutar en la librería de piet-ql.
2. El lenguaje respam se acepta en crudo y se delega en su totalidad el parseo al parser de la librería respam.

D.3.2. Construcción de los parser

Los parser están construidos con JavaCC[jcc]. Esta potente librería permite embeber parsers lo que permite una buena modularización.

Las gramáticas están en la carpeta **src/grammars**.

Nota El parser de PietQL al ser embebido necesita que se defina un símbolo que indique que deje de parsear, comunmente es el EOF. En este caso el EOF es el carácter ;. Esta decisión no debe interferire si se invoca al método **parsePietQL(String queryString)** de la interfaz **ParserFacade**. Por lo tanto en la implementación se añade el carácter ; si no está.

[1] La gramática PietQL y RE-SPaM
[2] Por ejemplo **crosses**

D.4. Core

Este módulo provee la implementación de las interfaces del módulo API[3] y el modelo de configuración.

D.4.1. Utilización

Para utilizar esta librería debe incluir en el classpath de la aplicación los siguientes Jar:

1. **erespam-api.jar**
2. **erespam-parser.jar**
3. **erespam-core.jar**

y sus dependencias:

```
ar.edu.unlp:RE-SPaM++-core:jar:0.1
+- ar.edu.unlp:RE-SPaM++-api:jar:0.1:compile
\- ar.edu.unlp:RE-SPaM++-parser:jar:0.1:runtime
+- commons-logging:commons-logging:jar:1.1.1:compile
+- commons-configuration:commons-configuration:jar:1.6:compile
|  +- commons-collections:commons-collections:jar:3.2.1:compile
|  +- commons-lang:commons-lang:jar:2.4:compile
|  +- commons-digester:commons-digester:jar:1.8:compile
|  |  \- commons-beanutils:commons-beanutils:jar:1.7.0:compile
|  \- commons-beanutils:commons-beanutils-core:jar:1.8.0:compile
+- libpql:provided
<<DEPENDENCIAS DE PIET-QL>>
<<DEPENDENCIAS DE RE-SPAM>>
```

Si se utiliza Maven[mvn] incluir solamente la dependencia:

```
<dependency>
 <groupId>ar.edu.unlp</groupId>
 <artifactId>erespam-core</artifactId>
 <version>...</version>
 <scope>runtime</scope>
</dependency>
```

D.4.2. Configuración

En esta sección se detallan como configurar los adaptadores a las librerías PietQL y RE-SPaM.

Binders

Los adaptadores se denominarán de ahora mas **Binders**. La implementación delega a los binders la responsabilidad del parseo final y ejecución de las consultas. La definición de los mismos se encuentra en la Figura 8.2.

[3]Menos **ParserFacade** que se implementa en el módulo Parser.

Definición de los binders

Los binders están configurados en el archivo

`/ar/edu/unlp/RE-SPaM++/erspam-binders.properties`

que se encuentra en el archivo `erspam-core.jar`.

```
erspam-binders.properties
-------------------------
ar.edu.unlp.erespam.binders.PietQLBinder=...
ar.edu.unlp.erespam.binders.RE-SPaMBinder=...
```

El binder asociado a PietQL es

`ar.edu.unlp.erespam.binders.defaults.PietQLBinderDefault`

el mismo es totalmente funcional, pero el asociado a RE-SPaM[4] no[5]. Para modificar los binders hay que incorporar un archivo `properties` con el nombre `user-binders.properties`[6]. Por ejemplo si se quiere que se utilize otro binder para RE-SPaM el archivo debería ser:

```
user-binders.properties
-------------------------
ar.edu.unlp.erespam.binders.RE-SPaMBinder=foo.MiBinder
```

Notar que no es necesario indicar cual es el binder de PietQL porque si no lo encuentra en `user-binders.properties` busca el que se definió en `erspam-binders.properties`

[4] `ar.edu.unlp.erespam.binders.defaults.RSPaMBinderDefault`

[5] Esto fue porque al momento del desarrollo no se contaba con la librería RE-SPaM

[6] En la Sección 8.4.3 se muestra otra forma de realizar lo mismo

Apéndice E

Sintaxis de PietQL, RE-SPaM y RE-SPaM++

En el siguiente apéndice se describen las grámaticas mediante Extended BackusNaur Form (E-BNF) de los lenguajes PietQL, RE-SPaM y RE-SPaM++.

E.1. Sintaxis de PietQL

En el cuadro E.1 se describe la sintaxis de PietQL.
Para más información y ejemplos puede consultar [GVZ08].

E.2. Sintaxis de RE-SPaM

En el Cuadro E.2 se detalla la sintaxis del lenguaje RE-SPaM.

E.3. Sintaxis de RE-SPaM++

El lenguaje RE-SPaM trabaja sobre elementos geómetricos y el lenguaje PietQL retorna elementos de un cubo u objetos geométricos. Esto significa que de las cuatro tipos de consultas sólo se podrán utilizar GIS y GIS-OLAP, donde ambas retornan objetos geométricos[1].

Sea `piet-qlLite` la sintaxis que retorna GIS de PietQL y RE-SPaM de RE-SPaM definida en la Sección E.2. La sintaxis de RE-SPaM++ se describe en el Cuadro E.3:

[1] En la Sección 6.3 se puede ver en detalle los tipos de consultas de PietQL

```
query_ ::= SELECT q2_
q2_ ::= GIS gislist_ fromclause_g_ [whereclause_g_]
q2_ ::= CUBE olaplist_ ON COLUMNS olaplist_
      ON ROWS fromclause_o_ [whereclause_o_] [sliceclause_]
q2_ ::= CUBE olaplist_ ON ROWS olaplist_
      ON COLUMNS fromclause_o_ [whereclause_o_] [sliceclause_]
q2_ ::=CUBE olaplist_ON ROWS fromclause_o_[whereclause_o_][sliceclause_]
q2_ ::= CUBE olaplist_ON COLUMNS fromclause_o_ [whereclause_o_]
        [sliceclause_]
single_result_subquery_ ::= SELECT q3_
q3_ ::= GIS attribute_from_g_ fromclause_g_ [whereclause_g_]
q3_ ::= CUBE attribute_path_o_ fromclause_o_ [whereclause_o_]
        [sliceclause_]
gislist_ ::= attribute_path_g_ [, gislist_]
attribute_path_g_ ::= attribute_g_.attribute_path_g_
attribute_g_ ::= GIS_ATTRIBUTE | FUNCTION_G
olaplist_ ::= attribute_path_o_ [, olaplist_]
attribute_path_o_ ::= attribute_o_.attribute_path_o_
attribute_o_ ::= OLAP_MEMBER | OLAP_LEVEL$ | FUNCTION_O
fromclause_g_ ::= FROM layerslist_
layerslist_ ::= attribute_from_g_ [, layerslist_]
attribute_from_g_ ::= GIS_LAYER
fromclause_o_ ::= FROM attribute_from_o_
attribute_from_o_ ::= OLAP_CUBE
whereclause_g_ ::= WHERE gis_filter_
gis_filter_ ::= gis_predicate_ AND gis_filter_
gis_filter_ ::= gis_predicate_ OR gis_filter_
gis_filter_ ::= NOT(gis_filter_)
gis_filter_ ::= (gis_filter_)
whereclause_o_ ::= WHERE olap_filter_
olap_filter_ ::= olap_predicate_
olap_filter_ ::= olap_predicate_ OR olap_filter_
olap_filter_ ::= NOT(olap_filter_)
olap_filter_ ::= (olap_filter_)
sliceclause_ ::= SLICE slicefilterlist_
slicefilterlist_ ::=attribute_path_ [, slicefilterlist_]
gis_predicate_ ::= attribute_path_g_ IN (single_result_subquery_)
gis_predicate_ ::= GIS_PREDICATE
olap_predicate_ ::= attribute_path_o_ IN (single_result_subquery_)
```

Cuadro E.1: Sintaxis de PietQL.

```
respam ::= reg_exp
reg_exp ::= atom [ kleene_plus | kleene | optional ]
            [ ( concatenation | union )  reg_exp ]
atom ::= constraint | "(" reg_exp ")"
kleene ::= "*"
kleenePlus ::= "+"
optional ::= "?"
concatenation ::= "."
union ::= "|"
contraint ::= "[" condition "]"
condition ::= equality [ "^" condition ]
equality ::= function "=" value  | attribute "=" value
value := literal | variable
attribute ::=  common_attribute | date_attribute |
               time_attribute   |  timestamp_attribute
variable ::= "@" identificador
literal ::= "'" carcteres "'"
common_attribute ::= ("A"-"Z" |"a"-"z")+
date_attribute ::= "ts\_date" | "tf\_date"
time_attribute ::= "ts\_time" | "tf\_time"
timestamp_attribute ::= "ts" | "tf"
function ::= identificador (parametros)
```

Cuadro E.2: Sintaxis de RE-SPaM.

```
e-RE-SPaM :=
 ["WITH" "TABLE" name (attribute_name [, attribute_name]*) "AS"
  piet-qlLite ";"]* respam

piet-qlLite := SELECT q2_
q2_ := GIS gislist_ fromclause_g_ [whereclause_g_]
single_result_subquery_ := SELECT q3_
q3_ := GIS attribute_from_g_ fromclause_g_ [whereclause_g_]
q3_ = CUBE attribute_path_o_ fromclause_o_ [whereclause_o_] [sliceclause_]
gislist_ := attribute_path_g_ [, gislist_]
attribute_path_g_ := attribute_g_.attribute_path_g_
attribute_g_ := GIS_ATTRIBUTE | FUNCTION_G
olaplist_ := attribute_path_o_ [, olaplist_]
attribute_path_o_ := attribute_o_.attribute_path_o_
attribute_o_ := OLAP_MEMBER | OLAP_LEVEL$ | FUNCTION_O
fromclause_g_ := FROM layerslist_
layerslist_ := attribute_from_g_ [, layerslist_]
attribute_from_g_ := GIS_LAYER
fromclause_o_ := FROM attribute_from_o_
attribute_from_o_ := OLAP_CUBE
whereclause_g_ := WHERE gis_filter_
gis_filter_ := gis_predicate_ AND gis_filter_
gis_filter_ := gis_predicate_ OR gis_filter_
gis_filter_ := NOT(gis_filter_)
gis_filter_ := (gis_filter_)
whereclause_o_ := WHERE olap_filter_
olap_filter_ := olap_predicate_
olap_filter_ := olap_predicate_ OR olap_filter_
olap_filter_ := NOT(olap_filter_)
olap_filter_ := (olap_filter_)
sliceclause_ := SLICE slicefilterlist_
slicefilterlist_ :=attribute_path_ [, slicefilterlist_]
gis_predicate_ := attribute_path_g_ IN (single_result_subquery_)
gis_predicate_ := GIS_PREDICATE
olap_predicate_ := attribute_path_o_ IN (single_result_subquery_)
```

Cuadro E.3: Sintaxis de RE-SPaM++.

Apéndice F

Análisis de memoria

Los tiempos de ejecución del algoritmo Húngaro (Munkres) son demasiado lentos, por ejemplo 5000vs5000 tarda 24hs. Se puede seguir optimizando las estructuras de datos para minimizar aún más la escritura a disco (tanto por parte de código ó por paginación de memoria a memoria virtual).

Vamos a realizar un análisis en cuanto a las estructuras de datos que se utilizan. Para dar una idea, comparar 20000vs20000 actualmente necesita **6,5Gb** de memoria, pero como veremos en este análisis, en algunos casos, se podría reducir a **560Mb**.

F.1. Lectura de las fuentes de datos

En primer lugar en el archivo de configuración se definen las dos fuentes de datos, en este caso, una consulta a una BD. Esos registros en su totalidad son cargados a memoria.

Suponer que *records*1 son N registros y *records*2 son M registros.

Para saber la memoria ocupada tenemos que calcular cuanto ocupa cada elemento de records1 y de records2 y multiplicarlos por N y M respectivamente.

La estructura donde se almacenan los registros es un HashSet. La estructura almacena objetos del tipo Record. Cada puntero a objeto en Java es de 12 bytes. Luego:

TM(X) = Tamaño en memoria de X

TM(records1) = N*(12+TM(Record)) = N*12

Simplificando, si tenemos que N=20000, TM(records1) = 240Kb

Podríamos decir que es despreciable la memoria utilizada por las colecciones en Java que almacenan los registros y asumir que es la estructura más conveniente y a su vez más simple de utilizar.

F.2. Aplicando las funciones de comparación

Una vez leídos los registros se procede a aplicar las funciones de comparación y guardar los resultados para su posterior procesamiento.

Para meternos en contexto, el proceso que resuelve el match, que implementa `IMatcher` tiene el siguiente método:

```
public abstract List<IPair<Record>> matching
  (
   ArrayList<Record> records1 ,
   ArrayList<Record> records2 ,
   RecordFunction score ,
   Result result
  )
```

Donde *score* es la función de comparación y *result* es el objeto donde se van a ir almacenando los resultados.

La decisión de optar por `ArrayList` para pasar los registros es por su lectura en $O(1)$ debido a que los datos son almacenados en back-end por un arreglo. Las comparaciones se almacenan en una matriz, siempre y cuando supere el valor de *minScore*.

```
float v = score.score(records1.get(i), records2.get(j))
if (v >= minScore)
   costos[i,j] = v
```

El valor *minScore* es definido como parámetro en la función. La matriz de costos almacena valores del tipo `float` y no `double` debido a que float ocupa 4 bytes y double 8.

El espacio en disco de la matriz de costos es:

$$TM(costos) = N * M * 4$$

Si tuviéramos que

$$s(records1) = s(records2) = 20000 \rightarrow TM(costos) = 1600000000 = 1600Mb$$

F.3. Matriz de costos

Como se aprecia, la matriz de costos utilizándola de manera nativa ocupa mucha memoria. Esta estructura para gran volumen de datos debe ser optimizada.

El uso del parámetro *minScore* es de mucha utilidad. Por ejemplo en ejemplos realizados configurándolo en 0.6, donde $s(records1) = s(records2) = 5000$ en vez de almacenar $5000 * 5000 = 25000000$ valores sólo de utilidad tenía 345952 valores. Por lo tanto información útil es del sólo del **1.3 %**.

Hasta aquí se describe el procesamiento común. Las demás estructuras son propias de cada algoritmo que implemente el matching, es decir, el proceso de resolución.

F.4. Algoritmo húngaro

La implementación utilizada está descripta en `http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html`

Vamos a analizar las estructuras utilizadas por el algoritmo húngaro. Este algoritmo es de $O(n^4)$.

El algoritmo recibe como parámetro la matriz de costos y retorna una matriz **mask** con ceros y unos. La interpretación de esta matriz es:

$$mask[i][j] = 1 \Rightarrow records1[i],\ records2[j]\ es\ un\ match$$

Las estructuras utilizadas en el algoritmo son:

costos' Una copia de la matriz de costos.

mask Una matriz de $N * M$.

path Una matriz de $N * M * 2$. Su definición en Java es: `path[N*M][2]`

La matriz **mask** almacena valores tipo byte. La utilización de byte es que una parte de el algoritmo escribe valores distintos a 0 o 1 para marcarlos. La matriz **path** almacena `int`. Esta matriz se utiliza para ir guardando índices.

Utiliza otras estructuras pero son despreciables en comparación con las matrices.

F.4.1. Cálculo de memoria utilizada

Analicemos ahora cuanto necesita de memoria para el caso de $N * M$. El cálculo lo realizaremos sumando los espacios de cada estructura utilizada. Las estructuras utilizadas son:

costos Utilizada para almacenar los costes y luego leerlos para mostrar los resultados.

mask El algoritmo húngaro está pensado para minimizar la suma total de la asignación. Nosotros queremos maximizar. Si la matriz de costes se le encuentra el valor máximo *largest* y a cada elemento hacemos

$$costos[i][j] = largest - costos[i][j]$$

Encontrar la asignación que minimice para esa matriz de costos modificada (costos') nos resuelve la asignación máxima para costos.

path Se utiliza en el paso 5 del algoritmo. Ver HungarianAlgorithm

mask matriz donde se devuelve el resultado

El tamaño requerido para resolver el matching donde $s(records1) = N$ y $s(records2) = M$ es:

$$\begin{aligned}
TM(ALL) &= TM(costos) + TM(costos') + TM(mask) + TM(path) \\
&= 2TM(costos) + TM(mask) + TM(path) \\
&= 2NM * 4 + NM + NM4 * 2 \\
&= 8NM + NM + 8NM \\
&= 17NM
\end{aligned}$$

Luego para s(records1)=s(records2)=20000

$$TM(ALL) = 17 * 20000 * 20000 = 6800000000b = 6485Mb$$

TM(costos)=TM(costos') son matrizes del tipo `float` y en Java ocupan 4 bytes. La matriz `path` es una matriz de tipo `int` que en Java ocupa 2 bytes.

F.5. Optimización

En esta sección vamos a ver posibles optimizaciones.

F.5.1. Almacenamiento en disco

El almacenamiento en disco nos salva del uso excesivo de memoria. Ya se realizadon pruebas pero el resultado en los tiempos son muy malos debido a la cantidad de loops que realiza el algoritmo húngaro lo que lleva a muchas lecturas sobre la matriz y por ende al disco.

Se descarta del uso del disco para ir guardando porciones de la matriz.

F.5.2. Naturaleza de las matrices

Una matriz podemos catalogarla con respecto a la cantidad de información útil como densa ó espaciada.

Las densas contienen mucha más información, es decir cada elemento tiene un valor asignado. En cambio las espaciadas tienen muy pocos valores asignados, la mayoría de los elementos nunca fueron asignados con un valor.

La matriz de costos con ayuda del parámetro minScore, la podemos armar como una matriz espaciada. De esta manera podemos reducir de manera drástica su espacio.

Si la matriz de costos es espaciada recalculemos el nuevo espacio total necesario. Se calcula con minScore=0.8. El total asignado es sólo el 1 %.

$$\begin{aligned} TM(ALL) &= 8NM * 0{,}01 + NM + 8NM \\ &= 0{,}08NM + 9NM \\ &= 9{,}08NM \end{aligned}$$

Luego para $s(records1) = s(records2) = 20000$

$$T(ALL) = 9{,}08 * 20000 * 20000 = 3632Mb$$

Realmente como vamos a utilizar colecciones para implementar las matrices (si utilizamos colecciones estándar) son punteros a `Float`. Los punteros en Java ocun 12 bytes. Luego:

$$\begin{aligned} TM(ALL) &= (12 + 8)NM * 0{,}01 + NM + 8NM \\ &= 0{,}2NM + 9NM \\ &= 9{,}2NM \end{aligned}$$

$$T(ALL) = 9{,}2 * 20000 * 20000 = 3680Mb$$

Si analizamos en el algoritmo la matriz **path** por fila y columna sólo un dato es útil. Por ello es una matriz espaciada. La matriz **mask no** es espaciada.

$$\begin{aligned} TM(ALL) &= 8NM * 0{,}01 + NM + 8NM * 0{,}01 \\ &= 0{,}08NM + NM + 0{,}08NM \\ &= 1{,}16NM \end{aligned}$$

$$T(ALL) = 1{,}28 * 20000 * 20000 = 448Mb$$

Si utilizamos colecciones estándar para implementarlo:

$$\begin{aligned} TM(ALL) &= (12 + 8)NM * 0{,}01 + NM + (12 + 8)NM * 0{,}01 \\ &= 0{,}2NM + NM + 0{,}2NM \\ &= 1{,}4NM \end{aligned}$$

$$T(ALL) = 1{,}4 * 20000 * 20000 = 560Mb$$

Hemos analizado y asumiendo que utilizen minScore adecuado y la función de comparación retorna valores adecuados (por ejemplo que no retorne siempre 1) y no se están comparando tablas donde todos sus datos son iguales se ha reducido el espacio necesario de 6485Mb a 560Mb, un total del 91,36 % menos.

F.6. Conclusión

Como se demuestra en este apartado, es muy performante la utilización de matrices espaciadas cuando se está utilizando el algoritmo Húngaro con muchos datos.
En casos que no haya demasiados datos es más eficiente la utilización de arreglos nativos del lenguaje. Por lo tanto para pocos datos se debe utilizar arreglos nativos y para mmuchos datos matrices espaciadas.

Apéndice G

Descripción de los archivos utilizados en el caso práctico

G.1. paises1

id	nombre
3010	AFGANISTAN
4010	ALBANIA
4380	ALEMANIA
4040	ANDORRA
1490	ANGOLA
2271	ANGUILA
2370	ANTIGUA Y BARBUDA
2302	ANTILLAS
3020	ARABIA SAUDITA
1020	ARGELIA
2000	ARGENTINA
3490	ARMENIA
2301	ARUBA
1451	ASCENSION
5010	AUSTRALIA
4050	AUSTRIA
3500	AZERBAIDZHAN
2390	BAHAMAS
3030	BAHREIN
3450	BANGLADESH
2010	BARBADOS
4060	BELGICA
2360	BELICE
1120	BENIN
2272	BERMUDAS
4390	BIELORUS

2020	BOLIVIA
1030	BOTSWANA
5224	BRASIL - (CEARA)
5225	BRASIL - (GOIAS)
5218	BRASIL - (MINAS GERAIS)
5220	BRASIL - (RIO GRANDE DEL NORTE)
5219	BRASIL - (SAN PABLO)
2030	BRASIL
212121	brasill
3460	BRUNEI
4070	BULGARIA
1010	BURKINA FASO
1040	BURUNDI
3050	BUTAN
1500	CABO VERDE
2273	CAIMAN
3060	CAMBOYA (EX KAMPUCHEA)
1050	CAMERUN
2040	CANADA
1461	CANARIAS
1462	CEUTA Y MELILLA
1110	CHAD
4080	CHECOSLOVAQUIA
2080	CHILE
3100	CHINA
4350	CHIPRE
2050	COLOMBIA
1550	COMORO
1080	CONGO
3080	COREA
3090	COREA (N O U S A R)
1100	COSTA DE MARFIL
2060	COSTA RICA
4470	CROACIA
2070	CUBA
1454	DEL OCEANO INDICO
4090	DINAMARCA
1530	DJIBOUTI
2330	DOMINICA
2100	ECUADOR
1130	EGIPTO
2110	EL SALVADOR
4480	ELOVAQUIA
3310	EMIRATOS ARABES UNIDOS

5217	ESCOCIA
4490	ESLOVENIA
4100	ESPAA
2120	ESTADOS UNIDOS
4400	ESTONIA
1140	ETIOPIA
5120	FIJI
3120	FILIPINAS
4110	FINLANDIA
4120	FRANCIA
1150	GABON
5221	GALES (REINO UNIDO)
1160	GAMBIA
3140	GAZA
3510	GEORGIA
1170	GHANA
4331	GIBRALTAR
4130	GRECIA
2400	GRENADA
2281	GROENLANDIA
2291	GUADALUPE
5111	GUAM
2130	GUATEMALA
2292	GUAYANA
1180	GUINEA
1560	GUINEA BISSAU
1190	GUINEA ECUATORIAL
2140	GUYANA
2150	HAITI
5118	HAWAI(ESTADO DE ESTADOS UNIDOS)
4230	HOLANDA
2160	HONDURAS
3411	HONG KONG
5116	HOWLAND
4140	HUNGRIA
3150	INDIA
3160	INDONESIA
3180	IRAN
3170	IRAQ
4150	IRLANDA
4160	ISLANDIA
5214	ISLAS COMORES
5101	ISLAS COOK
5071	ISLAS DEL MAR DE CORAL

5210	ISLAS MARIANAS
5200	ISLAS MARSHALL
5081	ISLAS PITCAIRN
5180	ISLAS SALOMON
5215	ISLAS VIRGENES (ESTADO ASOCIADO)
3190	ISRAEL
4170	ITALIA
2170	JAMAICA
3200	JAPON
5115	JOHNSTON
3210	JORDANIA
3520	KAZAJSTAN
1200	KENYA
5117	KINGMAN REEF
3530	KIRGUISTAN
5140	KIRIBATI
3230	KUWAIT
3240	LAO
1210	LESOTHO
4410	LETONIA
3250	LIBANO
1220	LIBERIA
1230	LIBIA
4180	LIECHTENSTEIN
4420	LITUANIA
4190	LUXEMBURGO
3441	MACAO
5213	MACEDONIA
5073	MACQUARIE
1240	MADAGASCAR
3260	MALASIA
1250	MALAWI
3270	MALDIVAS
1260	MALI
4200	MALTA
1270	MARRUECOS
2293	MARTINICA
1280	MAURICIO
1290	MAURITANIA
1471	MAYOTTE
2180	MEXICO
5150	MICRONESIA ESTADOS FEDERADOS
5113	MIDWAY
4430	MOLDOVA

4210	MONACO
3290	MONGOLIA
5223	MONTENEGRO
2276	MONTSERRAT
1510	MOZAMBIQUE
3040	MYANMAR (EX BIRMANIA) UNION DE
1580	NAMIBIA
5030	NAURU
3300	NEPAL
2190	NICARAGUA
1300	NIGER
1310	NIGERIA
5102	NIVE
5072	NORFOLK
4220	NORUEGA
5092	NUEVA CALEDONIA
5040	NUEVA ZELANDA
3280	OMAN
5160	PALAU
2200	PANAMA
5130	PAPUA NUEVA GUINEA
3320	PAQUISTAN
2210	PARAGUAY
5226	PARAGUAY (CONCEPCION)
2220	PERU
5091	POLINESIA FRANCESA
4240	POLONIA
4250	PORTUGAL
2230	PUERTO RICO (ESTADO ASOCIADO)
3220	QATAR
4260	REINO UNIDO
5212	REINO UNIDO (INGLATERRA)
2090	REPBLICA DOMINICANA
1070	REPUBLICA CENTROAFRICANA
4510	REPUBLICA CHECA
1472	REUNION
4270	RUMANIA
4440	RUSIA
1330	RWANDA
2296	SAINT MARTIN
5112	SAMOA
5060	SAMOA OCCIDENTAL
2380	SAN CRISTOBAL Y NEVIS
4280	SAN MARINO

2295	SAN PEDRO Y MIQUELON
2350	SAN VICENTE Y LAS GRANADINAS
1452	SANTA ELENA
2340	SANTA LUCIA
4310	SANTA SEDE (VATICANO)
5211	SANTO DOMINGO
1570	SANTO TOME Y PRINCIPE
1340	SENEGAL
5222	SERBIA
5216	SERBIA Y MONTENEGRO (NO USAR)
1520	SEYCHELLES
1350	SIERRA LEONA
3330	SINGAPUR
3340	SIRIA
1360	SOMALIA
3070	SRI LANKA
1590	SUDAFRICA
1380	SUDAN
4290	SUECIA
4300	SUIZA
2320	SURINAME
1370	SWAZILANDIA
3350	TAILANDIA
3130	TAIWAN
1390	TANZANIA
3540	TAYIKISTAN
5070	TERRITORIOS VINCULADOS A AUSTRALIA
2280	TERRITORIOS VINCULADOS A DINAMARCA
1460	TERRITORIOS VINCULADOS A ESPAA
2310	TERRITORIOS VINCULADOS A ESTADOS UNIDOS
5110	TERRITORIOS VINCULADOS A ESTADOS UNIDOS
1470	TERRITORIOS VINCULADOS A FRANCIA
5090	TERRITORIOS VINCULADOS A FRANCIA
2290	TERRITORIOS VINCULADOS A FRANCIA
2300	TERRITORIOS VINCULADOS A LOS PAISES BAJOS
5100	TERRITORIOS VINCULADOS A NUEVA ZELANDA
3440	TERRITORIOS VINCULADOS A PORTUGAL
3410	TERRITORIOS VINCULADOS AL REINO UNIDO
2270	TERRITORIOS VINCULADOS AL REINO UNIDO
5080	TERRITORIOS VINCULADOS AL REINO UNIDO
4330	TERRITORIOS VINCULADOS AL REINO UNIDO
1450	TERRITORIOS VINCULADOS AL REINO UNIDO
1400	TOGO
5103	TOKELAU

5190	TONGA
2240	TRINIDAD Y TOBAGO
1453	TRISTAN DA CUNHA
1410	TUNEZ
2274	TURCAS Y CAICOS
3550	TURKMENISTAN
4360	TURQUIA
5170	TUVALU
4450	UCRANIA
1420	UGANDA
2250	URUGUAY
3560	UZBEKISTAN
5050	VANUATU
2260	VENEZUELA
3370	VIETNAM
2311	VIRGENES
2275	VIRGENES
5114	WAKE
5093	WILLIS Y FUTUNA
3480	YEMEN
4320	YUGOSLAVIA (NO USAR - USAR SERBIA Y MONTE)
1090	ZAIRE
1440	ZAMBIA
1320	ZIMBABWE

G.2. paises2

id	nombre
1	Afghanistán
2	Albania
81	Alemania
5	Andorra
6	Angola
202	Anguilla
7	Antigua y Barbuda
157	Antillas Holandesas
208	Arabia Saudi
183	Arch. Bismarck
3	Argelia
17	Armenia
162	Aruba
10	Australia
13	Austria
8	Azerbaidzhán
191	Azores
14	Bahamas
15	Bahrain

16	Bangladesh
18	Barbados
19	Bélgica
26	Bélice (Honduras Británica)
56	Benín (Dahomey)
20	Bermuda
21	Bhután
33	Bielorrusia
22	Bolivia
161	Bonaire
23	Bosnia Herzegovina
24	Botswana
68	Bougainville
25	Brasil
29	Brunei
30	Bulgaria
258	Burkina Faso(A.Volta
32	Burundi
37	Cabo Verde
34	Camboya
35	Camerún
36	Canadá
85	Canton y Enderbury
93	Carriacou
42	Chile
43	China
54	Chipre
45	Colombia
48	Congo
49	Congo
120	Corea del Norte
121	Corea del Sur
114	Costa de Marfil
51	Costa Rica
52	Croacia
53	Cuba
149	Curaao
57	Dinamarca
76	Djibouti
58	Dominica
236	Dubai
60	Ecuador
246	Egipto
61	El Salvador

235	Emiratos Arabes Unidos
64	Eritrea
216	Eslovenia
220	Espaa
252	Estados Unidos
65	Estonia
63	Etiopía
69	Fiji
186	Filipinas
70	Finlandia
72	Francia
289	Franja de Gaza
77	Gabón
79	Gambia
78	Georgia
82	Ghana
83	Gibraltar
247	Gran Bretaa
92	Granada
90	Grecia
91	Groenlandia
94	Guadalupe
96	Guatemala
192	Guinea-Bissau
97	Guinea Conakry/Rep.
62	Guinea Ecuatorial
98	Guyana
74	Guyana Francesa
99	Haití
253	Hawaii
100	Heard y MacDonald
156	Holanda
101	Honduras
102	Hong Kong
103	Hungría
105	India
106	Indonesia
108	Irak
107	Irán
109	Irlanda del Sur
71	Is. Aland
221	Is. Canarias
179	Is. Carolinas
38	Is. Cayman

11	Is. Cocos
47	Is. Comoros/Mayotte
50	Is. Cook
242	Is. Ellice
66	Is. Faeroe
86	Is. Fénix
87	Is. Gilbert
95	Is. Guam
248	Is. Guernsey
249	Is. Jersey
254	Is. Johnston
88	Is. Line
190	Is. Madeira
137	Is. Maldivas
67	Is. Malvinas
250	Is. Man
174	Is. Marianas Norte
177	Is. Marshall
140	Is. Martinica
89	Is. Navidad
171	Is. Norfolk
187	Is. Pitcairn
173	Is. Reina Maud
46	Is. San Andrés
27	Is. Solomon
232	Is. Tokelau
176	Is. Truk
240	Is. Turks & Caicos
28	Is. Vírgenes Británicas
257	Is. Vírgenes USA
255	Is. Wake
262	Is. Wallis & Futuna
104	Islandia
313	Islas Feroe
111	Israel
112	Italia
115	Jamaica
116	Japón
118	Jordania
117	Kazajstán
119	Kenya
123	Kirguistán
84	Kiribati
291	Kosovo

122	Kuwait
124	Laos
126	Lesotho
127	Letonia
125	Líbano
128	Liberia
129	Libia
130	Liechtenstein
131	Lituania
132	Luxemburgo
133	Macao
245	Macedonia
134	Madagascar
136	Malasia
135	Malawi
138	Malí
139	Malta
150	Marruecos
142	Mauricio
141	Mauritania
143	México
175	Micronesia E.Federadas
256	Midway Islands
146	Moldavia
144	Mónaco
145	Mongolia
147	Montenegro
148	Montserrat
151	Mozambique
31	Myanmar (Birmania)
153	Namibia
154	Nauru
155	Nepal
167	Nicaragua
168	Níger
169	Nigeria
170	Niue
id	nombre
172	Noruega
163	Nueva Caledonia
166	Nueva Zelanda
165	Nuevas Hébridas
152	Omán
180	Pakistán

178	Palau
80	Palestina
181	Panamá
182	Papúa y Nueva Guinea
184	Paraguay
185	Perú
75	Polinesia Francesa
188	Polonia
189	Portugal
194	Puerto Rico
195	Qatar
39	Rep. Centroafricana
55	Rep. Checa
59	Rep. Dominicana
214	Rep. Eslovaca
196	Reunión
197	Rumania
198	Rusia
199	Rwanda
158	Saba
222	Sahara Occidental
263	Samoa
4	Samoa Americana
159	San Eustaquio
206	San Marino
203	Santa Lucía
207	Sao Tomé & Principe
209	Senegal
210	Serbia
211	Seychelles
212	Sierra Leona
213	Singapur
228	Siria
217	Somalía
40	Sri Lanka (Ceylán)
200	St. Helena
201	St. Kitts/Nevis
160	St. Maarten
204	St. Pierre/Miquelón
205	St. Vincent & Granad
218	Sudáfrica
223	Sudán
226	Suecia
227	Suiza

224	Surinam
290	Svalbard y Jan Mayen
225	Swazilandia
73	Tahití
44	Taiwán
229	Tajikistán
251	Tanzania
12	Tasmania
41	Tchad
230	Thailandia
193	Timor Oriental
231	Togo
233	Tonga
234	Trinidad y Tobago
237	Túnez
239	Turkmenistán
238	Turquía
241	Tuvalu
244	Ucrania
243	Uganda
110	Ulster (Irlanda Norte)
259	Uruguay
260	Uzbekistán
164	Vanuatu
113	Vaticano
261	Venezuela
215	Viet Nam
264	Yemen
265	Zambia
219	Zimbabwe

Índice de figuras

Índice de cuadros