Gestión y detección de inconsistencias en colecciones de referencias bibliográficas*

Jose A. Royo, Raquel Trillo, Sergio Ilarri, Eduardo Mena
{Departamento IEC, Departamento IIS, Departamento IIS, Departamento IIS}
Universidad de Zaragoza. María de Luna 1, 50018. Zaragoza, España.
{joalroyo, raqueltl, silarri, emena}@unizar.es

Resumen

Durante el ciclo de vida de un sistema de gestión de referencias bibliográficas es frecuente que aparezcan necesidades no previstas que obliguen a introducir nuevas funcionalidades. Es necesario un diseño basado en patrones que posibilite la introducción de nuevas características por distintos equipos de desarrollo. Por otra parte, el mantenimiento de la coherencia y eliminación de redundancias en las referencias bibliográficas es una tarea compleja: los diferentes usuarios pueden introducir datos distintos para una misma publicación o cometer errores tipográficos, lo cual debería ser detectado automáticamente.

En este artículo describimos un sistema de gestión de referencias bibliográficas, basado en patrones de diseño para facilitar la extensibilidad y escalabilidad. Proponemos y comparamos tres mecanismos que permiten mantener automáticamente la consistencia entre referencias bibliográficas.

Palabras clave: arquitectura de bibliotecas digitales, técnicas de filtrado, integración de recursos de información.

1. Introducción

El rápido crecimiento de Internet en los últimos tiempos y las ventajas que proporcionan las aplicaciones web han hecho que cada vez se demanden más. De hecho, la mayoría de las organizaciones están migrando la mayor parte de sus sistemas de información para hacerlos accesibles a través de un navegador web.

Dentro de éstos, los sistemas que permiten la búsqueda y recuperación de información constituyen una de las áreas de mayor interés actualmente. Durante el ciclo de vida del sistema, normalmente es necesario añadir nuevas funcionalidades, que pueden ser diseñadas por distintos equipos de desarrollo. Un diseño basado en el uso de patrones conocidos y probados permite una base de comunicación entre los miembros del proyecto y facilita la comprensión de la aplicación.

Nos centramos en el ámbito de las referencias bibliográficas y en los problemas que surgen en su gestión, tanto desde el punto de vista del mantenimiento y extensión de las funcionalidades como de la detección de inconsistencias y redundancias.

Para abordar el primer problema utilizamos una arquitectura en capas y los patrones [4] MVC (*Model View Controller*), VO (*Value Object*), DAO (*Data Access Object*), *Iterator* y *Facade*, lo cual nos proporciona dos claros beneficios: facilidad de mantenimiento y separación de responsabilidades en los equipos de trabajo.

Por otro lado, es necesario disponer de un mecanismo automático que permita mantener la consistencia de los datos, especialmente cuando la biblioteca digital es consultada y actualizada por varios miembros de distintos grupos de investigación. Así, cuando se añade una nueva referencia bibliográfica al sistema se debe verificar que no exista. Esta tarea no es

*Trabajo financiado por el proyecto CICYT TIN2004-07999-C02-02.

fácil debido a que las referencias pueden contener errores tipográficos, pueden desconocerse algunos datos (ISBN, mes de la publicación, etc.) o los mismos pueden diferir (por error o porque se expresa la misma información de forma sintácticamente distinta).

Aunque el trabajo presentado en este artículo podría ser utilizado con otros formatos de citas bibliográficas, nosotros consideramos BibTEX por ser el estándar de facto utilizado en entornos TEX [13]. De acuerdo con dicho formato, a una referencia se le asigna un identificador y consta de una serie de campos (autor, título, ISBN, etc). Sin embargo, diferenciar referencias bibliográficas utilizando el identificador no es válido para el mantenimiento de una biblioteca digital, ya que la información de una publicación digital es más extensa y rica semánticamente: en realidad lo que identifica a una publicación es la información contenida en sus campos. Por tanto, deben desarrollarse mecanismos más sofisticados de comparación.

En este artículo proponemos un sistema basada en patrones [4], servicios web [10], agentes inteligentes [19] y páginas web dinámicas [7], para el mantenimiento de referencias bibliográficas de investigación. Además, estudiamos y comparamos experimentalmente tres aproximaciones para detectar inconsistencias en las referencias bibliográficas. El funcionamiento de nuestra propuesta para gestionar un depósito de publicaciones utilizado conjuntamente por el grupo de Bases de Datos Interoperantes (BDI) y el de Sistemas de Información Distribuidos (SID), de las universidades del País Vasco y Zaragoza respectivamente, avalan la viabilidad y ventajas de nuestra aproximación.

A continuación presentamos la estructura del artículo. En la Sección 2 detallamos los patrones de diseño utilizados. En la Sección 3 describimos la arquitectura de nuestro sistema. En la Sección 4 proponemos y comparamos tres técnicas para la detección de inconsistencias. En la Sección 5 describimos algunos trabajos relacionados y, finalmente, en la Sección 6 recogemos nuestras conclusiones y trabajo futuro.

2. Patrones de diseño

En esta sección se presentan de forma resumida algunos de los patrones de diseño utilizados en la aplicación, y las ventajas que implica su utilización.

El patrón arquitectónico Model-View-Controller (MVC) hace una separación entre el modelo (lógica de negocio) y la vista (interfaz gráfica), gracias a un controlador que los mantiene desacoplados y los comunica gestionando las peticiones del usuario. De este modo se posibilita la reutilización de un mismo modelo con distintas vistas (por ejemplo, una vista de servicios web [10] y una basada en páginas dinámicas JSP [7]). En la capa modelo reside la lógica de la aplicación y diferenciamos dos partes: 1) objetos que se interrelacionan para representar la lógica de negocio del sistema, y 2) objetos que se encargan de la gestión de la persistencia de los datos (acceso, modificación,inserción, etc.).

Para el acceso a datos se ha implementado el patrón Data Access Object (DAO) en el que se han incluido una serie de métodos no comunes para proporcionar mayor generalidad y evitar modificar su implementación ante posibles cambios en la lógica de negocio. La estructura de este patrón es la siguiente:

- *DAOFactory*: Se encarga de crear una instancia del DAO del tipo adecuado según la fuente de datos utilizada proporcionando gran flexibilidad en la instalación/configuración de una aplicación.

- *DAO*: Abstrae las operaciones sobre la fuente de datos, proporcionando una API para acceder y manipular datos e independencia de la fuente.

- *DAOImpl*: Adapta el interfaz anterior a una fuente de datos concreta. Proporciona acceso a la fuente de datos y manipulación de datos mediante una API que necesita ser adaptada.

En general las operaciones que suele proporcionar un DAO son las siguientes: creación, borrado, actualización de un objeto y

búsquedas a partir de su identificador. Además se suelen incluir operaciones de búsqueda de conjuntos aplicando, por eficiencia, el patrón *Iterator*. Estas últimas dan lugar a varios métodos de búsqueda que reciben como argumentos las distintas combinaciones de criterios por los que se desea buscar. Por generalidad, y para minimizar el tamaño de interfaces y clases, agrupamos todas estas operaciones en un método general que recibe como argumento una lista de criterios. Además observamos que los resultados de las búsquedas suelen requerir ordenación por campos, por lo que esta tarea también se delega en el DAO, dando lugar a un método de la forma: *findByCriteriosByOrden(Criterios:Collection, startIndex: Entero, count:Entero, Orden:Collection)*. De este modo se aumenta la eficiencia de búsquedas ordenadas y se reduce la complejidad de la implementación de la lógica de negocio (ya que no se tiene que encargar de la gestión del acceso a datos y su ordenación).

3. Arquitectura

En esta sección presentamos la arquitectura del sistema, que mejora la presentada en [22]. Dicha arquitectura se diseñó como una serie de servicios Web [10], permitiendo una fácil interoperación con páginas HTML dinámicas y con otras aplicaciones mediante el uso de XML [15] y del protocolo SOAP [23]. Además, se consideraba la integración de colecciones de referencias bibliográficas preexistentes.

Desde entonces hemos mejorado nuestro trabajo mediante la aplicación de patrones de diseño (ver sección 2) y ampliando el tipo de información que puede asociarse a las publicaciones (digitalizaciones de las mismas, abstracts, etc.).

La arquitectura actual del sistema (ver Figura 1) mantiene la compatibilidad con la preexistente y además añade nuevas funcionalidades (destacamos el nombre de los modulos que las implementan):

- *Almacenamiento digital de información*: se permite almacenar otra información, tal como el fichero postcript, o imágenes

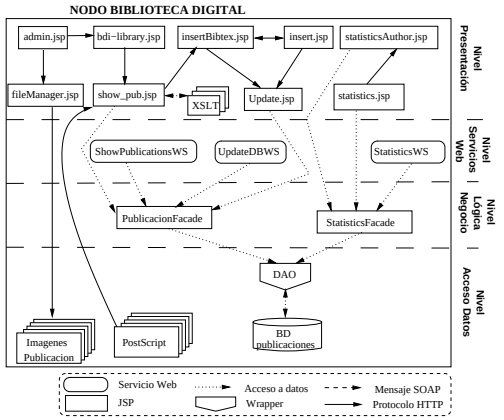

Figura 1: Arquitectura del Sistema

asociadas a la referencia bibliográfica de la publicación (*fileManager.jsp*) que contiene el trabajo investigador (por ejemplo, la portada, contraportada, índice, etc.), como se muestra en la Figura 2.

- *Definición de campos personalizados*: se permite definir campos no estándar para las referencias bibliográficas almacenadas en la base de datos. Dado que se trata de campos opcionales no se almacenan en la misma tabla que los campos estándar; se almacenan en una tabla aparte que tiene tres campos: uno para el identificador de la publicación, otro para el nombre del campo de la referencia bibliográfica y otro para el valor de dicho campo. De este modo evitamos el crecimiento desmesurado del número de columnas.

- *Adaptación de la presentación a distintos formatos*: se utilizan plantillas XSLT para mostrar la salida deseada (*show_pub.jsp y ShowPublicationsWS*).

- *Generación de estadísticas de los datos almacenados*: (*statistics.jsp, statisticsAuthor.jsp y StatisticsWS*) se permite consultar el número y tipo de publicación por autor o por grupo de autores y por año. Puede verse un ejemplo en la Figura 3.

El sistema presentado permite la paginación

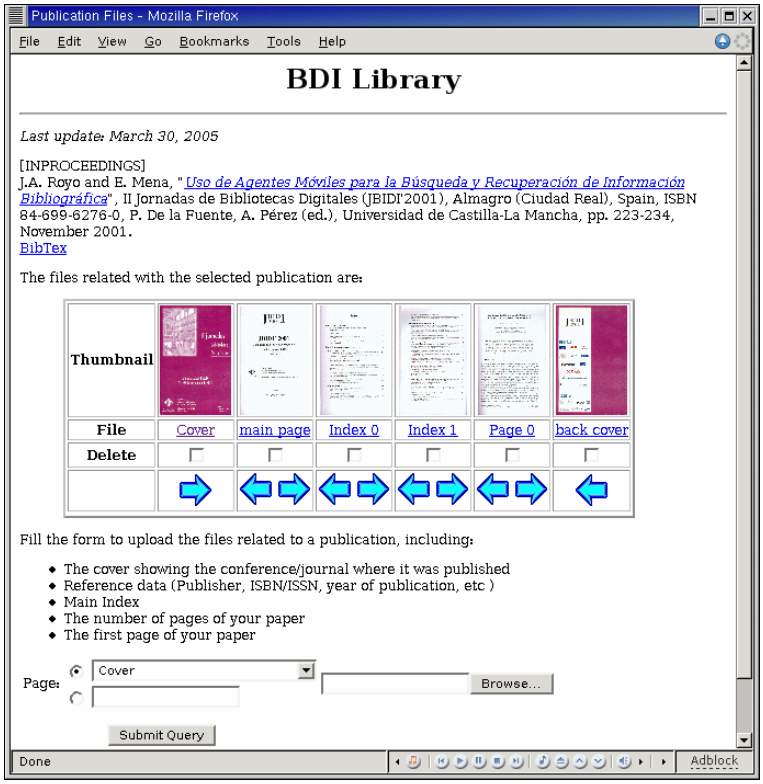

Figura 2: Digitalización de publicaciones

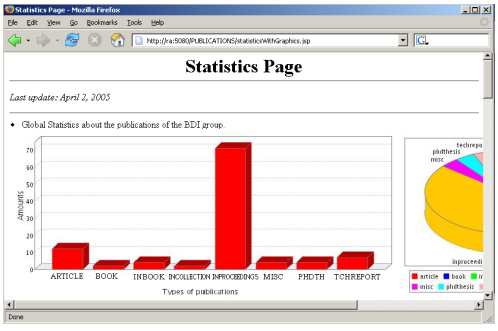

Figura 3: Estadísticas

de los resultados de salida mediante la utilización de los patrones *DAO* e *Iterator* [4].

4. Detección de inconsistencias

Los depósitos de referencias bibliográficas son utilizados de forma concurrente por un gran número de personas, tanto para consulta como para actualización. Del mismo modo, una tarea común en estos contextos es integrar varios depósitos de referencias bibliográficas.

En consecuencia, se precisa algún mecanismo que evite que se introduzca más de una referencia a la misma publicación. El identificador que se asigna a las referencias en el formato BibTEX es a todas luces insuficiente porque funciona sólo a modo de puntero para no tener que introducir todos los datos de la referencia directamente en el fichero TEX. Lo que verdaderamente identifica a una referencia es el conjunto de datos asociados.

El mecanismo de detección de inconsisten-

cias puede aplicarse de forma preventiva (para detectar que una referencia que se pretende insertar se refiere a la misma publicación que otra ya existente) o de forma correctiva (para examinar la lista de referencias actualmente almacenadas y detectar posibles inconsistencias y duplicados). La acción a realizar cuando se detecta que dos referencias son la misma es independiente del método de detección utilizado. En particular, se mostrará un aviso al usuario incluyendo las entradas BibTEX correspondientes y en el sistema quedará finalmente almacenada una sola referencia de acuerdo con los siguientes criterios:

- *Completitud*: la referencia resultante contendrá todos los campos incluidos en ambas referencias de entrada. Por ejemplo, si en una referencia se incluye información referente al número de páginas y en la otra no, es deseable guardar esta información en la referencia que finalmente se almacene.
- *Resolución de inconsistencias*: cuando las dos referencias tienen distintos valores asociados al mismo campo, se genera un mensaje de error y se solicita ayuda al usuario para que solucione el problema. Por ejemplo, si consideramos que dos referencias identifican a la misma publicación y tienen distinto número de autores (o aparecen en distinto orden), se le pedirá al usuario que verifique e introduzca la información correcta.

A continuación presentamos tres métodos que difieren en el criterio utilizado para determinar si dos referencias son iguales[1]. El primero es un método sencillo que utiliza simplemente un conjunto de reglas de comparación predefinidas. Los dos siguientes utilizan el algoritmo de Levenshtein para obtener las distancias de edición de las cadenas de caracteres correspondientes a los distintos campos BibTEX[2], y se diferencian únicamente en la manera en que dichas distancias se combinan para tomar una decisión final con respecto a la similitud de las referencias: utilizando redes neuronales o calculando una media ponderada adaptativa con pesos asignados a cada campo.

[1] Antes de realizar la comparación de las referencias bibliograficas se realiza una normalización [1] de las mismas (eliminación de mayúsculas, espacios redundantes, caracteres especiales).

[2] Incluyendo los campos personalizados, que influirán minimamente debido a su escasa aparición.

4.1. Comparador basado en reglas

Este comparador (presentado en [21]) se basa en la comparación carácter a carácter de las cadenas de texto correspondientes a un determinado conjunto de campos BibTEX seleccionados previamente. En nuestra implementación, los campos que comparamos son el título y los autores y no distinguimos entre mayúsculas y minúsculas (esta última diferencia sólo generará un aviso para el usuario).

Este método presenta notables deficiencias, como su incapacidad para detectar dos campos con distinto valor pero que contienen la misma información; por ejemplo, el contenido de uno de ellos podría incluir errores tipográficos, información incompleta, o emplear abreviaturas. Además, no se puede adaptar automáticamente a distintos contextos en los que se utilicen campos BibTEX no estándar, ya que no se utilizarían en la comparación. Las dos aproximaciones que se presentan a continuación pretenden solventar estos inconvenientes. Este método no detecta ninguna inconsistencia en el depósito de datos utilizado para realizar las pruebas.

4.2. Comparadores basados en el algoritmo de Levenshtein

En esta sección describimos dos aproximaciones para la detección de inconsistencias mediante el cálculo del grado de similitud de dos referencias bibliográficas: si el grado de similitud calculado supera un cierto umbral (α) se considera que ambas referencias son la misma.

Estas aproximaciones se basan en el algoritmo de Levenshtein [14], ampliamente utilizado para comparación de secuencias [12, 20], que calcula la distancia de edición entre dos cadenas de entrada (número mínimo de pasos para pasar de una a otra). El algoritmo de Levenshtein se ha venido utilizando en contextos tales como biología computacional (compara-

ción de cadenas de ADN) y para la implementación de correctores ortográficos. En nuestro contexto, la utilizamos entre cada par de campos BibTEX con el mismo nombre de las dos referencias de entrada, obteniendo así una medida de la similitud de los campos. A partir de los grados de similitud de los campos se obtiene un grado de similitud global de acuerdo con una de las dos aproximaciones que se muestran a continuación: utilizando redes neuronales o calculando una media ponderada adaptativa entre los campos.

4.2.1. Cálculo del grado de similitud con redes neuronales

En esta sección proponemos la utilización de redes neuronales [6] para calcular el grado de similitud de dos referencias bibliográficas a partir de las distancias de Levenshtein de sus campos. Consideramos una red neuronal de propagación hacia atrás con una capa de entrada con 29 neuronas[3] que reciben como entrada los grados de similitud de los campos de las dos referencias que se comparan, una capa intermedia con 14 neuronas [4], y una capa de salida con una neurona. Se utiliza una función de transferencia lineal entre neuronas, y la neurona de salida devuelve un número real en el rango $(-\infty, +\infty)$ indicando el grado de similitud global. En nuestra implementación el umbral que determina si dos referencias son o no la misma es igual a 0: si la salida es menor igual que 0 se consideran distintas, y la misma en caso contrario.

La red neuronal se ha entrenado utilizando aprendizaje supervisado, utilizando para ello un conjunto muestra de 170000 referencias bibliográficas, que se han obtenido a partir del depósito de datos utilizado (contiene 1000 referencias aproximadamente). La muestra utilizada para entrenar la red neuronal se ha creado introduciendo variaciones (insertar/eliminar campos, alterar el valor de los campos, etc) de forma aleatoria en las referencias bibliográficas. Para evitar un entrenamiento positivo/negativo de la red se realiza aproximadamente el mismo número de pares que devuelvan un resultado de igualdad que desigualdad.

Esta aproximación basada en redes neuronales mejora considerablemente a la anterior, dado que es mucho más flexible y proporciona una mayor tasa de aciertos que la que se obtiene comparando simplemente cadenas de caracteres. La red neuronal descrita se ha implementado utilizando Matlab [5] y en las pruebas realizadas se ha obtenido un porcentaje de error del 0.27 %.

Como contrapartida, hay que invertir un esfuerzo inicial en entrenar la red, y un aumento en el número de campos a considerar supone modificar la estructura de la red. Además, el método depende de cómo de representativo sea el conjunto de entrenamiento, lo cual no es en absoluto fácil de predecir.

4.2.2. Cálculo del grado de similitud con media ponderada dinámica

En esta sección comentamos un comparador de publicaciones basado en la distancia de Levenshtein que permite adaptarse automáticamente al depósito de datos y a la información almacenada acerca de las publicaciones.

Para ver si dos referencias bibliográficas, r_1 y r_2, son iguales se calcula una media ponderada de las distancias de Levenshtein entre cada uno de sus campos t. El valor del peso de un campo, $w(t)$, se calcula en cada momento como el cociente entre el número de valores distintos almacenados hasta entonces en el depósito de datos, $\#(t)$, y el número total de referencias bibliograficas, $\#referencias$. Por tanto, la distancia entre dos referencias bibliográficas es:

$$d(r_1, r_2) = \frac{\sum_t \frac{DL(r_1^t, r_2^t)}{w(t)}}{\sum_t w(t)}, t \in \{C(r_1) \cup C(r_2)\}$$

$$w(t) = \frac{\#(t)}{\#referencias}$$

[3] Igual al número de campos de una publicación que considera el comparador basado en redes neuronales.

[4] El número de capas intermedias y de neuronas en dicha capa se ha determinado experimentalmente tras realizar diversas pruebas.

siendo $C(r)$ los campos que tiene una referencia bibliográfica, $DL(r_1^t, r_2^t)$ es la distancia de Levenshtein entre r_1 y r_2 para el campo t.

Este método considera que dos publicaciones son iguales si $d(r_1, r_2)$ es menor que un cierto umbral (α). Experimentalmente hemos determinado que un valor adecuado para α en nuestro contexto es 0.3 (ver Tabla 1).

α	Porcentaje de Fallos
0.7	6.24 %
0.5	0.89 %
0.3	0.09 %

Cuadro 1: Errores Detectados según el Umbral

La ventaja de utilizar este método es que los pesos de cada uno de los campos se actualizan dinámicamente adaptándose a los datos almacenados. Es decir, la importancia asignada a cada uno de los campos de una referencia bibliográfica varía dependiendo de el número de veces que se repite cada uno de los valores de ese campo en el depósito de datos. Por ejemplo, si solamente se almacenan referencias de un único autor entonces ese campo será poco relevante para determinar si dos publicaciones son iguales.

5. Trabajos relacionados

Se han propuesto diversas heurísticas para medir la similaridad entre dos campos de una base de datos [16, 20, 24, 2]. Aunque podríamos aplicar en nuestro trabajo cualquiera de estas aproximaciones, hemos decidido utilizar la distancia de Levenshtein porque es el modelo más ampliamente estudiado y para el que se han desarrollado los algoritmos más eficientes [18].

En [2] se propone combinar las medidas de similaridad de campos individuales utilizando *support state machines*, pero este método sufre desventajas similares a nuestra aproximación con redes neuronales (coste de entrenamiento).

En [9] se proponen diversos métodos para agrupar conjuntos de referencias bibliográficas que pueden ser potenciales duplicados. Proponen un algoritmo de comparación del número de palabras y frases no coincidentes. En consecuencia, es de esperar que su método no sea capaz de detectar errores tipográficos.

En [3] se propone un sistema *peer-to-peer* para el intercambio de referencias bibliográficas. Proponen utilizar varias métricas de distancia simultáneamente para mejorar la precisión de la detección de duplicados. Sin embargo, a diferencia del nuestro, su método no se adapta automáticamente al depósito de datos analizado.

Como en nuestra propuesta inicial de comparador basado en reglas, en [11] se propone el uso de reglas para detección de duplicados. La novedad es que su objetivo es generar estas reglas automáticamente a partir de coeficientes de similaridad obtenidos previamente. Sin embargo, su contexto es distinto (datos biológicos) y por lo que sabemos no ha habido ningún intento de aplicar esas ideas a colecciones de referencias bibliográficas.

Otros trabajos complementarios al nuestro tienen como principal objetivo determinar algoritmos para detectar eficientemente (con menor coste que la solución trivial que es $O(n^2)$) los duplicados en un conjunto preexistente de referencias bibliográficas [8, 17].

6. Conclusiones y trabajo futuro

En este artículo hemos presentado los avances realizados en la biblioteca digital de nuestro grupo de investigación, centrándonos en los principales problemas que surgen en la fase de mantenimiento. Nuestro sistema se ha realizado utilizando patrones de diseño, lo cual conlleva una mayor robustez, modularidad y fiabilidad, facilitando futuras ampliaciones y cambios. Además, se han cuidado los aspectos de interfaz de usuario y versatilidad, permitiendo por ejemplo la generación automática de bibliografía en diferentes formatos. Por último, hemos estudiado y comparado tres mecanismos para la detección de inconsistencias y eliminación de redundancias en citas bibliográficas. Proponemos finalmente el método de media ponderada adaptativa basado en distancia de Levenshtein, que proporciona buenos resultados y además es independiente del dominio

de aplicación.

Actualmente estamos elaborando plantillas XSLT para hacer accesible el sistema de publicaciones a dispositivos móviles, como PDAs.

Referencias

[1] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval.* Adisson-Wesley, 1999.

[2] Bilenko, M., and Mooney, R. J. Learning to combine trained distance metrics for duplicate detection in databases. Tech. Rep. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX, Feb. 2002.

[3] Broekstra, J., and et al. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of SemPGRID '04, 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing* (New York, USA, May 2004), pp. 3–22.

[4] D, A., Crupi, J., and Malks, D. *Core J2EE Patterns. Best Practices and Design Strategies.* Prentice Hall, 2001.

[5] Gustafsson, Fredrik / Bergman, N. *MATLAB® for Engineers Explained.* Springer-Verlag, 2003.

[6] Hagan, M., Demuth, H., and Beale, M. *Neural Network Design.* Martin Hagan, January 2002.

[7] Hall, M. *Core Servlets and Java Server Pages(JSP).* Prentice Hall PTR/Sun Microsystems Press, May 2000.

[8] Hernandez, M. A., and Stolfo, S. J. The merge/purge problem for large databases. In *SIGMOD Conference* (1995), pp. 127–138.

[9] Hylton, J. A. Identifying and merging related bibliographic records. Tech. Rep. MIT/LCS/TR-678, 1996.

[10] Jewell, T., and Chappell, D. *Java Web Services.* O'Reilly & Associates, March 2002.

[11] Koh, J. L., Lee, M. L., Khan, A. M., Tan, P. T., and Brusic, V. Duplicate detection in biological data using association rule mining. In *ECML/PKDD Workshop on Data Mining and Text Mining for Bioinformatics* (September 2004).

[12] Kruskal, J. B. An overview of sequence comparison: time warps, string edits, and macromolecules. *SIAM Review 25*, 2 (April 1983), 201–237.

[13] Lamport, L. *Latex: a document preparation system.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[14] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory 10*, 8 (1966), 707–710. Original in *Doklady Akademii Nauk SSSR* 163(4): 845–848 (1965).

[15] McLaughlin, B. *Java & XML, 2nd Edition: Solutions to Real-World Problems.* O'Reilly & Associates, September 2001.

[16] Monge, A. E., and Elkan, C. The field matching problem: Algorithms and applications. In *Knowledge Discovery and Data Mining* (1996), pp. 267–270.

[17] Monge, A. E., and Elkan, C. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Research Issues on Data Mining and Knowledge Discovery* (1997), pp. 0–.

[18] Navarro, G., and Raffinot, M. *Flexible Patern Matching in Strings.* Cambridge University Press, 2002.

[19] Pitoura, E., and Samaras, G. *Data Management for Mobile Computing*, vol. 10. Kluwer Academic Publishers, 1998.

[20] Ristad, E. S., and Yianilos, P.Ñ. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 5 (1998), 522–532.

[21] Royo, J., and Mena, E. Uso de agentes móviles para la búsqueda y recuperación de información bibliográfica. In *II Jornadas de Bibliotecas Digitales (JBIDI'2001), Almagro (Ciudad Real), Spain* (November 2001), Universidad de Castilla-La Mancha, pp. 223–234.

[22] Royo, J., and Mena, E. Gestión de bibliotecas digitales de publicaciones de investigación. In *III Jornadas de Bibliotecas Digitales (JBIDI'2002), El Escorial (Madrid), Spain* (November 2002), Universidad Politécnica de Madrid, pp. 77–86.

[23] Snell, J., Tidwell, D., and Kulchenko, P. *Programming Web Services With SOAP.* O'Reilly & Associates, December 2001.

[24] Yianilos, P. The LikeIt intelligent string comparison facility. Tech. Rep. 97-093, 1997.