Ried

REVISTA IBEROAMERICANA
DE EDUCACIÓN A DISTANCIA
Vol. 3 • Nº 1
Junio, 2000

La Ingeniería Lingüística con Java

(Linguistic Engineering with Java)

TIM READ
ELENA BÁRCENA
(UNED, España)

RESUMEN: *Este artículo intenta hacer una reflexión teórica sobre Java que sea de interés y utilidad para la comunidad informática, y especialmente para los programadores de aplicaciones lingüísticas, ya que, como ocurre con muchas otras tecnologías asociadas con la Web, hay cierta confusión, falta de información y malentendidos sobre este lenguaje de programación. Aunque muchos lingüistas computacionales hayan oído hablar de Java, considerado su uso e incluso llegado hasta su primer applet (aplicación pequeña para una página Web) 'Hola mundo', quizás no se hayan planteado todas las implicaciones del uso de este lenguaje y del papel que podría desempeñar en el desarrollo de aplicaciones lingüísticas hoy en día. Es necesario clarificar de antemano que aquí no se tiene el propósito de presentar Java como sustituto absoluto de los lenguajes de programación tradicionales para aplicaciones lingüísticas ya que, entre otras cosas, la historia de Java es demasiado corta para examinar comparativamente las aplicaciones escritas en este lenguaje y en otros y extraer conclusiones definitivas.*

Java - Ingeniería Lingüística - Aplicaciones basadas en la Web

ABSTRACT: *In this article the Java programming language and runtime environment is examined, with special reference being made to its applicability to the creation of applications in the area of linguistic engineering. Java has been presented by Sun Microsystems as an easy to use, secure and robust programming language, which is capable of being used to produce distributed platform independent applications, capable of running across the Web. Following a brief analysis of these claims, some of the principal requirements of linguistic applications are outlined, and the properties of Java that can be used to meet these requirements are considered. In this process a few brief remarks are made regarding the comparison of Java and other programming languages like Lisp, Prolog, C and C++. Finally, an example system which is being developed by the authors is presented, in order to illustrate the main arguments contained in this article.*

Java - Linguistic Engineering - Web-based Applications.

De vez en cuando se crea una nueva tecnología que ofrece interesantes posibilidades a distintos sectores de la sociedad: organismos oficiales, instituciones académicas, centros de investigación, compañías privadas y públicas, e individuos particulares. Uno de esos avances tecnológicos es la Internet (o, para ser más específicos, la Web), que está produciendo cambios revolucionarios en las vías de intercambio de información a escala mundial. Además, los primeros resultados del proyecto Internet2 (www.internet2.edu) pronostican notables mejoras en la calidad y velocidad actuales de la transmisión de información, lo cual abrirá aún más posibilidades para un futuro no muy lejano.

En los primeros días de la Web, la incorporación del CGI (*Common Gateway Interface*) diseñado por Denny y herramientas relacionadas [2] permitieron hacer algo más con las páginas Web que una simple recuperación de documentos HTML (*Hypertext Markup Language*). Con dichos elementos, las páginas Web podían llevar enlaces a programas en el servidor que desempeñaban tareas como el acceso a bases de datos. Esta tecnología fue adoptada por lingüistas computacionales para producir herramientas lingüísticas para la Web, como diccionarios monolingües y bilingües. Las herramientas de este tipo han demostrado ser de considerable utilidad, aunque las múltiples limitaciones del CGI (por ejemplo, interfaz basada en plantillas, capacidad gráfica limitada, tipos de datos no arbitrarios, carga pesada en el servidor, interacción de usuario limitada) han contribuido a su falta de sofisticación.

Quizás Sun Microsystems haya ofrecido una solución con el lanzamiento del lenguaje de programación Java. Java empezó como el lenguaje Oak a principios de los 90 cuando la Web todavía se desarrollaba en CERN [5]. Desde entonces ha ido ganando mucha popularidad como lenguaje de programación con el que se pueden desarrollar aplicaciones portátiles sofisticadas que funcionan en entornos heterogéneos, desde las páginas Web a través de la Internet, hasta los ordenadores independientes [3].

Java ha sido presentado por Sun Microsystems como un lenguaje de programación que es, ante todo, portátil, seguro, robusto, fácil de usar y, como acabamos de mencionar, capaz de producir aplicaciones que pueden funcionar distributivamente en la Web. Pasemos a comentar estas afirmaciones.

Primero, veamos la afirmación de que Java es portátil a través de distintas plataformas de hardware, no sólo en términos de su código fuente como el lenguaje C, sino también de su código compilado. Java debe su independencia de la plataforma a la combinación de un compilador que produce código byte independiente de la máquina real (es decir, como si fuera para una máquina virtual [MV]), y de un intérprete que funciona en la máquina de ejecución para convertir el código byte MV en código nativo. Además, Java está disponible en la mayoría de las plataformas. Parece, por tanto, que la afirmación de Sun Microsystems sobre su transportabilidad binaria entre diferentes plataformas es cierta. Aunque esto queda todavía por demostrar para las grandes aplicaciones de Java en las pá-

ginas Web, los *applets* funcionan en cualquier plataforma con un navegador que contenga un intérprete MV de Java.

El problema con la técnica de la compilación parcial a un código byte y su consiguiente interpretación se reduce a una cuestión de velocidad de ejecución, ya que los lenguajes interpretados funcionan más lentamente que los compilados. Por lo tanto, la velocidad de un programa escrito en Java depende del tamaño de la aplicación (sobre todo si está funcionando en Internet), la potencia del ordenador cliente y, sobre todo, el funcionamiento de la MV. Una mala combinación de estos factores puede producir programas muy lentos. Entre las soluciones que se han proporcionado cabe destacar la modificación del diseño del intérprete MV de Java para que incluya una técnica que puede acelerar la ejecución del código hasta 50 veces, conocida como compilación JIT (*Just In Time*). Además, ya existe una nueva MV basada en la optimización en tiempo real de la compilación y ejecución del código byte que se llama *HotSpot* (www.javasoft.com/javaone/sessions/slides/TT01/tt01_43.htm). Debe mencionarse también el desarrollo de la MV de Java en silicona, o sea, el desarrollo de microprocesadores que ejecutan la interpretación de código byte de Java. Aparte de estos procesadores dedicados, el desarrollo actual de la nueva tecnología de microprocesadores, en combinación con intérpretes rediseñados para aprovecharla, eliminará sin duda en el futuro el problema de la velocidad de interpretación.

Segundo, examinemos la afirmación de que Java proporciona una operación segura de programas a través de una red. Esto se puede lograr de tres modos. En primer lugar, la compilación del código byte MV no permite ningún acceso directo a la memoria ni otras operaciones inseguras. En segundo lugar, con el fin de evitar la generación de código especialmente modificado por compiladores maliciosos, el intérprete MV comprueba la seguridad e integridad del código byte antes de ejecutarlo (utilizando el cargador de clases o *class loader*). En tercer lugar, los dos navegadores principales (Netscape Navigator y Microsoft Internet Explorer) también añaden un nivel más de seguridad al evitar que los applets de Java sin una firma digital accedan al sistema de archivo local. Los agujeros de seguridad que se han encontrado en la MV de Java hasta la fecha han sido reparados rápidamente y se sigue mejorando su seguridad por medio de claves de codificación y códigos con firmas digitales incorporadas. De todos modos, los únicos *applets* maliciosos que existen hoy en día no han conseguido más que producir irritaciones sin importancia al usuario, como bloquear su navegador o tocar música de fondo molesta (ataques conocidos como *denial of service attacks*) [1].

Tercero, la afirmación de robustez. Para ayudar al desarrollo de programas robustos, Java no contiene punteros. Un mecanismo de control de basura maneja la asignación de la memoria, eliminando así una fuente de múltiples problemas: los errores de los punteros relacionados con la memoria. Java ha sido diseñado para minimizar los errores de los programas incluyendo el tratamiento consistente de errores, *strong typing* (cuando el lenguaje obliga al programador a

ceñirse a un uso rígido de tipos de datos con sus variables), y la ausencia de construcciones inseguras, indefinidas o dependientes de la arquitectura.

Cuarto, la afirmación de facilidad de uso. Java es un lenguaje pequeño, lo cual hace que sea fácil de aprender. Su estructura orientada a objetos produce un código que es sencillo de leer y escribir, y facilita su reutilización. El único pequeño problema para los programadores es su forzada visión del mundo orientada a objetos. Los programadores que no tengan experiencia en el diseño orientado a objetos tendrán que soportar la curva de aprendizaje asociada con la adopción de esta nueva visión antes de que adquieran fluidez en la tarea de diseño e implementación de este tipo de código. Un rasgo poderoso de Java es su mecanismo entretejido, que permite la concurrencia de tareas. Sin embargo, aunque las aplicaciones entretejidas pueden mejorar la actuación o ejecución de un programa, también pueden causar algunos quebraderos de cabeza hasta que se aprendan los principios básicos de la programación orientada a objetos.

Quinto y último, se dice que Java permite la operación de aplicaciones distribuidas en la Web. En efecto, se pueden desarrollar *applets* para ponerlos en las páginas Web, lo cual permite a los programas funcionar a través de la Internet. Si estos programas funcionan en la máquina cliente, y no en el servidor, el código MV tendrá que ser transferido por la Internet antes de su puesta en marcha. Esto, dependiendo de la velocidad de la conexión a la Internet, puede resultar en una larga espera. Sin embargo, una vez que se cargan los *applets* en la memoria, funcionan tan rápidamente como los programas locales. Esto quiere decir que, o bien todos los datos que necesita un *applet* deben cargarse cuando se transfiere el applet por la Internet por primera vez (lo cual podría ser muy lento), o si no el applet debe ponerse en contacto con el servidor para acceder a la información siempre que sea necesaria.

Además de los applets están los servlets. Son aplicaciones de Java que funcionan en el servidor. Entre las ventajas de esta tecnología con respeto a los programas CGI está el hecho de que solamente hay un proceso para cada servlet, independientemente del número de usuarios que accedan a él, y que se puede aprovechar el mismo código Java para el cliente y el servidor.

Después de este breve análisis debería estar claro que Java posee varias características que pueden hacer de él una alternativa interesante para aplicaciones informáticas. En cuanto a las aplicaciones de Lingüística Computacional propiamente dichas, Java tiene similitudes con algunos de los lenguajes de programación tradicionalmente empleados para este tipo de aplicaciones como Lisp, Prolog o C, pero ha sido diseñado desde cero para evitar algunos de sus problemas, como los de la aritmética de punteros y el manejo de memoria.

Hay lenguajes, como Lisp y Prolog, con mecanismos computacionales (por ejemplo, la manipulación de listas, el emparejamiento de patrones) que han de ser incorporados en otros lenguajes. Por ejemplo, C y C++ se caracterizan por su

falta de adecuación para el procesamiento y representación de estructuras de datos abstractas y complejas como las palabras y las oraciones. Sin embargo, Lisp y Prolog se utilizan normalmente en el entorno de los laboratorios informáticos y en estos días cada vez se están diseñando más programas para funcionar autónomamente en ordenadores desde casa. Por esto, a pesar de todo, hay una tendencia hacia lenguajes como C y C++, ya que los programas escritos en estos lenguajes no requieren un entorno de ejecución complicado para el usuario.

Esto último también es aplicable a los programas en Java. Además, Java contiene mecanismos para el procesamiento y representación de estructuras abstractas, como el stream tokenizer (para, por ejemplo, extraer unidades lingüísticas de un contexto oracional). Debe reconocerse que tales mecanismos no han sido hasta ahora tan flexibles como los que proporcionan Lisp y Prolog pero, como explicaremos más adelante, la combinación de lo que ofrece Java con su fácil acceso a una base de datos proporciona un modo flexible para procesar el lenguaje natural. Java también proporciona facilidades para acceder a la tecnología del reconocimiento y síntesis de la voz. Se podría añadir que un rasgo clave de los lenguajes orientados a objetos como Java es que el código es reutilizable. Por lo tanto, el coste inicial del diseño y desarrollo de bibliotecas en Java para la representación y manipulación de tipos de datos abstractos y complejos será compensado con su uso. Además, se espera que las librerías empiecen a distribuirse rápidamente por la comunidad investigadora.

Para ver la adecuación de cualquier tecnología a un tipo específico de aplicación es conveniente hacer un análisis de requisitos. Por supuesto, estas condiciones varían de aplicación a aplicación, y el campo de la Lingüística Computacional no es una excepción. Aun así, creemos posible elaborar una lista de requisitos, que no pretende ser exhaustiva, sino una selección de requisitos genéricos básicos, a la que cada proyecto concreto puede añadir sus rasgos propios.

En la elaboración de esta lista de requisitos se obviará la calidad del output del sistema porque se considera que este factor depende más de la teoría lingüística que del lenguaje de programación [4]. Con varios niveles de dificultad, un ingeniero puede implementar la mayoría de los algoritmos lingüísticos en lenguajes de programación que van desde el ensamblador 6802 hasta el Lisp común. Los requisitos a continuación están más relacionados con cuestiones computacionales. Primero, simplicidad de la modificación y puesta al día del conocimiento lingüístico del sistema, es decir, que el conocimiento lingüístico pueda almacenarse de tal modo que los lingüistas y otros expertos no informáticos puedan acceder a él y modificarlo fácilmente. Segundo, una interfaz que no requiera tampoco conocimiento informático experto para su utilización, y que ofrezca un acceso a la información rápido, sofisticado desde un punto de vista lingüístico y flexible. Tercero, independencia total del sistema, es decir, un sistema que funcione con cualquier combinación de hardware y software, sin necesidad de hacer una versión para cada combinación de ordenador y sistema operativo. Cuarto, una aplicación

autónoma que no requiera ningún entorno de apoyo adicional durante el tiempo de ejecución que sea difícil de instalar y manejar para los expertos no informáticos. Quinto, accesibilidad a la aplicación, es decir, un sistema que sea accesible a escala local (en un ordenador sin conexión a la red) y remotamente a través de la Internet. Sexto y último, la reutilización del conocimiento lingüístico, es decir, la transportabilidad del conocimiento lingüístico a un nuevo sistema sin ningún esfuerzo adicional.

Veamos cómo satisface Java estos requisitos. Primero, uno de los aspectos más prometedores de Java como herramienta (especialmente para el desarrollo de aplicaciones que funcionen desde el servidor en un entorno de red, como por ejemplo los servlets, es su capacidad de acceder a cualquier base de datos, lo cual se hace a través del JDBC (Java Database Connectivity). Esto permite a las aplicaciones y applets acceder a una base de datos independientemente de su categoría y del sistema operativo subyacente. Por lo tanto, una elección lógica para asegurar la facilidad del acceso al conocimiento lingüístico es almacenar la información en una base de datos. Entonces, dicho conocimiento puede cargarse durante el tiempo de ejecución o cuando se transfiere el applet a través de la red.

Segundo, la interfaz puede desarrollarse de distintos modos ya que Java contiene librerías para asistir en la construcción de interfaces. La interfaz de una herramienta lingüística puede variar dependiendo de si está contenida en una página Web como un applet o es una aplicación autónoma con su propio look and feel (sus propiedades visuales y capacidades interactivas). Ambos tipos de interfaz pueden funcionar a escala local, a través de una intranet, o a través de Internet. Como se mencionaba anteriormente, Java puede desempeñar múltiples tareas a través de su mecanismo de entretejido, y cada tarea puede dividirse en hilos (por ejemplo, uno que controle el acceso a la lengua fuente y otro que controle el acceso a la lengua de destino) para aumentar la velocidad del procesamiento.

Tercero, las aplicaciones de Java son, en principio, completamente independientes del sistema. Por lo tanto, si están implementadas como un applet, la herramienta lingüística puede funcionar a escala local, en una intranet o en Internet sin necesidad de cambiar una sola línea del código. La única diferencia en los tres casos es la localización de la base de datos (en la máquina local o en un servidor en algún punto de la red). Esta información puede ser controlada por una sola variable del sistema. El uso de method overloading (sobrecarga de método, que es un rasgo de los lenguajes orientados a objetos) implica que se pueden escribir las distintas versiones de un mismo método de acceso a la base de datos para tratar automáticamente las distintas localizaciones de la base de datos. Una de las ventajas de una base de datos localizada en un servidor de la red es que puede modificarse y ponerse al día sin la necesidad de enviar nuevos archivos de datos a todos los usuarios.

Cuarto, una ventaja de insertar la herramienta lingüística como un applet dentro de una página Web es que la mayoría de los navegadores vienen con in-

térpretes Java VM y, por lo tanto, no se necesitan sistemas de ejecución de Java adicionales. Sin embargo, debido a la falta de estandarización en la MV de Java de los principales navegadores, Sun Microsystems ha producido un software plug-in (extensión) para proporcionar una MV de Java estándar (www.javasoft.com/products/plugin).

Quinto, como se ha explicado anteriormente, la herramienta lingüística puede funcionar a través de una red y también autónomamente. Los usuarios que tienen una copia local y acceso ocasional a la Internet pueden acceder on-line a nuevas versiones de los archivos de la base de datos para la puesta al día del conocimiento lingüístico de la herramienta, sin necesidad de que se distribuyan nuevas copias en disquete o en CD.

Sexto y último, la separación del conocimiento lingüístico (en la base de datos) de la correspondiente herramienta implica que el conocimiento puede ser reutilizado para nuevas aplicaciones sin mayor dificultad. Además, como es fácil cambiar la estructura de las tablas de una base de datos (o crear nuevas tablas integrando las existentes), también es fácil modificar el conocimiento lingüístico sin tener que volver a introducir la información desde el principio. Finalmente, si las plantillas y entradas originales se dejan intactas en las tablas, es irrelevante si se añaden o no nuevas entradas, ya que la naturaleza del acceso de SQL (Structured Query Language) no generará mensajes de error (a diferencia de los programas que cargan su conocimiento lingüístico desde un archivo).

Veamos ahora dos sistemas lingüísticos escritos en Java como ejemplos de aplicaciones en este lenguaje: el Profesor Virtual de Inglés (PVI), desarrollado por los autores y JCord, desarrollado por el primer autor.

El PVI es un agente inteligente basado en la Web que está siendo diseñado y desarrollado utilizando sofisticadas técnicas de Lingüística Computacional para complementar las vías disponibles de aprendizaje a distancia, ofreciendo de modo autónomo orientaciones inteligentes e infinita práctica al estudiante. Las características fundamentales de la arquitectura del PVI son las siguientes:

- generatividad: el PVI no funciona sobre una base de datos de ejemplos limitada, sino sobre una gramática productiva al uso de los hablantes humanos;
- autocorrección: no es necesario que intervenga un profesor para proporcionar las soluciones correctas;
- integración y reutilización: el conocimiento central lingüístico y el de dominio pueden compartirse y emplearse para distintas aplicaciones e idiomas;
- modularidad y organización jerárquica: separación del conocimiento lingüístico y los algoritmos, y otras divisiones sub-modulares;

- transportabilidad (al estar programado en Java).
- En el momento de escribir este artículo, el PVI contiene cinco módulos completos y operativos:
- JaBot - Secretario Virtual, para la búsqueda con input aproximado y bilingüe de información dentro de un sitio Web [6];
- The Virtual Verb Trainer, para la práctica de formación y uso de verbos en inglés en un contextooracional;
- The Virtual Corrector, para la comprobación de la corrección gramatical y léxica de un texto escrito en inglés;
- The Virtual Armchair Companion, para la consulta sobre textos adecuados de lectura en inglés según el nivel y preferencias de cada estudiante;
- The Virtual Word Feeder, para la adquisición de nuevo vocabulario en inglés.

Hay además otros módulos en diversos estadios de diseño y desarrollo. Por el momento todos son pequeños para facilitar su carga desde la Web, pero se prevé una segunda fase en la que se creará una arquitectura de cliente-servidor para el PVI. Esto se hará de modo que el conjunto de los módulos de su arquitectura integrada permanezca en el servidor durante el acceso como medida de protección y control de acceso, así como para mejorar la velocidad de arranque y funcionamiento y la interactividad entre las partes del sistema (consecuentemente, los módulos formarán parte del servidor y podrán ser más grandes y complejos). El desarrollo progresivo del PVI conllevará un aumento de su inteligencia, conocimiento del dominio y capacidad de gestión autónoma, una interfaz sofisticada basada en el lenguaje natural con mayor integración de los módulos y, por último, la incorporación de información sobre cada estudiante (su progreso, problemas particulares, etc.).

JCord es una herramienta de concordancias basada en XML que ha sido desarrollada como parte del proyecto RILE (un proyecto para la construcción de un servidor de recursos para el desarrollo de la ingeniería lingüística en español. Está financiado en el marco del Programa de Fomento de la Tecnología Industrial de la iniciativa ATYCA en el sector temático «Cultura y lengua». Este sector está integrado en el ámbito de aplicación «Tecnologías y Aplicaciones para la Sociedad de la Información» del Ministerio de Industria y Energía, www.miner.es), y se puede acceder a ella a través de la página Web de entrada http://rile.sema.es. Después de pinchar en 'ACESSO AL SERVIDOR', hay que pinchar en 'Concordancias'. Se trata de un servlet de Java que usa el parser XML de IBM. Ha sido elaborado para trabajar con archivos en formato XML según la DTD (Document Type Definition - un conjunto de reglas que definen la estructura de cualquier archivo XML basado en él) que ha sido desarrollada para este proyecto.

No se trata simplemente de archivos de texto etiquetados en XML, sino de archivos que contienen el resultado de un análisis morfosintáctico realizado por herramientas desarrolladas por los grupos que colaboran en el proyecto, es decir, que cada palabra en el archivo lleva información gramatical. Por lo tanto, al elegir un archivo entre los disponibles (en el momento de escribir este artículo los usuarios no pueden introducir sus propios textos, pero esto está previsto para una futura versión del proyecto), el servlet forma un objeto DOM para contener y representar la información del archivo y reconstruye el texto presentado al usuario en forma de una página Web.

Además de poder realizar búsquedas de palabras en el texto como cualquier herramienta de concordancias, se puede hacer búsquedas para la co-ocurrencia de palabras, y también, basándose en el conocimiento lingüístico subyacente al texto, para la (co-) ocurrencia de lemas, es decir, que se puede buscar un lema en el texto y se obtendrán todas las palabras en el texto que tienen ese mismo lema. En el caso de la búsqueda de pares de palabras o lemas, se puede especificar una separación de palabras de cero a nueve. Además, se puede combinar la búsqueda de palabras/lemas con signos de puntuación para poder ver la relación entre ellos. Por ejemplo, se podría buscar el punto con el lema 'ser' y una separación de cero palabras para obtener todas las oraciones que empiecen con una palabra cuyo lema es 'ser'...

Se han comentado anteriormente las ventajas de los servlets de Java sobre los programas CGI. Además, queda por añadir que el servlet de Java que forma JCord está diseñado de tal modo que el objeto DOM construido para cada archivo esté mantenido en la memoria del servidor entre las consultas del usuario. Por lo tanto, la primera vez que el usuario accede a un texto, tendría que esperar a que JCord construya el objeto DOM y realice la búsqueda sobre él. En la versión actual de la herramienta, los cuatro tipos de búsquedas que se pueden hacer (palabra, palabra múltiple, lema, lema múltiple) son fijos. En la próxima versión, las categorías de búsqueda se formarán a partir de la DTD de cada documento, es decir, que no habrá límite de categorías de búsqueda y que la herramienta funcionará con cualquier combinación de archivo XML y DTD. En lugar de ser simplemente una herramienta de concordancias, se tratará de una herramienta de búsqueda genérica para archivos XML (¡que tengan DTD!).

Esperamos con este artículo haber disipado algunos malentendidos sobre Java, sobre todo a aquellos programadores que quieran desarrollar aplicaciones lingüísticas. El atractivo de Java descansa sobre cinco pilares: transportabilidad, seguridad, robustez, facilidad de uso y funcionamiento distribuido en la Web. Además, Java tiene otros puntos fuertes como mecanismos para el procesamiento y representación de estructuras abstractas y fácil acceso a las bases de datos que almacenan el conocimiento, que lo hacen interesante para aplicaciones lingüísticas. Esto no quiere decir que se deban abandonar necesariamente los lenguajes de programación más tradicionales para este tipo de aplicaciones, como

Lisp, Prolog o C, sino simplemente que convendría estar alerta de lo que sucede en el entorno de Java y tomar en consideración las posibilidades que se van abriendo con el desarrollo continuo de esta tecnología.

REFERENCIAS BIBLIOGRÁFICAS

[1] Dean D., E. Felten & D. Wallach (1996) 'Java Security: From HotJava to Netscape and beyond'. En Proceedings of the IEEE Symposium on Security and Privacy. (Oakland, California).

[2] Heslop B. & L. Budnick (1996) Publicar con HTML en Internet. Madrid: Anaya.

[3] Howard E.R. (1996) 'The Java Book List' (http://sunsite/unc.edu/javafaq/books.html).

[4] Lehrberger J. & L. Bourbeau (1988) Machine translation: linguistic characteristics of MT systems and general methodology of evaluation (Linguisticae Investigationes: Supplementa, 15). Amsterdam: John Benjamins.

[5] Naughton P. (1996) The Java Handbook. Londres: MacGraw Hill.

[6] Read T. & E. Bárcena (1998) 'JaBot: a multilingual Java-based intelligent agent for Web sites'. Pendiente de publicación en Proceedings of COLING-ACL'98. Université de Montrèal.

[7] Read T., E. Bárcena & P. Faber (1997) 'Java and its role in Natural Language Processing and Machine Translation'. En Proceedings of the Machine Translation Summit VI. University of California at San Diego.

PERFIL ACADÉMICO Y PROFESIONAL DE LOS AUTORES

Tim Read es Licenciado y Doctor en Informática por las universidades de The West of England y Birmingham respectivamente. Tras doctorarse en 1995, ha trabajado como profesor en la Universidad de Granada y la UNED, participando en proyectos de investigación nacionales y europeos. Actualmente es profesor del Departamento de Ingeniería Eléctrica, Electrónica y de Control de la UNED. Sus intereses de investigación incluyen la aplicación de Java y XML para programas basados en la Web.
tread@ieec.uned.es

Elena Bárcena ha realizado una licenciatura en Filología Inglesa (Universidad de Deusto), M.Sc. in Machine Translation y Ph.D. in Computational Linguistics (UMIST, Manchester). Tras doctorarse en 1994, trabajó como investigadora postdoctoral en las universidades de Lieja, Sevilla y Granada. Actualmente es profesora del Departamento de Filologías Extranjeras y sus Lingüísticas de la UNED, donde colabora en diversos proyectos de investigación nacionales y europeos sobre ingeniería lingüística.
mbarcena@flog.uned.es