Procesamiento del Lenguaje Natural, núm. 35 (2005), pp. 303-310 recibido 22-04-2005; aceptado 01-06-2005

Una Propuesta de Infraestructura para el Procesamiento del Lenguaje Natural*

Lorenza Moreno-Monteagudo y **Armando Suárez Cueto**
Departamento de Lenguajes y Sistemas Informáticos.
Universidad de Alicante
Carretera San Vicente del Raspeig s/n
03690 San Vicente del Raspeig (Alicante)
{loren, armando}@dlsi.ua.es

Resumen: La generación de recursos dentro de un grupo de investigación se ve fuertemente influenciada por la movilidad del personal eventual, por la propia evolución profesional del personal fijo, así como por la constante revisión de las técnicas y materiales necesarios. El resultado es un conjunto de herramientas y datos poco cohesionados que suponen un esfuerzo extra a la hora de establecer un marco de prototipado. En este sentido hay ya varias propuestas de entornos de integración y evaluación de herramientas pero con una curva de aprendizaje a veces notable, al tiempo que poco flexible en determinadas situaciones. Este artículo presenta una propuesta de organización de los recursos ajenos y propios de un grupo de investigación que permite, con una muy fácil implementación, un ahorro en tiempo y costes de programación, y una gestión eficiente mediante el uso transparente de las capacidades actuales de las redes cliente-servidor.
Palabras clave: Entornos PLN, prototipado, anotación de corpus

Abstract: The generation of resources within a research group is strongly influenced by the features of the staff, such as its professional evolution or its own temporariness, as well as the constant revision of techniques and materials. The result is a set of tools and data poorly coordinated that means an extra effort when establishing a prototyping frame. In this sense, there are already several Text Engineering Architectures for integration and evaluation of tools. However, they are hard to understand and use in some cases, and suffer from a lack of flexibility in others. This article presents a proposal of organizing own and other people's resources in a research group that allows, with a very easy implementation, low costs in time and programming, besides an efficient management, by means of using the current capacities of the client-server networks.
Keywords: NLP environments, prototyping, corpus annotation

1. Introducción

La evolución en el tiempo de nuestro grupo de investigación (Procesamiento del Lenguaje y Sistemas de Información, GPLSI[1]), tanto en la adscripción permanente y temporal de nuevo personal como en objetivos y proyectos ya finalizados (incluidos los muchos trabajos de Tesis Doctoral), nos ha llevado a una situación no deseada fruto de la urgencia en unos casos y la poca planificación en otros. Esta situación se resume en que el grupo dispone de gran cantidad de herramientas, aplicaciones y recursos (tanto propios como ajenos), en diferentes lenguajes de programación y sistemas operativos, con formatos de salida y entrada particulares, de localización muchas veces difícil puesto que depende de la persona o personas que los han generado, y de reutilización compleja ya que requiere un esfuerzo adicional para integrarlos en otros desarrollos.

En este artículo pretendemos resumir el análisis realizado dentro del grupo sobre esta situación y las soluciones aportadas. Esta propuesta plantea el diseño e implementación de un sistema distribuido de herramientas para el procesamiento del lenguaje natural (PLN) basado en la sencillez y la flexibilidad, utilizando la arquitectura de red cliente-servidor, y convirtiendo esas herramientas en servicios cuya ubicación física sea transparen-

* Este trabajo esta subvencionado por el Ministerio de Ciencia y Tecnología mediante el proyecto *R2D2* de referencia TIC2003-07158-C04

[1] http://gplsi.dlsi.ua.es

te al usuario, todo ello con un coste de desarrollo relativamente bajo. Además, debe convertirse en la base de una normativa ágil y consensuada que dirija nuestro trabajo en el futuro, de forma que se facilite la comunicación y la cooperación entre los participantes de futuros proyectos de investigación que requieran nuevos componentes software.

Dentro del marco de las arquitecturas denominadas como *Software Architecture for Language Engineering* (SALE) (Cunningham, 2000), podemos identificar dos aproximaciones: aquellas centradas en la anotación de textos y creación de repositorios, dentro de la que podríamos mencionar ATLAS (Bird et al., 2000) y EMU (Cassidy y Harrington, 2001), y las enfocadas hacia el desarrollo de componentes y su integración para la creación de sistemas PLN como, por ejemplo, GATE (Cunningham et al., 2002), o TEXTRACT (Neff, Byrd, y Boguraev, 2004).

La arquitectura que estamos desarrollando estaría encuadrada dentro de éstas últimas pero con ciertos matices, más dirigida a la integración de componentes ya existentes de distinta índole, lo que de todas formas facilita la posterior construcción de nuevos sistemas PLN. Una arquitectura de este tipo debe ofrecer, entre otras, las siguientes características:

1. Uso de múltiples herramientas o técnicas PLN, bien de forma individual o conjunta.
2. Generación automática de corpus lingüísticos con texto enriquecido por la anotación mediante esas herramientas, como base para otras nuevas que necesitan esta información.
3. Intercambio de herramientas del mismo tipo, dentro de un mismo sistema, para la evaluación y comparación.
4. Flexibilidad y escalabilidad, con una fácil integración de nuevos componentes PLN.
5. Transparencia en cuanto a lenguajes de programación, sistemas operativos usados y máquinas donde están instaladas esas herramientas.

El resto de este documento está organizado de la siguiente forma: la sección 2 hace un estudio de algunos de los sistemas actualmente disponibles; en la sección 3 se describe la propuesta de nuestro sistema, detallando sus aspectos básicos; finalmente, la sección 4 muestra las conclusiones de este trabajo.

2. Algunos sistemas disponibles

Antes de describir nuestro sistema, comentaremos algunas de las arquitecturas existentes en las que se ha basado nuestro trabajo.

- ATLAS *Architecture and tools for linguistic analysis systems*

  El objetivo de ATLAS es proporcionar una arquitectura general para anotación, que permita el desarrollo de nuevas aplicaciones de este tipo, y que incluya un formato determinado de los datos, un API[2], un conjunto de herramientas y una representación persistente de los datos, mediante un repositorio que se va enriqueciendo con las distintas anotaciones realizadas.

  Está centrado también en facilitar el uso e intercambio de recursos lingüísticos, así como el desarrollo de nuevos recursos sin imponer un formato fijo a los mismos, ofreciendo a nivel físico la implementación de almacenes de datos, tanto en formato ATLAS como en XML[3], que pueden ser usados en cliente-servidor, en forma de librerías, etc.

- GATE *General Architecture for Text Engineering*

  Se trata de una arquitectura que permite construir sistemas PLN a partir de componentes previamente desarrollados. Proporciona un núcleo implementado en Java y un entorno de desarrollo, a través del cual se pueden ejecutar módulos ya existentes o crear nuevas aplicaciones a partir de los mismos.

  Ofrece un conjunto inicial de herramientas básicas y además permite desarrollar nuevos recursos. La integración de componentes es inmediata en el caso de Java, aunque requiere desarrollo adicional en el caso de otros lenguajes de programación, ya que todo recurso se entiende como una clase Java.

[2] Application Program Interface
[3] Extensible Markup Language

- WHITEBOARD (Crysmann et al., 2002)

  Esta arquitectura se centra en la anotación de textos, y pretende integrar distintos componentes a través de capas XML. En lugar de tener un único documento asociado al texto proporciona diferentes capas. Cada capa almacenaría información de un determinado tipo sobre el documento, dependiendo del componente o herramienta ejecutada, en vez de estar todo incluido en un único XML. Ofrece un interfaz genérico orientado a objetos para acceder al sistema. La incorporación de un nuevo componente se lleva a cabo implementando el interfaz y especificando la DTD[4] o las reglas de transformación adecuadas.

- NLTK: *Natural Language Programming Toolkit for Python* (Loper y Bird, 2002)

  Es un sistema o conjunto de herramientas, con un marcado enfoque pedagógico que permite construir sistemas PLN usando Python. Proporciona herramientas básicas para la manipulación de datos y para la realización de tareas PLN. Permite implementar nuevos componentes combinando los ya existentes mediante unos interfaces ya definidos, que deben implementarse según las necesidades de cada nueva clase.

  Desde el punto de vista didáctico NLTK se estructura en tres bloques: ejercicios, demostraciones y proyectos, todo ello pensado para ser utilizado en los cursos de doctorado de manera que el alumno pueda realizar los ejercicios propuestos y acometer la realización de nuevos proyectos: implementación de nuevos algoritmos, desarrollo de nuevos componentes, etc.

- TEXTRACT *The TALENT*[5] *System*

  Es similar a GATE, pero pensado para el procesamiento a gran escala de textos y con un enfoque industrial (IBM). Permite la creación y evaluación de sistemas PLN, sobre todo en el contexto de aprendizaje automático. Está desarrollado en C++, y ofrece una serie de componentes (*plugins*) con las funciones básicas de PLN (*tokenizer*, *PoSTagger*, etc), al igual que GATE y NLTK. Estos plugins comparten un repositorio, asociado a cada documento, a través del cual se comunican. Se configura desde línea de comandos aunque también proporciona una GUI[6], si bien no tiene un peso tan importante como en el caso de GATE.

- EULIA (Artola et al., 2004)

  Se trata de una herramienta que facilita la edición de los corpus anotados por diversas herramientas de PLN. Se basa en una arquitectura cliente-servidor, que utiliza RMI[7] para la comunicación entre ambas partes, y que si bien en un principio hizo uso de SGML (Artola et al., 2002) como lenguaje de intercambio entre herramientas, actualmente utiliza XML. De forma similar a WHITEBOARD, basa la integración en el uso de distintas capas XML. Así, la salida de cada proceso de análisis sobre un texto es, en realidad, un conjunto de documentos XML. La especificación del XML usado, se basa en el uso de *features structures* (Jacobson, 1949).

  En cuanto a la arquitectura elegida es, quizás, la más parecida en objetivos y metodología a nuestro trabajo, si bien nosotros nos hemos centrado más en el problema de la integración de herramientas no diseñadas específicamente para este tipo de entornos.

Estos no son los únicos sistemas desarrollados, si bien son una muestra representativa de las tendencias que siguen este tipo de arquitecturas.

3. Una arquitectura cliente-servidor para PLN

La idea de destinar esfuerzos y recursos a implementar este tipo de arquitectura surge después de observar ciertas necesidades existentes dentro del grupo de investigación. Se han ido desarrollando distintas herramientas PLN, resultantes de proyectos de investigación concretos o de las propias tesis doctorales. El problema que presentan estas herramientas es que han sido implementadas con características muy específicas. Se han utilizado distintos lenguajes de programación, formatos particulares de entrada y salida, y,

[4] Document Type Definition

[5] Text Analysis and Language Engineering Tools, proyecto desarrollado en IBM

[6] Graphical User Interface

[7] Remote Method Invocation

en muchos casos, no son multiplataforma. Todo ello hace difícil su reutilización en la construcción de nuevas herramientas implicando un esfuerzo extra en formatear, adaptar o procesar la salida concreta de cada componente. Otro problema derivado de este desarrollo y uso *ad hoc* es que las herramientas y los datos no siempre están inmediatamente disponibles y localizados, lo cual dificulta todavía más su uso. Además se debe tener en cuenta la incorporación de nuevo personal contratado por la ejecución de proyectos de investigación, siendo necesario formarles en un primer momento, al tiempo que se pretende que aprovechen al máximo los recursos y el tiempo disponibles.

Algunas de las aplicaciones y recursos a los que nos estamos refiriendo son:

- Aplicaciones: implementación de máxima entropía para distintos usos, WSD (basado en máxima entropía, dominios, patrones, marcas de especificidad...), analizadores sintácticos (SUPAR), reconocedores de entidades (NERUA), recuperación de información, segmentador de oraciones, detector del idioma, *PoSTaggers*, así como diversas herramientas ajenas del mismo tipo que las mencionadas.
- Recursos: lexicones, tesauros, ontologías, corpus, etc. y la infraestructura hardware disponibles para el trabajo del grupo.

3.1. Diseño global

Tomando como partida las herramientas descritas en la sección 2, en su momento se decidió abordar los problemas anteriormente planteados de forma gradual. Concretamente se habló de tres pasos: *banco de pruebas, librerías y arquitectura PLN*, descritos a continuación.

3.1.1. Banco de pruebas

El banco de pruebas consiste en una página web que proporciona el acceso a las distintas herramientas y recursos que son de interés para el grupo. Se trata de una página que permite seleccionar la herramienta a utilizar, introduciendo un texto por pantalla o dándole un fichero a procesar. Permite además guardar la salida resultante del proceso. Con esto se pretende facilitar el acceso a las herramientas y recursos existentes e identificar problemas a la hora de integrar las distintas herramientas (formatos de entrada y salida, nombres de ficheros no apropiados, etc.)

3.1.2. Librerías

La implementación de librerías software nos permite utilizar las distintas herramientas no solo desde la web diseñada como banco de pruebas sino también desde nuestro propio código. El problema que presenta este enfoque es que se debe decidir en qué lenguajes se desarrollan las librerías, ya que, como se ha comentado con anterioridad, se parte de una situación en la que se utilizan distintos lenguajes de programación.

Actualmente existe una librería en C, otra en C++ y una clase Java, que permiten ejecutar las mismas herramientas o servicios que están disponibles en el banco de pruebas (si bien ambas cosas son independientes ya que se podrían tener servicios disponibles desde librerías que no lo estuvieran desde el banco de pruebas)

Se están estudiando también otras alternativas para llevar a cabo este paso, de forma que sea más sencillo utilizar las herramientas desde un lenguaje de programación u otro, y no haya que realizar una librería específica para cada uno. En este sentido se está considerando la posibilidad de encapsular la funcionalidad del banco de pruebas como *servicio web*[8]. Así, las herramientas podrían ser utilizadas desde lenguajes de programación que soporten este tipo de estándares y protocolos, sin necesitad de un desarrollo adicional, entendiendo por desarrollo adicional la implementación de una librería para un lenguaje concreto.

3.1.3. Arquitectura del sistema PLN

Nuestro sistema ha de favorecer la integración de diferentes componentes, con el objetivo final de construir y evaluar sistemas PLN basados en los mismos. Esto pasa por definir la arquitectura a utilizar y establecer algún mecanismo de estandarización de entradas y salidas de los distintos componentes de forma que éstos sean fácilmente integrables, ya sea de forma secuencial o paralela. En nuestro caso se ha optado por una arquitectura cliente-servidor, y por el uso de XML para la estandarización de entradas y salidas.

La arquitectura cliente-servidor está muy extendida ya que permite disponer de distintos componentes distribuidos en varias

[8] Protocolos y estándares que definen la interfaz a un sistema vía http

máquinas, desarrollados para diferentes plataformas y con distintos lenguajes de programación. A esta arquitectura básica cliente-servidor se le añade una capa, denominada *middleware*, que se encargaría de aportar transparencia al sistema, proporcionando una interfaz de comunicación entre el cliente y el servidor, localizando los recursos o servicios disponibles y facilitando la incorporación de nuevos servicios, así como el mantenimiento del propio sistema. De esta forma, si bien cada herramienta o servicio puede estar localizado en una máquina diferente, el usuario no necesita saber la localización exacta, aunque sí un nombre o identificador del servicio.

Evidentemente esta arquitectura ha de cumplir con una serie de requisitos, como son una correcta distribución de los servicios, una definición del interfaz a utilizar para el intercambio de información, o flexibilidad a la hora de incorporar nuevos componentes. Una arquitectura cliente-servidor admite distintas implementaciones: *sockets*[9], *RPC*[10], *Java RMI*, etc. En nuestro caso hemos decididdido utilizar la comunicación vía *sockets*, ya que la mayoría de los lenguajes de programación soportan este tipo de primitivas (en otros casos, como por ejemplo RMI, las primitivas de comunicación son específicas de un lenguaje determinado). Se ha decidido utilizar Java para la implementación por razones de portabilidad y sencillez. En los siguientes apartados se explican las partes básicas de nuestro sistema cliente-servidor.

1. **Directorio de servicios**. Se trata de una base de datos donde se especifican los servicios disponibles, así como la máquina donde reside el servicio y el puerto por el que atienden peticiones.

2. **Servidor *middleware***. O localizador de servicios, recibe las peticiones (solicitud de un servicio determinado) de un cliente a través de la lógica *middleware* y, en función del servicio requerido y tras consultar el directorio de servicios, devuelve los datos necesarios para conectar con el servicio concreto. Estos datos concretos son la dirección de la máquina donde se encuentra el servicio, y el puerto de comunicaciones por donde se atenderá la petición de dicho servicio.

3. ***Listener* de herramientas.** Se ha implementado un programa *listener*, que se ejecuta en tantas instancias como herramientas haya. Lo que hace este programa es "escuchar" en una máquina y puerto determinado y ejecutar la herramienta solicitada. Cuando se intenta incorporar un componente nuevo no hay que generar nuevo código, sino que se da de alta el servicio en el directorio, asignándole la máquina y el puerto correspondiente, y se lanza un *listener* "escuchando" en esa máquina y ese puerto.

   Actualmente todas las herramientas trabajan de la misma manera: reciben un fichero que procesan de alguna forma, y cuyo resultado es otro fichero que es devuelto al cliente.

4. **Lógica *middleware*.** Es la encargada de realizar las peticiones al servidor *middleware*. Debe conocer las características de este servidor (puerto y máquina donde se encuentra), así como la identificación del servicio requerido. Realiza la conexión con el servidor *middleware* para obtener los datos del servicio que quiere ejecutar (máquina y puerto donde se encuentra dicho servicio o herramienta). Con estos datos establece la conexión con el *listener* correspondiente. Le envía el fichero a procesar, el identificador del servicio, los parámetros que necesite la herramienta y recibe el fichero ya procesado.

Puede verse un esquema del funcionamiento de esta arquitectura en la figura 1.

Esta arquitectura es la que se ha utilizado tanto en la implementación del banco de pruebas como en la implementación de las librerías.

Actualmente cuando se ejecuta un servicio, la salida, el fichero procesado que nos devuelve, tiene el formato propio de la herramienta. Esto sigue imposibilitando, lógicamente, la integración entre las distintas herramientas en la construcción de un sistema PLN final. De ahí que otro punto a tener en cuenta en nuestro sistema sea la estandarización de entradas y salidas.

Estandarización de entradas y salidas. Es necesario normalizar las entradas y salidas de nuestras herramientas, para así aportar facilidad de uso e integración a las

[9]Comunicación a bajo nivel entre dos procesos
[10]Remote Procedure Call

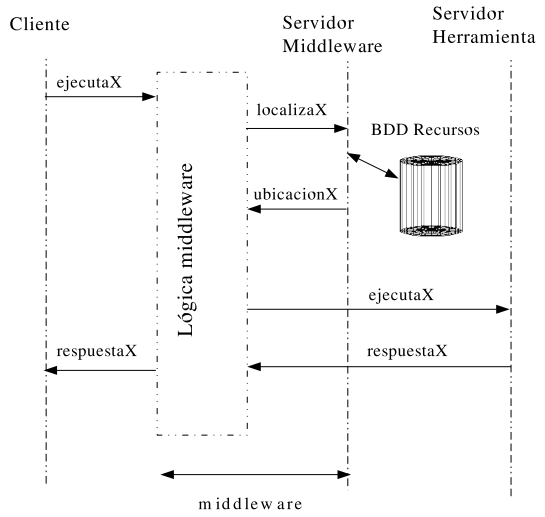

Figura 1: Ejecucion del servicio X

mismas, además de validar estructuralmente la información que estamos manejando. Pensamos que XML es la mejor alternativa actualmente.

XML es un lenguaje de marcado que permite estructurar la información de acuerdo a las definiciones que se especifiquen y, al mismo tiempo, validar la propia información, es decir, comprobar que ésta tiene la "forma" correcta. La DTD, o el *XML Schema*[11] es el conjunto de reglas que definirán la forma en que se van a construir estos documentos XML.[12]

A la hora de diseñar este *XML Schema* debe tenerse en cuenta nuestro objetivo final, un entorno para la construcción y evaluación de sistemas PLN. En este sentido se ha intentado diseñar un *XML Schema* que fuera lo suficientemente amplio para abarcar toda la información proporcionada por las distintas herramientas. Al mismo tiempo debe ser flexible, que no requiera excesivas modificaciones cuando se incorpore una nueva herramienta al sistema. La estructura de los documentos XML debería permitir discriminar la información resultante de una herramienta en favor de otra de las mismas características. Por ejemplo, suponiendo que queremos construir un sistema PLN que utiliza un analizador sintáctico, debemos poder evaluar y comparar dicho sistema usando varios programas alternativos. También se debería expresar en el XML toda la información proporcionada por cada una de las herramientas.

Respecto a cómo representar esta información, se ha intentado definir unas etiquetas más o menos genéricas y a través de atributos o elementos, discernir entre unas salidas y otras. Esto en principio simplifica bastante el diseño y proporciona la flexibilidad suficiente para poder seguir incorporando herramientas e información al sistema de forma sencilla.

Incorporación de recursos. Otro punto que hay que tener en cuenta es el acceso a los distintos recursos, entendiendo por recurso todo aquello que pueda necesitar un proceso: *gazzeteers*, diccionarios, corpus, etc. La arquitectura debe permitir el acceso a los recursos de forma sencilla.

En un primer paso bastaría con tener localizados dichos recursos, del mismo modo que se tienen localizadas las herramientas, de forma que se supiera exactamente qué recursos hay, y que fueran fácilmente accesibles.

En un siguiente paso se debería no solo conocer qué recursos hay sino poder incorporarlos a la construcción de un sistema concreto. Es decir, si estamos construyendo un sistema que, por ejemplo, utilice aprendizaje basado en corpus, deberíamos poder incorporar dicho corpus a la construcción de nuestro sistema de forma semiautomática. Esto implica conocer de qué forma utilizará el sistema dicho corpus, de qué forma interactúan ambos, cuál es la información concreta que se necesita, qué estructura tiene, etc.

Publicación y consulta de servicios. La arquitectura debe permitir la fácil incorporación de nuevos servicios o herramientas así como la consulta rápida de los mismos.

Consultar los servicios disponibles simplemente consiste en acceder a la base de datos de servicios, por ejemplo a través de una página web, de forma que se muestren las características principales del servicio: identificador, descripción del mismo, parámetros que

[11] Puede considerarse como la evolución de las DTD y proporciona mayor potencia a la hora de definir la estructura de nuestros XML. http://www.3w.org

[12] El punto de partida de la definición de nuestro *XML Schema* es la DTD desarrollada en el proyecto *3LB* (Construcción de una base de datos de árboles sintáctico semánticos, http://www.dlsi.ua.es/projectes/3lb/), mejorando su estructura mediante la adición de capas y nuevas definiciones.

necesita para ser utilizado, etc. En resumen, proporcionar al usuario la información necesaria para que pueda utilizar este servicio.

La publicación de nuevos servicios, se refiere al alta de los mismos de forma más o menos automática. Por un lado cualquier usuario podría realizar la solicitud de un alta de un servicio, por ejemplo a través de una página web, aportando la información acerca de qué hace dicho servicio y, por supuesto, el ejecutable o ejecutables involucrados. A partir de ahí, una persona concreta (podemos pensar en la figura de un administrador) estudiaría la viabilidad de dicha alta, aportando el resto de información necesaria: en qué máquina va a estar el servicio, en qué puerto va a escuchar, etc.

Encadenamiento de herramientas. En la construcción de nuevos sistemas PLN, se pretende utilizar componentes ya desarrollados e incorporados a la arquitectura, de forma encadenada.

Parte del problema de encadenar estos componentes está solucionado con la utilización de XML para formatear las entradas y salidas de los distintos componentes. Aparte, se debe decidir de qué forma se realiza este encadenamiento. Por lo pronto podemos pensar en dos alternativas: ejecución secuencial o paralela. Habrá casos en los que simplemente necesitemos ejecutar de forma secuencial una serie de componentes. Sin embargo, puede darse el caso en que se puedan ejecutar varios componentes en paralelo. Sirva como ejemplo de este último caso un sistema de votación que tras comparar el resultado obtenido por diversas herramientas, tuviera que devolver el mejor. Si suponemos que estas herramientas son independientes, se podrían ejecutar paralelamente, con lo cual el sistema daría la respuesta en un menor tiempo.

En cuanto al interfaz el objetivo es proporcionar un diseño sencillo que permita ir añadiendo estos componentes y determinar, además, el orden en que deben ser ejecutados. Se trataría de un interfaz donde se representen y encadenen gráficamente los distintos componentes.

4. Conclusiones

Este artículo presenta una propuesta de gestión e integración de los recursos software de los que dispone un grupo de investigación que, habitualmente, provienen de fuentes tanto ajenas como propias. En organizaciones como la nuestra, donde la cohesión de un grupo de personas viene dado casi siempre por afinidades no regladas, la tendencia es generar programación y datos muy personalizados, dirigidos a la resolución de un problema actual, a menudo controlados por un único investigador, y con poca previsión de reutilización en nuevos proyectos o por personas distintas. Es normal que, dentro del conjunto de todas las aportaciones, podamos encontrar programación no estándar en muy diferentes lenguajes, formatos de entrada y salida, librerías, mecanismos de almacenamiento de datos y sistemas operativos. Por otro lado, las constantes revisiones de los métodos y técnicas, las distintas versiones del software básico, también generan problemas a la hora de afrontar un nuevo proyecto.

Examinando las diferentes propuestas que se encuentran disponibles actualmente, nos encontramos con algunos entornos software, dentro del campo del PLN, con un especial hincapié en los métodos de almacenamiento y representación, que integran herramientas de tratamiento de textos y de evaluación. Estos productos son, por un lado, muy completos pero, por otro, excesivamente complejos y adolecen de falta de flexibilidad en ciertas situaciones.

Nuestra propuesta se centra en dos objetivos básicos: transparencia para el usuario y escalabilidad. Apostamos por una arquitectura cliente-servidor donde cada programa se convierte en un servicio independientemente del lenguaje o del sistema operativo original de cada uno. El sistema puede ser totalmente abierto, cualquiera puede publicar un servicio, o a través de un administrador del sistema, pero siempre atendiendo a la facilidad de uso e inmediatez. Obviamente, todo esto pasa por una estandarización de las entradas y salidas de los programas que puede estar integrada en el propio código, pero que puede ser realizada *a posteriori* con las adecuadas herramientas de traducción.

Parte de lo que se ha descrito en este artículo está ya en funcionamiento en nuestro grupo de investigación o en fases de desarrollo muy avanzadas. Actualmente, estamos mejorando la definición del estándar de formatos de entrada y salida, y trabajando en la implementación plenamente funcional del servidor *middleware* que gestione los recursos en red. Nos hemos propuesto ofrecer, además, un en-

torno gráfico que permita combinar todos los recursos en la red y construir prototipos de forma visual como un paso más para facilitar su uso.

Nuestra intención es obtener un sistema de integración de herramientas y datos, enfocado hacia PLN pero abierto a otros campos de aplicación, que tenga una fácil puesta en marcha, no excluyente frente a otros productos similares, y que pueda ser de utilidad para el resto de investigadores como alternativa válida y abierta.

5. Agradecimientos

Queremos agradecer a Arantza Díaz de Ilarraza, del grupo IXA, su ayuda al proporcionarnos parte del material utilizado en este artículo.

Bibliografía

Artola, X., A. Díaz de Ilarraza, N. Ezeiza, K. Gojenola, A. Maritxalar, y A. Sorosa. 2002. A proposal for the integration of NLP tools using sgml-tagged documents. En *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas, Canary Islands, Spain, May.

Artola, X., A. Díaz de Ilarraza, N. Ezeiza, K. Gojenola, A. Sologaistoa, y A. Sorosa. 2004. EULIA: a graphical web interface for creating, browsing and editing liguistically annotated corpora. En *Proceedings of the LREC 2004 Workshop on XML-based Richly Annotated Corpora*, Lisbon, Portugal, May.

Bird, S., D. Day, J. Garofolo, J. Henderson, C. Laprun, y M. Liberman. 2000. ATLAS: A Flexible and Extensible Architecture for Linguistic Annotation. En *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000)*, páginas 1699–1706, Athens, Greek, May.

Cassidy, S. y J. Harrington. 2001. Multi-level annotation in the EMU speech database management system. *Speech Communication*, 33:61–77, January.

Crysmann, B., A. Frank, B. Kiefer, S. Müller, G.Ñeumann, J. Piskorski, U. Schäfer, M. Siegel, H. Uszkoreit, F. Xu, M. Becker, y H.U. Krieger. 2002. An integrated architecture for shallow and deep processing. En *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, páginas 441–448, Philadelphia, USA, July.

Cunningham, H., D. Maynard, K. Bontcheva, y V. Tablan. 2002. GATE: an architecture for development of robust hlt applications. En *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, USA, July.

Cunningham, Hamish. 2000. *Software Architecture for Language Engineering*. Ph.D. tesis, University of Sheffield, June.

Jacobson, Roman. 1949. The identification of phonemic entities. *Travaux du Cercle Linguistique de Copenhague*, 5:205–213.

Loper, Edward y Steven Bird. 2002. NLTK: The Natural Language Toolkit. En *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, páginas 62–69, Philadelphia, USA.

Neff, Mary S., Roy J. Byrd, y Branimir K. Boguraev. 2004. The TALENT System: TEXTRACT architecture and data model. *Natural Language Engineering*, 10:307–326.