ARQUITECTURA DISTRIBUIDA PARA UN SISTEMA DE TRADUCCIÓN DEL HABLA SOBRE LA PLATAFORMA UIMA

Marc Poch[†]*, David Cuestas*[†]*, José B. Mariño*[†]*, Francisco Méndez*[‡]*, Iñaki Sainz*[††]

[†]Centro de Investigacion TALP (UPC), *(mpoch, dcuestas, canton)@gps.tsc.upc.edu*
[‡]Grupo de Teoría de la Señal (Univ. Vigo), *fmendez@gts.tsc.uvigo.es*
[††]Grupo de Investigación Aholab (EHU), *inaki@bips.bi.ehu.es*

RESUMEN

En el presente trabajo se presenta una solución para la integración de las distintas tecnologías que intervienen en un sistema de traducción del habla. En primer lugar se introduce el software en el que se basa la solución adoptada. A continuación se describe la arquitectura diseñada en su versión local y distribuida entre sedes. Más adelante se explica como se han resuelto los problemas de intercambio de información y sincronismo entre la voz original y la sintetizada.

1. INTRODUCCIÓN

AVIVAVOZ[1] es un proyecto coordinado dirigido a la investigación en todas las tecnologías clave que intervienen en un sistema de traducción de voz (reconocimiento, traducción y síntesis de voz). El objetivo del proyecto es lograr avances en todos los componentes de un sistema de traducción de voz para alcanzar sistemas de intermediación oral entre personas en las lenguas oficiales del estado español (castellano, catalán, euskera y gallego) entre sí y entre el castellano y el inglés. El proyecto AVIVAVOZ ha sido propuesto por un consorcio formado por tres equipos investigadores: el grupo de Procesado de Voz del "Centro de Investigación de Tecnologías y Aplicaciones de la Tecnología del Lenguaje y el Habla" (TALP) de la Universidad Politécnica de Cataluña, el Grupo de Teoría de la Señal (GTS) del Departamento TSC de la Universidad de Vigo y el Grupo de Investigación Aholab (AHOLab) de la Universidad del País Vasco. Una de las actividades del proyecto está dedicada a la integración de los sistemas de reconocimiento, traducción y síntesis a fin de construir el sistema completo de traducción del habla. Uno de los intereses del proyecto es construir un sistema completo de traducción del habla utilizando tecnologías residentes en las distintas sedes, de modo que se pudiesen compartir módulos y se permitiese configurar diversas arquitecturas por cada sede según los intereses de la investigación. Por ello se ha optado por la realización de una arquitectura web para la comunicación entre los diversos subsistemas y sedes. Basándose en la experiencia del proyecto TC-STAR[2], se ha elegido la plataforma UIMA de IBM como solución práctica. Esta comunicación está organizada como sigue. En primer lugar se proporciona una breve descripción de la plataforma UIMA. A continuación se describe la arquitectura diseñada y su realización, que incluye una arquitectura local y otra distribuida, se explica el formato empleado para el flujo de información y se trata brevemente el problema del sincronismo entre el habla original y el habla producida por el sistema de traducción. El trabajo se cierra con las conclusiones.

Este trabajo ha sido subvencionado por el Gobierno Español mediante el proyecto coordinado AVIVAVOZ (TEC2006-13694-C03) y el Govern de la Generalitat de Catalunya mediante el proyecto TecnoParla.

[1] AVIVAVOZ. Tecnologías para la traducción de voz: reconocimiento, traducción estadística basada en corpus y síntesis (TEC2006-13694-C03-01/TCM) *http://www.avivavoz.es*

2. UIMA

UIMA (Unstructured Information Management Architecture) [1] es una plataforma de software extensible y escalable que permite ordenar y procesar datos no estructurados como texto, sonido, imagen, etc. El sistema fue inicialmente desarrollado por Alpha Works IBM aunque ahora es Apache Software Foundation quien se encarga del mantenimiento, de las licencias y de las nuevas versiones. UIMA es un software abierto programado en lenguaje Java que permite al desarrollador trabajar en este mismo lenguaje o en C++. La arquitectura UIMA se basa en el desarrollo de módulos independientes de proceso de datos. Estos se pueden enlazar de forma que un módulo recibe información del módulo anterior y transmite nueva información al módulo siguiente. Gracias a las herramientas de que dispone la plataforma UIMA se pueden utilizar arquitecturas distribuidas. Esto significa que cada uno de los módulos de proceso puede estar localizado en una máquina distinta y ser utilizado de forma remota. La plataforma UIMA cuenta con un conjunto de herramientas [2] para facilitar el desarrollo de módulos, componentes y arquitecturas completas. El software cuenta con distintos plugins para el entorno de programación libre Eclipse. Gracias a estos plugins se pueden editar ficheros fundamentales en UIMA de forma fácil y cómoda. A parte de los plugins, UIMA cuenta con algunas aplicaciones Java

[2] TC-STAR: Technology and Corpora for Speech to Speech Translation (IST-FP6-506738) *http://www.tc-star.org*

de ayuda que, por ejemplo, permiten ejecutar una cadena de módulos de forma simple.

2.1. Arquitectura básica

La arquitectura básica UIMA consiste en enlazar de forma consecutiva un conjunto de módulos. Estos módulos de proceso de datos reciben el nombre de "Analysis Engines" (AE) [3] y son la unidad básica de desarrollo en UIMA. Todos los módulos y componentes UIMA tienen asociados un fichero xml llamado descriptor. En él se definen distintas características del AE y se usa para hacer la llamada al modulo cuando queremos ejecutarlo. De esta forma, un AE consta de dos partes diferenciadas: su descriptor xml y el codigo Java (o C++) que el desarrollador ha implementado. Este codigo o programa recibe el nombre de Anotador. Para poder ejecutar una cadena completa hace falta un programa que sea capaz de llamar de forma ordenada a los distintos AE. Este programa está incluido en UIMA y se llama CPE (Collection Processing Engine). Como todos los componentes UIMA tiene un descriptor xml asociado que nos servirá para definir qué módulos queremos llamar y en qué orden. Para que todo el sistema funcione es fundamental el flujo de información de un AE al siguiente. UIMA utiliza una estructura de datos llamada CAS (Common Analysis structure) para pasar información de un módulo al siguiente. El CAS transporta información de UIMA (parámetros, etc) y los datos que se van a procesar llamados Sofa (Subject of Analysis).

2.2. Common analysis structure (CAS)

El CAS es una estructura de datos basada en objetos que permite la representación de objetos, propiedades y valores. Además puede albergar distintos análisis de unos mismos datos de entrada. Cada análisis corresponde a un Sofa. Para garantizar el flujo de información cada AE debe conocer la estructura del CAS. El Type System define la estructura del CAS mediante un fichero xml. De esta forma el AE puede conocer el CAS al detalle interpretando el Type System. El Type System está formado por un conjunto de Types que son representaciones de objetos y sus propiedades. La representación concreta de un objeto en el CAS recibe el nombre de Annotation.

3. ARQUITECTURA DE INTEGRACIÓN

Los sistemas de traducción del habla (voz a voz) pueden dividirse en tres tareas fundamentales: reconocimiento del habla (ASR), traducción de texto (SMT) y síntesis del habla (TTS). Estas tres tareas son desarrolladas habitualmente de forma independiente e incluso por grupos de investigación diferentes. Entonces, para poder utilizar todo el sistema, traduciendo de voz a voz, es necesario diseñar una arquitectura que sea capaz de lanzar el reconocimiento, la traducción y la síntesis de forma ordenada y de manera que las tres partes se entiendan. A tal efecto, se ha diseñado una arquitectura basada en la plataforma UIMA para poder realizar traducciones del habla. Esta arquitectura permite ejecutar traducciones de voz a voz a pesar de tener sistemas de reconocimiento, traducción y síntesis totalmente independientes. La arquitectura desarrollada permite la comunicación entre reconocedor, traductor y sintetizador aunque estos utilicen tipos de datos y lenguajes distintos. Para la implementación del sistema se han creado todos los descriptores necesarios para definir los componentes y el TypeSystem. A su vez, se han desarrollado todos los anotadores en Java. A continuación se describe la arquitectura diseñada suponiendo que todos los componentes están en una misma máquina. A esta arquitectura la llamaremos arquitectura local. En cambio, cuando algunos o todos los componentes (ASR, SMT, o TTS) están en distintas máquinas se utiliza una arquitectura distribuida.

3.1. Arquitectura local

La arquitectura local supone que tenemos los sistemas implicados (ASR, SMT, TTS) en una misma máquina que también dispone del software UIMA instalado. En caso de que alguno de los sistemas ASR, SMT o TTS estuviese implementado de forma cliente servidor lo que supondremos es que el programa cliente es el que está en la máquina local (con el UIMA y el resto de sistemas) pudiendo estar el servidor en otra máquina. Con tal de simplificar, supondremos que tanto el programa cliente como el programa servidor están en la misma máquina. Tal y como muestra la figura 1, todos los programas y ficheros se encuentran en la máquina local. El "Collection Reader" es un elemento común en todas las cadenas de ejecución UIMA y permite iniciar el sistema. Se encarga de leer el fichero "Audio file URI" (Uniform Resource Identifier) que contiene una lista de las localizaciones de los ficheros que queremos procesar. Se ha diseñado esta arquitectura asignando un AE para cada tarea de reconocimiento, traducción y síntesis. Cada AE cuenta con un anotador y un descriptor xml. El anotador ejecuta el código necesario para preparar y realizar la llamada al programa cliente ASR, SMT o TTS. Además recoge los datos de salida y los incorpora al CAS según corresponda para que el siguiente AE pueda procesarlos. El "CPE Descriptor" es el fichero xml que define toda la cadena y que se puede ejecutar utilizando un script de UIMA.

3.2. Arquitectura distribuida

Para solucionar el problema de las distintas localizaciones y las distintas máquinas para el ASR, SMT y TTS se ha diseñado una arquitectura distribuida basada en las posibilidades de la plataforma UIMA. Esta arquitectura permite ejecutar traducciones de voz a voz a pesar de tener cada componente en una máquina diferente. Para describir la arquitectura supondremos que cada sistema ASR, SMT y TTS está localizado en una máquina distinta. Además, supondremos que la máquina desde la que queremos

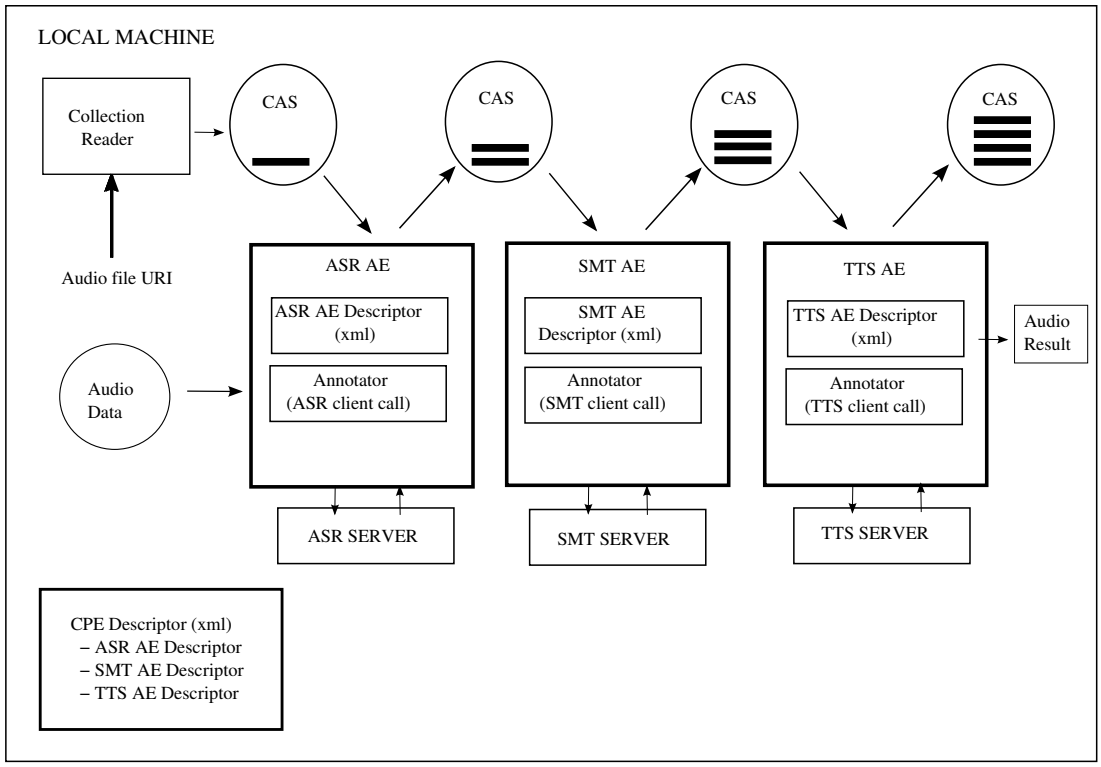

Figura 1. *Esquema de la arquitectura local.*

ejecutar la traducción es otra máquina adicional. La arquitectura distribuida requiere la utilización de un servidor central que en la suposición más general será otra máquina. Tal y como se ha descrito, y como muestra la figura 2, el caso más general de implementación de la arquitectura distribuida consta de 5 máquinas distintas. Para que el sistema funcione todas las máquinas deben tener el software UIMA funcionando. La idea de la arquitectura distribuida consiste en que se puedan ejecutar cadenas completas de forma transparente desde la máquina local que no dispone de ninguno de los servidores ASR, SMT o TTS. Para la arquitectura distribuida no es suficiente con tener los AE listos en cada máquina, hay que dar de alta cada AE como servicio en el servidor central. Éste utiliza un servidor VNS (Vinci Name Service) para gestionar una lista de los servicios. Cada servicio define de forma unívoca a cada AE y contiene información de la máquina en la que se encuentra y los puertos que utiliza. De esta forma con el nombre de un servicio disponemos de la información necesaria para acceder al AE que lo ejecuta. Gracias al sistema VNS no es necesario tener los descriptores de cada AE en la máquina local como pasaba en la versión no distribuida. En vez de eso, se utilizan unos descriptores simples (Vinci Client Descriptor) que sólo contienen el nombre del servicio al que queremos llamar. En la figura 2 aparece un componente llamado "CopyIn". Éste servicio permite copiar el fichero de entrada, situado el la carpeta "in" del servidor central, a la máquina ASR. En cambio, el servicio "CopyOut" accede a la máquina TTS para copiar el fichero de salida a la carpeta "out" del servidor central. De esta manera el usuario puede acceder al fichero de entrada y salida de la cadena completa sin tener que acceder a las máquinas que prestan los servicios.

3.3. Flujo de información

A la hora de construir plataformas de integración de software surge el problema de la comunicación entre componentes. En los apartados anteriores se explica como se ha resuelto el problema de como intercambiar información entre ASR, SMT y TTS. Una vez establecido el flujo de información hay que resolver el problemas de que datos hay que transmitir y de que manera para que todos los componentes los interpreten. Se ha decidido adoptar el estándar HTK [3] de grafos como formato de datos entre los distintos módulos. El sistema HTK cuenta con numerosas librerías y software ya desarrollado para crear e interpretar grafos. Estos grafos serán incluidos en el CAS para ser transportados al siguiente módulo. Los grafos HTK permiten representar múltiples hipótesis, incluir tiempos asociados a las hipótesis, incorporar información sobre la confianza en las hipótesis e incluir meta información.

3.4. Sincronismo

Para poder hacer una traducción de voz sincronizada con la voz original el intercambio de información entre

[3]The Hidden Markov Model Toolkit *http://htk.eng.cam.ac.uk*

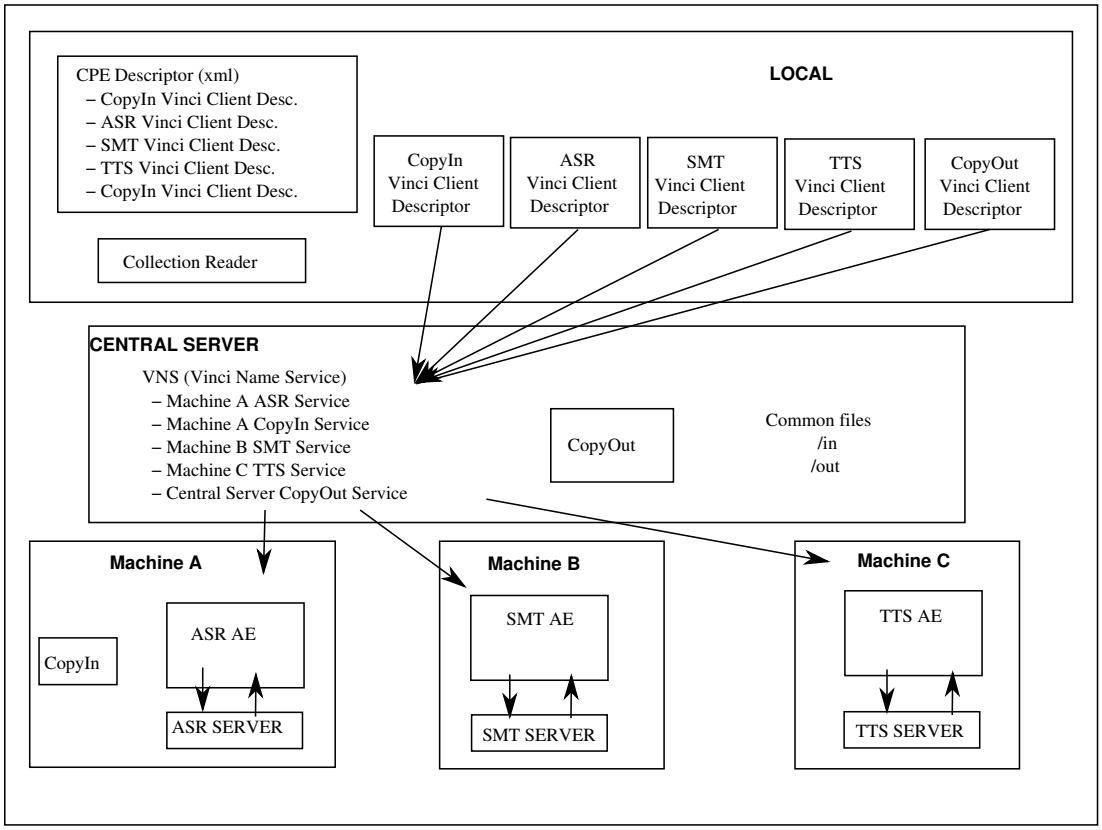

Figura 2. *Esquema de la arquitectura distribuida.*

los módulos no puede ser solo texto. El reconocedor debe ser capaz de dar información temporal asociada al texto. El traductor por su parte deberá ser capaz de establecer una relación entre el texto traducido y el texto original. Y para acabar, el sintetizador deberá ser capaz de interpretar los datos del ASR y el SMT de manera que pueda establecer una relación entre el texto traducido que debe sintetizar y los tiempos del habla original. Aparece otra vez un problema de comunicación entre programas totalmente independientes que hay que resolver. A tal efecto, la arquitectura diseñada cuenta con que el módulo de reconocimiento se encarga de incluir los datos temporales asociados a cada palabra en su grafo de salida. Por su parte, el traductor establece una relación basada en índices de palabras para relacionar una o varias palabras del texto traducido con una o varias palabras del texto original. Toda esta información será incluida en el grafo de salida del módulo SMT. Para acabar, el módulo TTS interpreta los grafos de salida del ASR y SMT y recupera la información que necesita para sincronizar la síntesis de voz.

4. CONCLUSIÓN

Las arquitecturas desarrolladas ofrecen soluciones a los principales problemas de la integración de sistemas de traducción de voz a voz independientes. Ambas arquitecturas, local y distribuida, han sido probadas con éxito. La arquitectura distribuida ha sido montada de forma que tres sedes (Universitat Politècnica de Catalunya, Universidade de Vigo y Universidad del País Vasco) prestaban alguno o todos los servicios de reconocimento, traducción y síntesis al sistema. Hay que destacar que estas arquitecturas no influyen en las prestaciones del reconocedor, del traductor y del sintetizador en cuestión pero permiten avanzar en el desarrollo de sistemas integrados de traducción de voz por su escalabilidad y su flexibilidad. Gracias a ello se puede plantear la concatenación de distintos traductores, la inclusión de nuevos módulos, etc. Para finalizar podemos decir que esta arquitectura es una valiosa herramienta para el desarrollo de las tecnologias del habla y la colaboración entre universidades.

5. BIBLIOGRAFÍA

[1] The Apache UIMA Development Community, *UIMA Overview and Setup*, The Apache Software Foundation, February 2007.

[2] The Apache UIMA Development Community, *UIMA Tools Guide and Reference*, The Apache Software Foundation, February 2007.

[3] The Apache UIMA Development Community, *UIMA Tutorial and Developers Guides*, The Apache Software Foundation, February 2007.